AMFIST: ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยรองรับรายการข้อมูลที่คล้ายคลึงกัน

AMFIST: An Algorithm for Mining Frequent Itemsets based on the Similarities of Transactions

ฟูไดละห์ ดือมอง

Fudailah Duemong

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Prince of Songkla University
2553

ชื่อวิทยานิพนธ์ AMFIST: ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน
ผู้เขียน นางสาวฟูไดละห์ ดือมอง
สาขาวิชา วิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

…….....
(ดร.ลัดดา ปรีชาวีรกุล)

อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม

…….....
(ผู้ช่วยศาสตราจารย์ ดร.ศิริรัตน์ วณิชโยบล)

คณะกรรมการสอบ

……..ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.กฤษณะ ชินสาร)

…….....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วิภาดา เวทย์ประสิทธิ์)

…….....กรรมการ
(ดร.ลัดดา ปรีชาวีรกุล)

…….....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.ศิริรัตน์ วณิชโยบล)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

……
(รองศาสตราจารย์ ดร.เกริกชัย ทองหนู)
คณบดีบัณฑิตวิทยาลัย

ชื่อวิทยานิพนธ์ AMFIST: ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยรองรับรายการข้อมูลที่คล้ายคลึงกัน
ผู้เขียน นางสาวฟูไดละห์ ดือมอง
สาขาวิชา วิทยาการคอมพิวเตอร์
ปีการศึกษา 2552

บทคัดย่อ

ขั้นตอนวิธีที่มีประสิทธิภาพสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets) เป็นสิ่งสำคัญในการสร้างกฎความสัมพันธ์ (Association Rules) ในการทำเหมืองข้อมูล (Data Mining) โดยทั่วไปแล้วการพัฒนาขั้นตอนวิธีให้มี ประสิทธิภาพก่อนหน้านี้ส่วนใหญ่จะคำนึงถึงเวลาที่ใช้ในการทำงาน ลดจำนวนของการสร้างกลุ่มข้อมูลท้าชิง (Candidate Itemsets) และลดจำนวนของการอ่านข้อมูลจากฐานข้อมูล อย่างไรก็ตามยังมีวิธีการหนึ่งที่สามารถลดระยะเวลาการทำงานได้ คือ การรวมรายการข้อมูลที่คล้ายคลึงกันในฐานข้อมูล

วิทยานิพนธ์นี้นำเสนอขั้นตอนวิธีที่มีประสิทธิภาพสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยชื่อ “AMFIST: ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน” (AMFIST: An Algorithm for Mining Frequent Itemsets based on the Similarities of Transactions) หลีกเลี่ยงการสร้างกลุ่มข้อมูลท้าชิง ลดจำนวนการอ่านข้อมูลจากฐานข้อมูลเพื่อใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเหลือเพียงครั้งเดียว ใช้การจัดเก็บข้อมูลในรูปแบบบิต และดำเนินการระดับบิตในการค้นหาค่าความถี่และสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย นอกจากนี้ใช้วิธีการรวมรายการข้อมูลที่ซ้ำกันหลังจากตัดกลุ่มข้อมูลที่ไม่ปรากฏร่วมกันบ่อย เพื่อให้การทำงานมีประสิทธิภาพและลดเนื้อที่ในการจัดเก็บข้อมูลในหน่วยความจำระหว่างการประมวลผล จากการเปรียบเทียบขั้นตอนวิธี AMFIST กับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori พบว่ามีประสิทธิภาพดีกว่าขั้นตอนวิธีทั้งสอง โดยนอกจากจะทำงานได้ดีกับฐานข้อมูลที่อัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏอยู่ในรายการข้อมูลมีค่ามากหรือน้อยแล้วยังสามารถทำงานได้ดีกับฐานข้อมูลขนาดใหญ่อีกด้วย

Thesis Title	AMFIST: An Algorithm for Mining Frequent Itemsets based on the Similarities of Transactions
Author	Miss Fudailah Duemong
Major Program	Computer Science
Academic Year	2009

ABSTRACT

Efficient algorithms for mining frequent itemsets are crucial for mining association rules in data mining tasks. Most of the previously used algorithms have generally been developed for using the computational time effectively, reducing the number of candidate itemsets and decreasing the number of scan in the database. However, the time can be reduced by aggregate transactions having similar itemsets in database.

This thesis proposes an efficient algorithm for mining frequent itemsets called AMFIST (An Algorithm for Mining Frequent Itemsets based on the Similarities of Transactions) by avoiding candidate itemsets generation, scanning database only once for mining frequent itemsets, storing items in the set of bits, using bitwise operations for finding support count and generating frequent itemsets. Also, the algorithm works efficiently and reduces memory space by aggregating the similarities of transactions after deletes infrequent itemsets. A comparison among AMFIST algorithm, Apriori algorithm and Modified-Apriori algorithm shows that AMFIST algorithm work more efficiently not only in sparse and dense database, but also in very large database.

สารบัญ

หน้า

สารบัญ (ต่อ)

รายการตาราง

รายการภาพประกอบ

ภาพประกอบ **หน้า**

รายการภาพประกอบ (ต่อ)

รายการภาพประกอบ (ต่อ)

ภาพประกอบ **หน้า**

รายการภาพประกอบ (ต่อ)

ภาพประกอบ **หน้า**

บทที่ 1

บทนำ

ในยุคสมัยปัจจุบัน ความก้าวหน้าทางเทคโนโลยีสามารถนำมาประยุกต์ใช้ให้เกิดประโยชน์ได้อย่างกว้างขวาง ทั้งทางด้านการแพทย์ การศึกษา อุตสาหกรรม คมนาคม หรืออื่นๆ เพื่อเพิ่มประสิทธิภาพในงานนั้นๆ จึงเกิดการพัฒนาโดยสร้างอุปกรณ์ เครื่องมือ เครื่องจักร หรือแม้กระทั่งอาจเป็นสิ่งที่จับต้องไม่ได้ เช่น ระบบหรือกระบวนการต่างๆ รวมไปถึงการค้นหาความรู้ในฐานข้อมูล (Knowledge Discovery in Databases: KDD) ก็เช่นกัน ซึ่งเป็นแนวคิดที่ใช้ในการค้นหารูปแบบของข้อมูลที่เป็นประโยชน์ซึ่งซ่อนอยู่ในฐานข้อมูล (Willen and Frank, 2005) เป็นความก้าวหน้าทางเทคโนโลยีอีกรูปแบบหนึ่ง ที่มีการพัฒนาขึ้นมาเพื่อให้ทราบถึงรูปแบบ หรือแนวโน้มของข้อมูลเหล่านั้นไปใช้ประโยชน์ต่อไป

ขั้นตอนของการทำเหมืองข้อมูล (Data Mining) เป็นส่วนหนึ่งที่สำคัญเป็นอย่างมากในการค้นหาความรู้ในฐานข้อมูล (Mitra and Acharya, 2003) เนื่องจากเป็นขั้นตอนของการเลือกเทคนิค(Techniques) หรือขั้นตอนวิธี (Algorithm) ที่จะใช้ในการค้นหาความรู้ในฐานข้อมูล ซึ่งการเลือกขั้นตอนวิธีนั้นจะต้องขึ้นอยู่กับปัจจัยหลายๆ อย่างเช่น การเลือกขั้นตอนวิธีที่เหมาะสมกับทรัพยากรที่เรามีอยู่ ลักษณะของข้อมูลหรือขนาดของฐานข้อมูล หรือแม้กระทั่งรูปแบบของผลลัพธ์ที่ต้องการจากฐานข้อมูลเหล่านั้น เพื่อให้ได้ผลลัพธ์ตรงตามความต้องการและสามารถทำงานหรือประมวลผลได้อย่างรวดเร็ว ดังวิทยานิพนธ์นี้ได้นำเสนอขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets Algorithm) ซึ่งเป็นขั้นตอนหนึ่งในการทำเหมืองข้อมูลสำหรับการค้นหากฎความสัมพันธ์ (Association Rules) ของข้อมูลในฐานข้อมูล

ปัจจุบันได้มีการพัฒนาขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพื่อใช้สำหรับการสร้างกฎความสัมพันธ์ขึ้นมาอย่างมากมาย ซึ่งแต่ละขั้นตอนวิธีนั้นมีการพัฒนาเพื่อให้มีประสิทธิภาพในการทำงานมากยิ่งขึ้น โดยเหตุผลที่ใช้ในการสร้างขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้น แต่ละขั้นตอนวิธีอาจคำนึงถึงการลดเนื้อที่หน่วยความจำที่ใช้ในการจัดเก็บข้อมูลในระหว่างการทำงาน การลดระยะเวลาในการทำงานให้มีความรวดเร็ว การสร้างขั้นตอนวิธีที่เหมาะสมกับรูปแบบของผลลัพธ์ที่ต้องการจากฐานข้อมูล หรือการสร้างขั้นตอนวิธีที่เหมาะสมกับลักษณะของข้อมูลในฐานข้อมูล

วิทยานิพนธ์นี้ได้นำเสนอขั้นตอนวิธีในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย สำหรับฐานข้อมูลที่มีจำนวนของรายการข้อมูล (Transactions) ที่คล้ายคลึงกันหลาย ๆ

รายการข้อมูลซึ่งปรากฏในฐานข้อมูลเดียวกัน ยกตัวอย่างเช่น ฐานข้อมูลการขายสินค้าแบบค้าส่งที่มีการสั่งซื้อสินค้าที่คล้ายหรือเหมือนกันของลูกค้าแต่ละรายในแต่ละครั้งของการสั่งซื้อสินค้า โดยบางครั้งการสั่งซื้อสินค้าของลูกค้าแต่ละรายก็อาจจะมีการสั่งซื้อสินค้าที่ซ้ำกันได้ในแต่ละครั้ง หรือการจ่ายยาของแพทย์ในโรงพยาบาลที่มีการจ่ายยาตามลักษณะโรคของผู้ป่วย ซึ่งหากเป็นช่วงของการเกิดโรคระบาดและมีคนเป็นโรคระบาดนั้นเป็นจำนวนมาก การจ่ายยาของแพทย์สำหรับโรคดังกล่าวก็จะเกิดขึ้นซ้ำๆ กันสำหรับผู้ป่วยแต่ละราย เป็นต้น โดยหลักการทำงานจะนำรายการข้อมูลที่ซ้ำกันเหล่านั้นมารวมกันเป็นรายการข้อมูลเดียวกัน ก่อนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย เพื่อลดระยะเวลาในการทำงานให้รวดเร็ว และใช้โครงสร้างข้อมูลที่มีประสิทธิภาพในการทำงานเพื่อลดเนื้อที่หน่วยความจำในการจัดเก็บข้อมูล อีกซึ่งช่วยให้การทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยมีประสิทธิภาพมากยิ่งขึ้น

1.1 การตรวจสอบเอกสารและงานวิจัยที่เกี่ยวข้อง

การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเป็นขั้นตอนการทำงานที่สำคัญสำหรับการค้นหากฎความสัมพันธ์ เนื่องจากต้องค้นหากลุ่มข้อมูล (Itemsets) ที่อาจเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets) เพื่อนำไปสร้างกฎความสัมพันธ์ ซึ่งขั้นตอนนี้จะใช้เวลาและเนื้อที่ในหน่วยความจำเป็นจำนวนมาก จึงทำให้มีงานวิจัยจำนวนมากที่ได้ทำการสร้างและพัฒนาขั้นตอนวิธีที่จะช่วยให้การทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นให้มีประสิทธิภาพมากที่สุด โดยทั่วไปขั้นตอนวิธีที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นสามารถแบ่งประเภทตามลักษณะของวิธีการค้นหาออกเป็น 2 วิธีการดังนี้

1.1.1 วิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยแบบการสร้างกลุ่มข้อมูลท้าชิงและทดสอบ (Candidate Generate and Test Approach)

ขั้นตอนวิธี Apriori (Apriori Algorithm) เป็นขั้นตอนวิธีพื้นฐานที่เป็นที่รู้จักในการทำเหมืองข้อมูลในส่วนของการค้นหากฎความสัมพันธ์ของข้อมูล โดยมี Agrawal และ Srikant เป็นคนคิดค้นขึ้นมาเพื่อแก้ปัญหาการค้นหากฎความสัมพันธ์ในฐานข้อมูลขนาดใหญ่ของรายการข้อมูลการขาย (Agrawal and Srikant, 1993) การตั้งชื่อของขั้นตอนวิธี Apriori นั้นเกิดขึ้นจากลักษณะการทำงานของขั้นตอนวิธี นั่นคือ ขั้นตอนวิธีจะทำงานแบบวนรอบไปเรื่อยๆ เป็นลำดับชั้น กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในรอบปัจจุบันจะนำไปสร้างเป็นกลุ่มข้อมูลท้าชิง (Candidate Itemsets) เพื่อใช้ในการตรวจสอบว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในรอบถัดไปหรือไม่ การทำงานเริ่มจากการสร้างกลุ่มข้อมูลสำหรับเป็นกลุ่มข้อมูลท้าชิง โดยการนับค่าความถี่ (Count) ของการเกิดกลุ่มข้อมูลท้าชิงเหล่านั้นจากฐานข้อมูล ซึ่งวิธีการนี้ต้องใช้

เวลานานในการสร้างกลุ่มข้อมูลท้าชิง และอ่านข้อมูลจากฐานข้อมูลหลายครั้ง อีกทั้งยังใช้เนื้อที่ในการจัดเก็บข้อมูลในระหว่างการทำงานเป็นจำนวนมาก จึงทำให้ขั้นตอนวิธี Apriori เหมาะสำหรับฐานข้อมูลที่มีความหนาแน่นของชิ้นข้อมูล (Item) ในรายการข้อมูลบางตา คือ มีจำนวนชิ้นข้อมูลต่อหนึ่งรายการข้อมูลน้อยๆ เหมาะกับขนาดของฐานข้อมูลที่ไม่ใหญ่ และมีการกำหนดค่าสนับสนุนขั้นต่ำ (Minimum Support) มากๆ โดยค่าสนับสนุนขั้นต่ำ คือค่าความถี่ที่ผู้ใช้กำหนดขึ้นมาเพื่อใช้ในการตัดสินว่ากลุ่มข้อมูลนั้นเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ ซึ่งหากกลุ่มข้อมูลนั้นจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้นั้นต้องมีค่าความความถี่มากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำที่กำหนด

การสร้างขั้นตอนวิธี Apriori ได้รับความนิยมและเป็นที่ยอมรับเป็นจำนวนมาก แต่เนื่องจากขั้นตอนวิธีดังกล่าวยังมีข้อจำกัดในหลายด้าน ทำให้มีนักวิจัยจำนวนมากได้คิดค้นขั้นตอนวิธีอื่นๆ ขึ้นมามากมาย เพื่อพัฒนาและแก้ปัญหาการทำงานของขั้นตอนวิธี Apriori ให้มีประสิทธิภาพมากยิ่งขึ้น แต่ยังคงหลักการทำงานของขั้นตอนวิธี Apriori ไว้ ยกตัวอย่างเช่น Pasquier และคณะได้นำเสนอขั้นตอนวิธีใหม่ขึ้นมาที่ช่วยลดจำนวนของการสร้างกลุ่มข้อมูลท้าชิงให้น้อยลง (Pasquier *et al.*, 1999) โดยใช้ชื่อเรียกขั้นตอนวิธีนี้ว่า ขั้นตอนวิธี Close (Close Algorithm) ซึ่งการทำงานของขั้นตอนวิธี Close จะใช้วิธีการของกลุ่มข้อมูลปิด (Closed Itemsets) ในการลดจำนวนของการสร้างกลุ่มข้อมูลท้าชิง ซึ่งใช้คุณสมบัติที่ว่า ถ้ากลุ่มข้อมูลปิดที่เป็นซูเปอร์เซตเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยแล้ว กลุ่มข้อมูลปิดที่เป็นเซตย่อยจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยด้วย และเซตของกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ยาวที่สุดจะเท่ากับเซตของกลุ่มข้อมูลปิดที่ปรากฏร่วมกันบ่อยที่ยาวที่สุดด้วย ทำให้ใช้เวลาในการประมวลผลน้อยกว่าขั้นตอนวิธี Apriori แต่ขั้นตอนวิธี Close ไม่สามารถหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นในฐานข้อมูลได้ทุกกรณีที่เป็นไปได้ เนื่องจากการค้นหาของขั้นตอนวิธี Close นั้นจะค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เป็นซูเปอร์เซตเท่านั้น จึงทำให้ขั้นตอนวิธีนี้เหมาะสำหรับงานที่ต้องการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดใหญ่ที่สุดโดยที่ไม่สนใจกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดเล็ก

Dong และ Han ได้ใช้หลักการทำงานหลักๆ ของขั้นตอนวิธี Apriroi แต่ได้ปรับเปลี่ยนในส่วนของโครงสร้างข้อมูลที่ใช้ในการจัดเก็บข้อมูลในระหว่างการประมวลผลและจำนวนของการอ่านข้อมูลจากฐานข้อมูล (Dong and Han, 2007) ขั้นตอนวิธีนี้มีชื่อว่าขั้นตอนวิธี BitTableFI (BitTableFI Algorithm) ตั้งชื่อตามโครงสร้างข้อมูลที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย นั่นคือ BitTable โดยลักษณะการจัดเก็บข้อมูลใน BitTable จะเก็บข้อมูลอยู่ในรูปของบิต (Bit) ซึ่งกำหนดค่าบิตเป็น 1 หากมีชิ้นข้อมูลนั้นหรือกำหนดค่าบิตเป็น 0 หากไม่มีชิ้นข้อมูลนั้นในรายการข้อมูล เพื่อลดเนื้อที่หน่วยความจำในการจัดเก็บข้อมูลระหว่างการทำงานและอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียวเพื่อลดระยะเวลาการทำงาน โดยหลักการทำงานขั้นตอนวิธี BitTableFI สามารถแก้ปัญหาการสร้างกลุ่มท้าชิงและการนับค่าความถี่ของกลุ่มท้า

ชิงได้อย่างรวดเร็วเมื่อเทียบกับขั้นตอนวิธี Apriori โดยใช้การดำเนินการระดับบิต (Bitwise Operations) ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย แต่ขั้นตอนวิธีนี้ก็ยังคงต้องทำการสร้างกลุ่มท้าชิงจำนวนมาก จึงทำให้ขั้นตอนวิธี BitTableFI เหมาะสำหรับฐานข้อมูลที่มีความหนาแน่นของชิ้นข้อมูลในรายการข้อมูลบางตาเช่นเดียวกับ Apriori แต่สิ่งที่แตกต่างกัน คือ ขั้นตอนวิธี BitTableFI ทำงานได้ดีทั้งฐานข้อมูลที่มีขนาดเล็กและขนาดใหญ่ อีกทั้งลดเวลาที่ใช้ในการสร้างกลุ่มข้อมูลท้าชิงให้สามารถสร้างได้เร็วขึ้น

1.1.2 วิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยแบบการเติบโตอย่างเป็นรูปแบบ (Pattern Growth Approach)

วิธีการนี้จะไม่มีการสร้างกลุ่มข้อมูลท้าชิง จึงทำให้ลดจำนวนการอ่านข้อมูลจากฐานข้อมูลหลายครั้ง และจากเหตุผลนี้ทำให้ขั้นตอนวิธีในวิธีการแบบนี้สามารถกำหนดค่าสนับสนุนขั้นต่ำน้อยๆ แล้วยังคงสามารถทำงานได้ดี ขั้นตอนวิธีที่เป็นที่รู้จักสำหรับการทำงานในลักษณะวิธีการแบบนี้ คือ ขั้นตอนวิธี FP-Growth (FP-Growth Algorithm) โดย Han และคณะ (Han *et al*., 2000) ขั้นตอนวิธีนี้ได้ใช้โครงสร้างข้อมูลแบบต้นไม้ (Tree Data Structure) ที่มีชื่อเรียกว่า FP-Tree (Frequent Pattern Tree) ในการจัดเก็บข้อมูลที่อ่านจากฐานข้อมูลเข้ามา โดยจำนวนของการอ่านข้อมูลจากฐานข้อมูลนั้นมีการอ่าน 2 ครั้ง การทำงานจะเริ่มจากการอ่านข้อมูลเพื่อสร้างต้นไม้ FP-Tree จากนั้นใช้การท่องต้นไม้ (Tree Traversal) ด้วยการค้นหาแบบแนวลึก (Depth First Search) ก่อน ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ขั้นตอนวิธีนี้ไม่มีการสร้างกลุ่มข้อมูลท้าชิงทำให้สามารถทำงานได้เร็ว แต่หากฐานข้อมูลที่มีความหนาแน่นของข้อมูลในรายการข้อมูลบางตาแต่มีจำนวนของชิ้นข้อมูลในฐานข้อมูลมากแล้ว จะทำให้ต้นไม้ที่สร้างเพื่อใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นมีกิ่งจำนวนมากและต้นไม้มีขนาดใหญ่ ซึ่งจะทำให้ต้องใช้เนื้อที่ในหน่วยความจำเป็นจำนวนมาก

นอกจากนั้นมี Sucahyo และ Gopalan ได้ร่วมกันคิดค้นขั้นตอนวิธี CT-PRO (CT-PRO Algorithm) ขึ้นมา ซึ่งมีลักษณะการทำงานที่คล้ายกับขั้นตอนวิธี FP-Growth แต่ขั้นตอนวิธีนี้อ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียว (Sucahyo and Gopalan, 2004) จึงทำให้การทำงานเร็วขึ้น อีกทั้งยังได้สร้างโครงสร้างข้อมูลแบบต้นไม้ใหม่ขึ้นมาที่มีชื่อเรียกว่า CFP-Tree (Compressed FP-Tree Structure) ซึ่งพัฒนามาจาก FP-Tree ออกแบบเพื่อลดจำนวนโหนดของต้นไม้ที่ใช้ในการจัดเก็บข้อมูล โดยอย่างน้อยที่สุดการจัดเก็บข้อมูลจะช่วยลดจำนวนโหนดเป็นครึ่งหนึ่งของจำนวนโหนดทั้งหมดในโครงสร้างข้อมูล FP-Tree แต่วิธีการดังกล่าวก็ยังต้องใช้เนื้อที่จำนวนมากในหน่วยความจำอยู่

ลักษณะการทำงานของขั้นตอนวิธีที่ใช้โครงสร้างข้อมูลแบบต้นไม้มาใช้ในการจัดเก็บข้อมูลนั้น เพื่อหลีกเหลี่ยงการสร้างกลุ่มข้อมูลท้าชิงและลดจำนวนการอ่านข้อมูลจากฐานข้อมูล ซึ่งมีนักวิจัยได้พัฒนาขึ้นมาอีกหลากหลายขั้นตอนวิธีนอกเหนือจากที่กล่าวมาข้างต้น

ยกตัวอย่างเช่น ขั้นตอนวิธี H-Mine (H-Mine Algorithm) ที่ได้นำหลักการทำงานแบบพลวัตร (Dynamic Programming) มาช่วยให้การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพื่อให้ประมวลผลได้เร็วขึ้น (Pei *et al*., 2001) ขั้นตอนวิธี ITL-Mine (ITL-Mine Algorithm) วิธีนี้ได้นำเสนอการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ใช้หลักการการเรียกตัวเอง (Recursive) และหากในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นมีการกำหนดค่าสนับสนุนขั้นต่ำครั้งถัดไปมากกว่าค่าสนับสนุนขั้นต่ำที่กำลังประมวลผลอยู่ก็จะไม่ต้องอ่านข้อมูลจากฐานข้อมูลใหม่ (Gopalan and Sucahyo, 2002) และขั้นตอนวิธี CT-ITL (CT-ITL Algorithm) เป็นขั้นตอนวิธีที่พัฒนามาจากขั้นตอนวิธี ITL-Mine เพื่อให้สามารถทำงานได้อย่างมีประสิทธิภาพยิ่งขึ้น (Sucahyo and Gopalan, 2003) เป็นต้น ซึ่งแต่ละขั้นตอนวิธีที่กล่าวมานั้นก็จะมีความเหมาะสมกับลักษณะข้อมูลและเนื้อที่ที่ใช้ในระหว่างการประมวลผลที่แตกต่างกันไป

จากขั้นตอนวิธีที่คิดค้นขึ้นมาดังกล่าวข้างต้น สามารถแบ่งออกเป็น 2 แบบหลักๆ คือ แบบการสร้างกลุ่มข้อมูลท้าชิงและทดสอบ และแบบการเติบโตอย่างเป็นรูปแบบ แต่ก็มีบางขั้นตอนวิธีที่มีการนำหลักการทำงานทั้งสองแบบนี้มาใช้ร่วมกัน ยกตัวอย่างเช่น ขั้นตอนวิธี Index-BitTable (Index-BitTable Algorithm) เสนอโดย Song และคณะ ซึ่งขั้นตอนวิธีนี้นำหลักการการทำงานผสมผสานระหว่างวิธีการแบบการสร้างกลุ่มข้อมูลท้าชิงและทดสอบกับวิธีการแบบรูปแบบการเติบโตอย่างเป็นรูปแบบ (Song *et al*., 2008) โดยได้นำการดำเนินการระดับบิตมาใช้เช่นเดียวกันกับขั้นตอนวิธี BitTableFI แต่ขั้นตอนวิธีนี้ได้หลีกเหลี่ยงในการสร้างกลุ่มข้อมูลท้าชิงให้น้อยลง เพื่อให้การทำงานเร็วขึ้นโดยการสร้างดัชนี (Index) ขึ้นมาในการแก้ปัญหาการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย การทำงานจะใช้การดำเนินการระดับบิตกับ BitTable ในแนวนอน (Horizontally) เพื่อค้นหากลุ่มข้อมูลและใช้ BitTable ในแนวตั้ง (Vertically) เพื่อนับค่าความถี่ของกลุ่มข้อมูล ขั้นตอนวิธีนี้อ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียวซึ่งเกิดขึ้นในขั้นตอนการสร้าง BitTable ทำให้การทำงานเร็วและมีประสิทธิภาพมากขึ้น โดยขั้นตอนวิธีนี้จะทำงานได้ดีกับฐานข้อมูลที่มีลักษณะรายการข้อมูลที่มีความหนาแน่นของชิ้นข้อมูลสูง อีกทั้งขั้นตอนวิธีนี้ยังสามารถทำงานได้ดีถึงแม้มีการกำหนดค่าสนับสนุนขั้นต่ำน้อยๆ

จากการตรวจสอบเอกสารและงานวิจัยที่เกี่ยวข้องสามารถสรุปได้ว่า แต่ละขั้นตอนวิธีได้พัฒนาขึ้นมาเพื่อจุดประสงค์ที่แตกต่างกัน บางขั้นตอนวิธีสร้างขึ้นเพื่อให้สามารถลดจำนวนของการอ่านข้อมูลจากฐานข้อมูลเพื่อให้ทำงานเร็วขึ้น บางขั้นตอนวิธีสร้างขึ้นเพื่อให้ลดการใช้เนื้อที่ในหน่วยความจำ บางขั้นตอนวิธีสร้างขึ้นเพื่อให้เหมาะสมกับลักษณะข้อมูลในฐานข้อมูล หรือบางขั้นตอนวิธีพัฒนาขั้นตอนวิธีที่มีอยู่แล้วให้มีประสิทธิภาพมากยิ่งขึ้น วิทยานิพนธ์นี้จึงนำเสนอขั้นตอนวิธีอีกขั้นตอนวิธีหนึ่งขึ้นมาเพื่อเป็นทางเลือกสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในฐานข้อมูลที่มีลักษณะรายการข้อมูลที่ซ้ำๆ ในฐานข้อมูล โดยได้ประยุกต์โครงสร้างข้อมูลที่มีประสิทธิภาพที่มีอยู่แล้วเพื่อลดเนื้อที่หน่วยความจำในระหว่าง

การทำงานและไม่มีการสร้างกลุ่มข้อมูลท้าชิง เพื่อให้สามารถทำงานได้เร็วขึ้นและมีประสิทธิภาพมากยิ่งขึ้น

1.2 วัตถุประสงค์

1.2.1 คิดค้นขั้นตอนวิธีใหม่ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยภายในฐานข้อมูลที่มีจำนวนของรายการข้อมูลซ้ำ ๆ กัน

1.2.2 เปรียบเทียบประสิทธิภาพการทำงานของขั้นตอนวิธีที่สร้างขึ้นกับขั้นตอนวิธีพื้นฐาน เช่น ขั้นตอนวิธี Apriori

1.3 ขอบเขตการดำเนินงาน

1.3.1 วิเคราะห์ ออกแบบ และพัฒนาขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

1.3.2 ประเมินประสิทธิภาพของขั้นตอนวิธีที่สร้างขึ้น โดยเปรียบเทียบกับขั้นตอนวิธีพื้นฐาน เช่น ขั้นตอนวิธี Apriori

1.4 ขั้นตอนและระยะเวลาการดำเนินงาน

1.4.1 ขั้นตอนการดำเนินงาน

1) ศึกษาทฤษฎี เอกสารและงานวิจัยที่เกี่ยวข้องที่นำมาใช้ในวิทยานิพนธ์นี้ พร้อมทั้งกำหนดขอบเขตของปัญหาให้ชัดเจน

2) ศึกษาเครื่องมือ (Tool) และซอฟต์แวร์ (Software) ที่ใช้สำหรับวิทยานิพนธ์นี้

3) วิเคราะห์และออกแบบขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

4) เขียนบทความวิจัย

5) เขียนโปรแกรมเพื่อสร้างขั้นตอนวิธีตามที่ได้ออกแบบไว้

6) ประเมินประสิทธิภาพโดยการเปรียบเทียบการทำงานของขั้นตอนวิธีที่สร้างขึ้นกับขั้นตอนวิธีอื่น เช่น ขั้นตอนวิธี Apriori

7) สรุปและวิเคราะห์ผลการประเมินประสิทธิภาพ

8) จัดทำเอกสารวิทยานิพนธ์

1.4.2 ระยะเวลาการดำเนินงาน

มิถุนายน 2551 – มีนาคม 2553

1.4.3 แผนการดำเนินการวิจัย

ตารางที่ 1.1 ระยะเวลาการดำเนินการวิจัย

กิจกรรม/ขั้นตอนการดำเนินงาน	เดือน																					
	2551							2552												2553		
	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1. ศึกษาทฤษฎี เอกสารและงานวิจัยที่เกี่ยวข้อง	◄	─	─	─	─	─	─	─	─	─	─	─	─	─	─	─	─	─	►			
2. ศึกษาเครื่องมือและซอฟต์แวร์		◄	─	─	─	►																
3. วิเคราะห์และออกแบบขั้นตอนวิธี			◄	─	─	►																
4. เขียนบทความวิจัย						◄	─	►														
5. เขียนโปรแกรมเพื่อสร้างขั้นตอนวิธี									◄	─	─	─	─	─	─	─	─	─	►			
6. ประเมินประสิทธิภาพของขั้นตอนวิธี																◄	─	►				
7. สรุปผลและวิเคราะห์การทำงาน																		◄	►			
8. จัดทำเอกสารวิทยานิพนธ์																		◄	─	─	─	►

1.5 สถานที่และเครื่องมือที่ใช้

1.5.1 สถานที่

ห้องปฏิบัติการวิจัยเทคโนโลยีระบบสารสนเทศและการประยุกต์ (CS207) ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์

1.5.2 เครื่องมือที่ใช้

1) ด้านฮาร์ดแวร์

เครื่องคอมพิวเตอร์ส่วนบุคคล หน่วยประมวลผลกลาง ความเร็ว 1.86 GHz หน่วยความจำขนาด 1 GB และฮาร์ดดิสก์ความจุ 150 GB สำหรับใช้ในการเขียนโปรแกรมเพื่อสร้างขั้นตอนวิธี และประเมินประสิทธิภาพของขั้นตอนวิธีที่สร้างขึ้น

2) ด้านซอฟต์แวร์

2.1) ระบบปฏิบัติการ Microsoft Windows XP

2.2) ตัวแปลภาษาซี (C Compiler)

1.6 ประโยชน์ที่คาดว่าจะได้รับ

ได้ขั้นตอนวิธีใหม่ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เหมาะสมสำหรับรายการข้อมูลภายในฐานข้อมูลที่มีความคล้ายคลึงกัน

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่นำมาใช้สำหรับการสร้างขั้นตอนวิธีในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกี่ยวข้องกับงานวิจัยในวิทยานิพนธ์เล่มนี้ประกอบด้วย ความรู้พื้นฐานทางด้านสาขาวิชาคณิตศาสตร์ในเรื่องของเซต (Set) นิยามเกี่ยวกับกฎความสัมพันธ์ที่นำมาใช้ในการสร้างขั้นตอนวิธี การดำเนินการระดับบิต (Bitwise Operations) อีกทั้งได้กล่าวถึงความรู้เกี่ยวกับการทำเหมืองข้อมูล การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย และสุดท้ายการสร้างกฎความสัมพันธ์จากกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเหล่านั้น ซึ่งรายละเอียดต่างๆ อธิบายแยกเป็นหัวข้อ ดังนี้

2.1 นิยามเกี่ยวกับเซต (Set)

สำหรับวิทยานิพนธ์นี้ได้ใช้ความรู้ทางด้านสาขาวิชาคณิตศาสตร์ในเรื่องเซต (Simovici and Djeraba, 2008; Koshy, 2005; Meinel and Theobald, 1998) มาช่วยในการสร้างขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย จึงขอกล่าวถึงบทนิยามต่าง ๆ ที่เกี่ยวข้องพอสังเขปดังนี้

<u>บทนิยามที่ 2.1.1</u> เซต (Set) คือ ลักษณะนามที่ใช้เรียกกลุ่มของสิ่งต่างๆ เช่น กลุ่มของคน สัตว์ สิ่งของ ตัวเลข เป็นต้น และสิ่งต่างๆ ที่อยู่ในเซต จะเรียกว่า สมาชิก (Element) ของเซต ใช้สัญลักษณ์ $\in$ แทนการเป็นสมาชิกของเซตและใช้สัญลักษณ์ $\notin$ แทนการไม่เป็นสมาชิกของเซต

<u>บทนิยามที่ 2.1.2</u> เซตว่าง (Empty Set) คือ เซตที่ไม่มีสมาชิกใด ๆ อยู่เลย หรือจำนวนสมาชิกเป็นศูนย์ เขียนแทนด้วยสัญลักษณ์ $\varnothing$ หรือ { }

<u>บทนิยามที่ 2.1.3</u> กำหนดให้เซต A และ B เป็นเซตใดๆ เซต A เป็นเซตย่อย (Subset) ของเซต B ก็ต่อเมื่อทุกสมาชิกของเซต A เป็นสมาชิกอยู่ในเซต B เขียนแทนด้วยสัญลักษณ์ $A \subseteq B$ และเซต B จะเป็นซูเปอร์เซต (Superset) ของเซต A ก็ต่อเมื่อ A เป็นเซตย่อยของ B เขียนแทนด้วยสัญลักษณ์ $B \supseteq A$

<u>บทนิยามที่ 2.1.4</u> กำหนดให้เซต A เป็นเซตใดๆ เพาเวอร์เซต (Power Set) ของ A คือ เซตของเซตย่อยทั้งหมดของ A เขียนแทนด้วย $P(A)$ ถ้า A มีสมาชิก n ตัว $P(A)$ จะ

มีสมาชิก 2^n ตัว ยกตัวอย่างเช่น $A = \{2, 3, 5\}$ จะได้ $P(A) = 2^3$ คือ มีสมาชิกทั้งหมด 8 ตัว ได้แก่ {{2}, {3}, {5}, {2, 3}, {2, 5}, {3, 5}, {2, 3, 5}, ∅}

บทนิยามที่ 2.1.5 กำหนดให้เซต A และ B เป็นเซตใดๆ ผลรวมของเซต A กับเซต B คือ เซตซึ่งประกอบไปด้วยสมาชิกที่อยู่ในเซต A หรืออยู่ในเซต B เรียกว่า ยูเนียน (Union) และเขียนแทนด้วยสัญลักษณ์ $A \cup B$ โดยที่ $A \cup B = \{x \mid x \in A$ หรือ $x \in B\}$

บทนิยามที่ 2.1.6 กำหนดให้เซต A และ B เป็นเซตใดๆ ผลร่วมของเซต A และเซต B คือ เซตซึ่งประกอบไปด้วยสมาชิกที่อยู่ในเซต A และอยู่ในเซต B เรียกว่า อินเตอร์เซ็คชั่น (Intersection) และเขียนแทนด้วยสัญลักษณ์ $A \cap B$ โดยที่ $A \cap B = \{x \mid x \in A$ และ $x \in B\}$

2.2 นิยามเกี่ยวกับกฎความสัมพันธ์ (Association Rules)

ในส่วนนี้จะให้นิยามเกี่ยวกับการค้นหากฎความสัมพันธ์ (Kantardzic, 2002; Han *et al.*, 2006) เพื่อใช้แทนสิ่งต่างๆ ต่อไปในวิทยานิพนธ์นี้ โดยกำหนดให้ $I = \{i_1, i_2, ..., i_n\}$ คือ เซตของชิ้นข้อมูล (Items) ที่ประกอบด้วยชิ้นข้อมูลเป็นสมาชิกจำนวน n ชิ้นข้อมูล และให้ $D = \{T_1, T_2, ..., T_m\}$ คือ เซตของรายการข้อมูล (Transaction) ในฐานข้อมูล D ที่ประกอบด้วยรายการข้อมูลเป็นสมาชิกจำนวน m รายการข้อมูล สำหรับใช้ในบทนิยามที่ 2.2.1 ถึง 2.2.5

บทนิยามที่ 2.2.1 ให้เซต $X \subseteq I$ และเรียก X ว่ากลุ่มข้อมูล (Itemsets) ซึ่งกลุ่มข้อมูลที่ประกอบด้วยชิ้นข้อมูลจำนวน k ชิ้นข้อมูล เรียกว่า กลุ่มข้อมูลขนาด k ชิ้นข้อมูล เขียนแทนด้วย k-Itemsets

บทนิยามที่ 2.2.2 ให้ค่าความถี่ (Count) ของ X สำหรับกลุ่มข้อมูล X ใดๆ จะมีค่าเท่ากับจำนวนของรายการข้อมูลทั้งหมดใน D ที่มี X อยู่

บทนิยามที่ 2.2.3 ให้ค่าสนับสนุน (Support) ของ X สามารถคำนวณได้จากสมการ (2.1)

$$\text{ค่าสนับสนุน } X = \frac{\text{ค่าความถี่ของ } X}{\text{จำนวนของรายการข้อมูลทั้งหมดใน } D} \qquad (2.1)$$

บทนิยามที่ 2.2.4 กฎความสัมพันธ์ (Association Rules) คือ กฎที่สร้างจากกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยอยู่ในรูปของ $X \rightarrow Y$ แทนเหตุการณ์ของการเกิด X แล้วเกิดเหตุการณ์ Y ร่วมกัน โดยที่ $X \subseteq I$, $Y \subseteq I$ และ $X \cap Y = \varnothing$

บทนิยามที่ 2.2.5 ค่าความเชื่อมั่น (Confidence) คือ ความน่าเชื่อถือของกฎความสัมพันธ์ที่สร้างขึ้น โดยกฎ $X \rightarrow Y$ สามารถคำนวณได้จากสมการ (2.2)

$$\text{ค่าความเชื่อมั่นของกฎ } X \rightarrow Y = \frac{\text{ค่าความถี่ของ } X \text{ และ } Y \text{ ที่เกิดร่วมกัน}}{\text{ค่าความถี่ของ } X} \quad (2.2)$$

บทนิยามที่ 2.2.6 ค่าสนับสนุนขั้นต่ำ (Minimum Support) คือค่าสนับสนุนที่ผู้ใช้กำหนดขึ้นเพื่อเป็นเกณฑ์ในการตัดสินใจว่ากลุ่มข้อมูลใดเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยกลุ่มข้อมูลใดจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้นั้นจะต้องมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ

บทนิยามที่ 2.2.7 กลุ่มข้อมูลใดๆ ที่มีค่าสนับสนุนไม่ต่ำกว่าค่าสนับสนุนขั้นต่ำที่กำหนด เรียกว่า กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets) และกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล k ชิ้นข้อมูล (Frequent k-Itemsets) เขียนแทนด้วย L_k แต่หากกลุ่มข้อมูลใดๆ ที่มีค่าสนับสนุนต่ำกว่าค่าสนับสนุนขั้นต่ำที่กำหนด เรียกว่า กลุ่มข้อมูลที่ปรากฏร่วมกันไม่บ่อย (Infrequent Itemsets)

บทนิยามที่ 2.2.8 กลุ่มข้อมูลท้าชิง (Candidate Itemsets) หมายถึงกลุ่มข้อมูลที่พร้อมจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย หากมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ โดยหากกลุ่มข้อมูลท้าชิงมีขนาดของชิ้นข้อมูล k ชิ้นข้อมูล (Candidate k-Itemsets) เขียนแทนด้วย C_k

บทนิยามที่ 2.2.9 ค่าความเชื่อมั่นขั้นต่ำ (Minimum Confidence) หมายถึงค่าความเชื่อมั่นที่ผู้ใช้กำหนดขึ้นเพื่อเป็นเกณฑ์ในการตัดสินใจว่ากฎใดบ้างเป็นกฎความสัมพันธ์ (Association Rule) ที่ยอมรับได้ โดยกฎดังกล่าวจะเป็นกฎความสัมพันธ์ที่ยอมรับได้ก็ต่อเมื่อกฎนั้นมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

2.3 การดำเนินการระดับบิต (Bitwise Operation)

2.3.1 ตัวดำเนินการ (Operator)

ตัวดำเนินการ (Schildt, 1990) หมายถึง เครื่องหมายกำหนดกรรมวิธีทางคณิตศาสตร์ โดยการเปรียบเทียบระหว่างข้อมูล 2 ตัว ซึ่งเรียกว่าตัวถูกดำเนินการ (Operand) โดยอาจมีค่าเป็นตัวเลข ข้อความ ค่าคงที่ หรือตัวแปรต่างๆ เป็นต้น

2.3.2 ตัวดำเนินการเชิงตรรกะ (Logical operator)

ตัวดำเนินการเชิงตรรกะ หมายถึง การดำเนินการตามกฎทางตรรกศาสตร์ โดยที่ค่าผลลัพธ์ของการถูกดำเนินการจะมีความหมายได้ 2 ค่าคือ ค่าจริง (True) และค่าเท็จ (False) ในการเปรียบเทียบการดำเนินการเชิงตรรกะประกอบด้วยเครื่องหมายดังนี้

1) ตัวดำเนินการ AND แทนด้วยเครื่องหมาย “&” หมายถึง การดำเนินการ AND จะเป็นจริงเมื่อค่าที่ใช้เปรียบเทียบทั้ง 2 ค่าเป็นจริงทั้งคู่ เช่น ถ้าค่าของคู่เป็นจริงเหมือนกัน ผลลัพธ์ที่ได้จะเป็นจริง แต่ถ้าไม่ใช่ผลลัพธ์จะเป็นเท็จ

2) ตัวดำเนินการ OR แทนด้วยเครื่องหมาย “|” หมายถึง การดำเนินการ OR จะเป็นจริงเมื่อค่าที่ใช้เปรียบเทียบทั้ง 2 ค่าเป็นจริงทั้งคู่หรือจริงเพียงค่าใดค่าหนึ่ง เช่น ถ้าค่าใดค่าหนึ่งของทั้งคู่เป็นจริง ผลลัพธ์ที่ได้จะเป็นจริง แต่ถ้าไม่ใช่ผลลัพธ์ที่ได้จะเป็นเท็จ

3) ตัวดำเนินการ XOR แทนด้วยเครื่องหมาย “^” หมายถึง การดำเนินการ XOR จะเป็นจริงเมื่อค่าที่ใช้เปรียบเทียบทั้ง 2 ค่าเป็นจริงหรือเท็จเพียงค่าใดค่าหนึ่ง เช่น ถ้าค่าของทั้งคู่มีค่าที่แตกต่างกัน ผลลัพธ์ที่ได้จะเป็นจริง แต่ถ้าไม่ใช่ผลลัพธ์ที่ได้จะเป็นเท็จ

4) ตัวดำเนินการ NOT แทนด้วยเครื่องหมาย “!” หมายถึง การดำเนินการ NOT จะเป็นการแปลงค่าตรงกันข้ามจากจริงจะเป็นเท็จ และจากเท็จจะเป็นจริง เป็นคำสั่งที่ใช้ในการเปลี่ยนค่าของตัวถูกดำเนินการให้เป็นค่าตรงกันข้าม

2.3.3 ตัวดำเนินการระดับบิต (Bitwise operator)

ตัวดำเนินการระดับบิต เป็นการดำเนินการเชิงตรรกะในระดับบิต โดยจะใช้มุมมองในแบบเลขฐานสองมาจัดการกับข้อมูล นั่นคือข้อมูลตัวเลขนั้นจะถูกแปลงเป็นเลขฐานสองในหน่วยความจำในขณะที่มีการดำเนินการเชิงตรรกะในระดับบิต ซึ่งแต่ละตำแหน่งของบิตมีค่าที่เป็นไปได้อยู่สองค่า คือ หากตำแหน่งบิตนั้นมีค่าเป็น 1 จะหมายถึงมีค่าเป็นจริง และหากตำแหน่งบิตนั้นมีค่าเป็น 0 จะหมายถึงมีค่าเป็นเท็จ โดยหากกำหนดตัวแปร *x* มีค่าเป็น 10110 และตัวแปร *y* มีค่าเป็น 01110 สามารถอธิบายการทำงานของตัวดำเนินการระดับบิตได้ดังนี้

1) ตัวดำเนินการระดับบิต AND (Bitwise AND Operator) เขียนแทนด้วยเครื่องหมาย “*x* & *y*” หมายถึง การเทียบบิตแบบ AND ระหว่าง *x* กับ *y* นั่นคือ “10110 & 01110” ให้ผลลัพธ์เป็น 00110

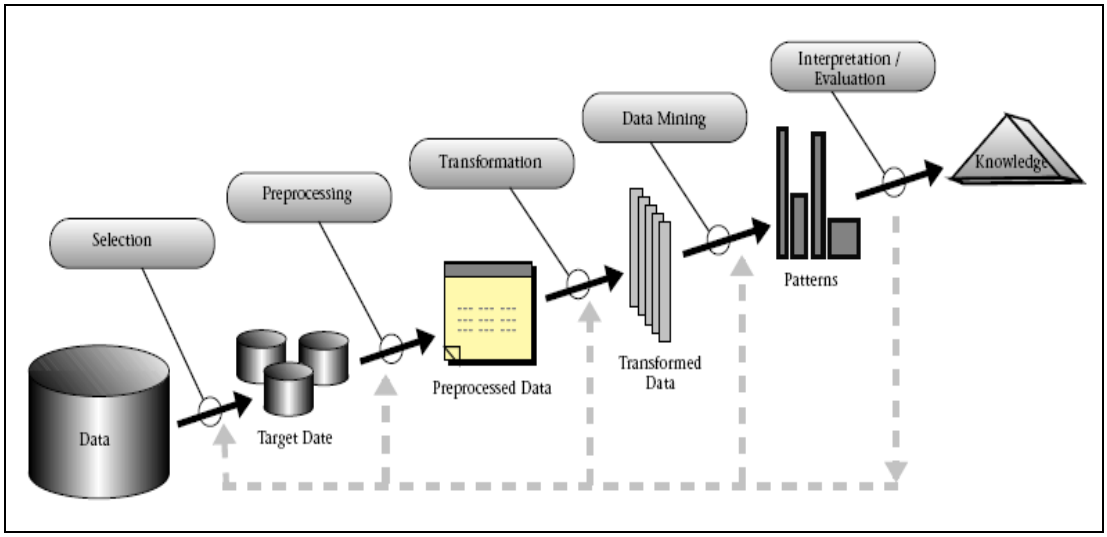
2) ตัวดำเนินการระดับบิต OR (Bitwise Inclusive OR Operator) เขียนแทนด้วยเครื่องหมาย “*x* | *y*” หมายถึง การเทียบบิตแบบ OR ระหว่าง *x* กับ *y* นั่นคือ “10110 | 01110” ให้ผลลัพธ์เป็น 11110

3) ตัวดำเนินการระดับบิต XOR (Bitwise Exclusive OR Operator) เขียนแทนด้วยเครื่องหมาย “*x* ^ *y*” หมายถึง การเทียบบิตแบบ XOR ระหว่าง *x* กับ *y* นั่นคือ “10110 ^ 01110” ให้ผลลัพธ์11000

4) ตัวดำเนินการระดับบิต NOT (Bitwise NOT Operator) เขียนแทนด้วยเครื่องหมาย “~*x*” หมายถึง การสลับค่าบิตให้ *x* จะให้ผลลัพธ์ของบิตมีค่าตรงข้าม จาก 10110 กลายเป็น 01001

2.4 การทำเหมืองข้อมูล (Data Mining)

การทำเหมืองข้อมูล เป็นขั้นตอนหนึ่งในการค้นหาความรู้ในฐานข้อมูล (Fayyad *et al.*, 1996) ซึ่งประกอบด้วยการเลือกขั้นตอนวิธีที่จะใช้ในการวิเคราะห์ พร้อมทั้งการวิเคราะห์ข้อมูลเพื่อค้นหารูปแบบที่ซ่อนอยู่ในข้อมูลเหล่านั้น (Olson *et al.*, 2008) โดยขั้นตอนการทำงานทั้งหมดสามารถแสดงดังภาพประกอบ 2.1



ภาพประกอบ 2.1 กระบวนการค้นหาความรู้ในฐานข้อมูล (Fayyad *et al.*, 1996)

การทำเหมืองข้อมูลเป็นส่วนสำคัญส่วนหนึ่งสำหรับการค้นหาความรู้ในฐานข้อมูล จุดประสงค์ของการทำเหมืองข้อมูล คือ การดึงรูปแบบแนวโน้มและกฎเกณฑ์จากข้อมูลเพื่อนำมาพัฒนาหรือประยุกต์ใช้กับองค์กร หรือเป็นตัวตัดสินในการปรับปรุงการดำเนินงานหรือวิธีการขององค์กร ผลลัพธ์ที่ได้จากการทำเหมืองข้อมูล คือ ได้ความรู้หรือสิ่งที่

เป็นประโยชน์ใหม่ๆ เพื่อระบุเหตุผลที่สนับสนุนการตัดสินใจ และเข้าใจถึงรูปแบบของข้อมูล (Taniar, 2007) ซึ่งการทำเหมืองข้อมูลมีกระบวนการทำงานเพื่อให้ได้มาซึ่งความรู้ มีขั้นตอนการทำงานที่สำคัญๆ ดังนี้

2.4.1 การเลือกข้อมูล (Data Selection)

การเลือกข้อมูล คือ การระบุแหล่งของข้อมูลที่มีและดึงเอาข้อมูลออกมาใช้สำหรับการวิเคราะห์เบื้องต้นเพื่อนำไปสู่ขั้นตอนการเตรียมข้อมูล ซึ่งต้องทำความเข้าใจถึงลักษณะของข้อมูล (Understanding Data) ในฐานข้อมูลว่าข้อมูลแต่ละตัวแปรนั้นมีความหมายว่าอย่างไร ต้องมีคำอธิบายที่ชัดเจนเกี่ยวกับชนิดของข้อมูล ค่าที่เป็นไปได้ แหล่งกำเนิดของข้อมูล รูปแบบของข้อมูล เพื่อใช้ในการกำหนดเป้าหมาย (Target Data) ของผลลัพธ์หรือความรู้ที่ต้องการจากฐานข้อมูลเหล่านั้นสำหรับในกระบวนการค้นหาความรู้ในฐานข้อมูล โดยเลือกเฉพาะข้อมูลที่คาดว่าเกี่ยวข้องกับเป้าหมายที่ต้องการจากฐานข้อมูลเพื่อใช้เป็นข้อมูลนำเข้า (Input) สู่ขั้นตอนต่อไปของกระบวนการค้นหาความรู้ในฐานข้อมูล ซึ่งการเลือกข้อมูลนั้นจะแตกต่างกันไปตามวัตถุประสงค์ของแต่ละงานที่ได้กำหนดไว้ตั้งแต่ต้น

2.4.2 การเตรียมข้อมูล (Data Preprocessing)

การเตรียมข้อมูลเป็นขั้นตอนที่สำคัญขั้นตอนหนึ่ง เนื่องจากต้องให้ความสำคัญในการตรวจสอบข้อมูลและทำความสะอาดข้อมูล (Cleansing Data) เพื่อให้ได้มาซึ่งข้อมูลนำเข้าที่มีคุณภาพ หากข้อมูลนำเข้าไม่มีคุณภาพแล้วข้อมูลนำออก (Output) หรือผลลัพธ์ที่ได้นั้นจะดีได้อย่างไร โดยขั้นตอนการเตรียมข้อมูลสามารถแยกออกเป็น 3 วิธีการดังนี้

1) การตรวจสอบขอบเขตของข้อมูล คือ กำจัดข้อมูลที่ไม่ถูกต้องหรือข้อมูลที่ไม่สอดคล้องออกไป เช่น ข้อมูลเงินเดือนต้องเป็นตัวเลข ห้ามเป็นตัวอักษรหรือสัญลักษณ์ เป็นต้น

2) การกำจัดข้อมูลสูญหาย (Missing Data) หรือข้อมูลที่ไม่สมบูรณ์ เช่น แก้ไขข้อมูลที่เป็นค่าว่าง (Null) เพื่อให้มีค่าหรือตัดข้อมูลนั้นออกไป เพื่อไม่ให้มีผลกระทบกับข้อมูลอื่นๆ เป็นต้น

3) การกำจัดข้อมูลที่ผิดปกติ (Noisy Data) คือ ค่าที่มีความแตกต่างไปจากข้อมูลอื่นๆ ในฐานข้อมูลอย่างชัดเจน โดยในการตรวจสอบเพื่อกำจัดข้อมูลเหล่านี้อาจใช้วิธีการทางสถิติมาประกอบการตัดสินใจ

2.4.3 การลดรูปหรือแปลงข้อมูล (Data Transformation)

การลดรูปหรือแปลงข้อมูลนำเข้า โดยข้อมูลที่ได้ผ่านการทำความสะอาดแล้วจะถูกแปลงให้อยู่ในรูปแบบของข้อมูลที่พร้อมจะถูกวิเคราะห์ ซึ่งภายในขั้นตอนนี้เป็นขั้นตอนที่สำคัญมาก เนื่องจากต้องการความถูกต้องเพื่อให้อยู่ในรูปแบบที่อำนวยต่อการทำงานสำหรับการค้นหาความรู้ในฐานข้อมูลและนำไปสู่ความรู้ที่ตั้งเป้าหมายไว้อย่างถูกต้อง

2.4.4 การทำเหมืองข้อมูล (Data Mining)

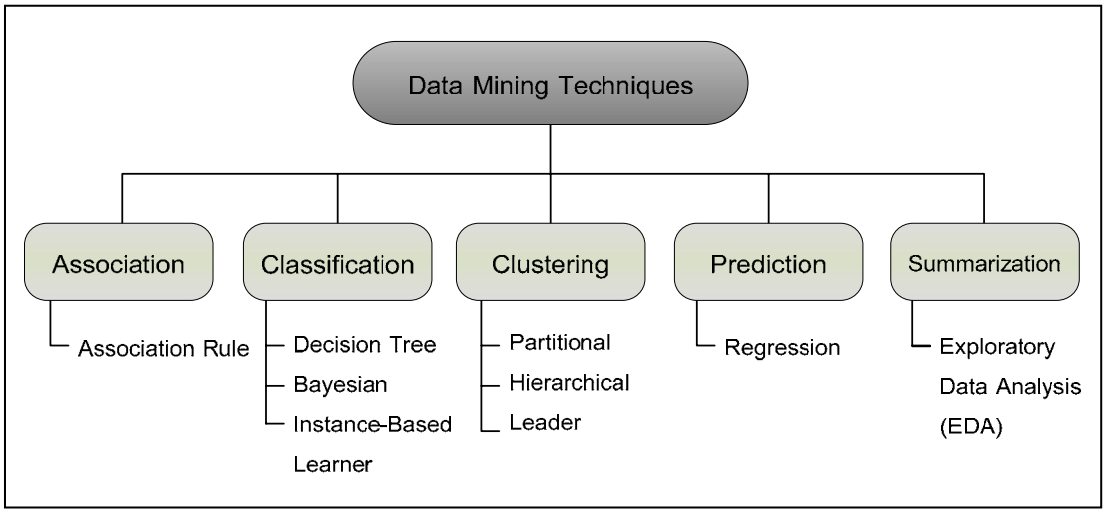
การทำเหมืองข้อมูล จะประกอบด้วยขั้นตอนของการเลือกเทคนิคของการทำเหมืองข้อมูล (Data Mining Techniques) เป็นขั้นตอนที่สำคัญ เนื่องจากเทคนิคที่เลือกนั้นต้องคำนึงถึงลักษณะข้อมูลที่เรามีอยู่และผลลัพธ์หรือข้อมูลนำออกที่เราต้องการว่ามีรูปแบบอย่างไร เพื่อที่จะเลือกใช้เทคนิคที่เหมาะสมในการค้นหาความรู้ในฐานข้อมูลตามที่ได้วางแผนไว้ เมื่อเลือกเทคนิคของการทำเหมืองข้อมูลแล้ว ขั้นตอนต่อไปคือการเลือกขั้นตอนวิธี (Selecting Algorithm) ที่จะใช้ในการค้นหาความรู้ในฐานข้อมูล โดยการเลือกขั้นตอนวิธีนั้นต้องคำนึงถึงลักษณะของข้อมูลนำเข้า เวลาที่ใช้ในการประมวลผล หรือแม้กระทั่งทรัพยากรที่มีอยู่ เพื่อให้ได้ขั้นตอนวิธีที่เหมาะสมและไม่ก่อให้เกิดปัญหาในกระบวนการของการประมวลผลเพื่อใช้ในการวิเคราะห์ ยกตัวอย่างเช่น หากต้องการความรู้ที่อยู่ในรูปของกฎความสัมพันธ์ อาจเลือกใช้เทคนิคการทำเหมืองข้อมูลที่เรียกว่า ความสัมพันธ์ (Association) และเลือกขั้นตอนวิธี Apriori (Apriori Algorithm) ในการประมวลผลเพื่อให้ได้กฎความสัมพันธ์ เป็นต้น

2.4.5 การแปลความหมายของผลลัพธ์ (Data Interpretation)

ข้อมูลนำออกหรือผลลัพธ์ที่ได้หลังจากการประมวลผลในขั้นตอนการทำเหมืองข้อมูลนั้นอาจอยู่ในรูปที่ไม่สามารถเข้าใจได้ง่าย จึงต้องมีการแปลความหมายหรือตีความของข้อมูลนำออกที่ได้นั้นให้อยู่ในรูปของความรู้ที่สามารถเข้าใจและนำไปใช้ได้ง่าย จึงต้องมีขั้นตอนของการแปลความหมายข้อมูล โดยข้อมูลที่แปลความหมายแล้วอาจอยู่ในรูปของตาราง กราฟ หรือรูปภาพ เป็นต้น ซึ่งหลังจากในขั้นตอนนี้แล้วผลลัพธ์ที่ได้มีความผิดพลาดหรือไม่ตรงกับเป้าหมายที่ตั้งไว้ก็อาจย้อนกลับไปทำงานในหัวข้อที่ 2.4.1 ถึง 2.4.5 เพื่อให้ได้ความรู้ที่ถูกต้องและมีประโยชน์มากที่สุด และเมื่อทำการแปลข้อมูลนำออกให้อยู่ในรูปแบบที่พร้อมนำไปใช้งานแล้ว ในขั้นตอนนี้ก็สามารถนำความรู้ที่ได้ไปใช้ให้เกิดประโยชน์ต่อไป โดยความรู้ที่ได้อาจนำไปใช้ในการสนับสนุนการตัดสินใจ พัฒนาองค์กรให้มีคุณภาพ หรือสร้างผลิตผลิตภัณฑ์ใหม่ๆ จากความรู้ที่ได้ เป็นต้น

2.5 เทคนิควิธีการทำเหมืองข้อมูล (Data Mining Techniques)

การเลือกเทคนิควิธีการทำเหมืองข้อมูลเป็นขั้นตอนที่สำคัญ (Mitra and Acharya, 2003) เนื่องจากการทำเหมืองข้อมูลมีวิธีการต่างๆ ที่ใช้ในการแก้ปัญหาอยู่หลายวิธีการ ซึ่งจะไม่มีวิธีการใดเลยที่สามารถแก้ปัญหาของทำเหมืองข้อมูลได้ทุกปัญหา ดังนั้นความหลากหลายของวิธีการเป็นสิ่งจำเป็นที่จะนำไปสู่การแก้ปัญหาที่ดีที่สุดของการทำเหมืองข้อมูล เพื่อให้ได้ความรู้ตามที่ต้องการจากการค้นหาความรู้ในฐานข้อมูลเหล่านั้น โดยสามารถสรุปเทคนิควิธีการทำเหมืองข้อมูลดังภาพประกอบ 2.2 พร้อมทั้งอธิบายได้ดังนี้



ภาพประกอบ 2.2 เทคนิควิธีการทำเหมืองข้อมูล

2.5.1 ความสัมพันธ์ (Association)

การค้นหาความสัมพันธ์ของข้อมูลเป็นเทคนิคหนึ่งของการทำเหมืองข้อมูล โดยหลักการทำงาน คือ การหาความสัมพันธ์ของข้อมูลภายในกลุ่มข้อมูล เพื่อใช้ลักษณะของข้อมูลหนึ่งในการบอกถึงลักษณะที่จะเกิดขึ้นกับข้อมูลอีกตัวหนึ่ง ซึ่งอาจจะเป็นการหาความสัมพันธ์ของข้อมูลในกลุ่มเดียวกัน ยกตัวอย่างเช่น การระบุว่าในกลุ่มของลูกค้าที่ซื้อน้ำอัดลมแล้วจะซื้อขนมคบเขี้ยวด้วยนั้นมีโอกาสเกิดขึ้นร่วมกัน หรืออาจจะเป็นการหาความสัมพันธ์ของตัวแปรระหว่างกลุ่มข้อมูล ยกตัวอย่างเช่น ในทุกๆ ครั้งที่ฝนตกหนักขึ้นจะทำให้มีปริมาณน้ำในแม่น้ำเพิ่มขึ้น เป็นต้น โดยลักษณะของการหาความสัมพันธ์นั้นอาจแบ่งได้เป็น 3 กลุ่ม ดังนี้

1) การหาความสัมพันธ์ระหว่างข้อมูล (Association Discovery) ยกตัวอย่างเช่น การวิเคราะห์หาความสัมพันธ์ของการซื้อสินค้าของลูกค้า (Market Basket Analysis) เพื่อใช้ในวางแผนสำหรับการจัดทำโปรโมชั่น (Liao and Chen, 2004) หรือการพัฒนาสินค้าให้ตรงตามความต้องการของลูกค้า (Liao *et al*., 2003) เป็นต้น

2) การหาความสัมพันธ์ในลักษณะที่เป็นลำดับของข้อมูล (Sequential Pattern Discovery) ยกตัวอย่างเช่น การระบุความเกี่ยวเนื่องกันของการซื้อสินค้าของลูกค้า โดยมีจุดมุ่งหมายที่จะเข้าใจถึงพฤติกรรมการซื้อสินค้าของลูกค้าในลักษณะระยะยาว (Long Term) ยกตัวอย่างเช่น ลูกค้าที่ซื้อเครื่องคอมพิวเตอร์ไปแล้วมีแนวโน้มที่จะกลับมาซื้อเครื่องพิมพ์ในเวลาต่อมา

3) การหาความสัมพันธ์ของข้อมูลกับช่วงเวลาใดๆ (Similar Time Sequence Discovery) ใช้สำหรับการค้นหาความเกี่ยวเนื่องกันระหว่างกลุ่มของข้อมูล 2 กลุ่มที่มีผลต่อกัน โดยจะมีช่วงเวลาเข้ามาเกี่ยวข้องด้วย ยกตัวอย่าง เช่น เมื่อใดก็ตามที่ยอดขายสินค้าเครื่องดื่มแอลกอฮอล์สูงขึ้น ยอดขายอาหารคบเขี้ยวจะสูงขึ้นตาม

ผลลัพธ์ที่ได้จากการค้นหาความสัมพันธ์นั้นสามารถนำไปใช้ในการวิเคราะห์สนับสนุนหรือพัฒนางานที่เกี่ยวข้องกับฐานข้อมูลนั้นๆ ได้อย่างมีประสิทธิภาพมากยิ่งขึ้น ซึ่งวิธีที่ได้รับความนิยมและเป็นที่รู้จัก คือ การค้นหากฎความสัมพันธ์ โดยขั้นตอนสำคัญสำหรับการค้นหากฎความสัมพันธ์นั้น คือ การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

2.5.2 การจัดหมวดหมู่ (Classification)

การจัดหมวดหมู่ข้อมูล คือ การจัดกลุ่มให้กับแต่ละข้อมูลในฐานข้อมูล เป็นการแบ่งกลุ่มข้อมูลในฐานข้อมูลตามลักษณะเด่นของหมวดหมู่นั้นๆ โดยมีการระบุค่าหรือลักษณะที่เป็นไปได้ของข้อมูลภายในแต่ละกลุ่ม การจัดหมวดหมู่จะให้ผลลัพธ์เป็นค่าที่แน่นอน ยกตัวอย่างเช่น การจัดกลุ่มของผู้ป่วยตามผลของการใช้ยารักษาเพื่อระบุรูปแบบการรักษาให้กับผู้ป่วยใหม่ที่เข้ารับการรักษา เป็นต้น โดยวิธีการที่เป็นที่รู้จักอย่างแพร่หลายสำหรับการจัดหมวดหมู่ คือ ต้นไม้ตัดสินใจ (Decision Tree) เนื่องจากผลลัพธ์ที่ได้จากการจัดหมวดหมู่สามารถทำความเข้าใจได้ง่าย

2.5.3 การรวมกลุ่ม (Clustering)

การรวมกลุ่ม คือ การแบ่งข้อมูลเป็นแบบกลุ่มๆ โดยการรวมกลุ่มข้อมูลดังกล่าวนั้นได้จากการพิจารณาคุณสมบัติในหลายๆมิติของข้อมูล ซึ่งหากถ้ารายการในข้อมูลมีลักษณะคล้ายคลึงกันเป็นกลุ่มเดียวกันได้ก็จะรวมเข้าด้วยกัน หลักการทำงานของการรวมกลุ่ม คือ พยายามมองหาความเหมือนและความแตกต่างภายในกลุ่มของข้อมูลและแบ่งกลุ่มต่าง ๆ ออกเป็นส่วน ๆ ยกตัวอย่างเช่น หากองค์กรต้องการทราบความเหมือนที่มีในกลุ่มลูกค้าของตนเอง เพื่อจะสามารถเข้าใจลักษณะเฉพาะของกลุ่มลูกค้าเป้าหมายขององค์กร และสามารถสร้างกลุ่มของลูกค้าเพื่อที่องค์กรจะสามารถขายสินค้าได้ในอนาคตนั้น องค์กรก็สามารถใช้วิธีการของการรวมกลุ่มเพื่อทำการแยกกลุ่มของลูกค้าออกเป็นกลุ่มๆ ได้ เป็นต้น โดยวิธีการที่เป็นที่รู้จัก คือ การรวมกลุ่มแบบ *K*-means (*K*-means Clustering) การทำงานของ *K*-means

จะเป็นการกำหนดเพียงจำนวนกลุ่มที่ต้องการเท่านั้น แล้วผลลัพธ์ที่ได้ข้อมูลจะถูกแบ่งออกเป็นกลุ่ม ๆ ตามจำนวนกลุ่มที่กำหนด

2.5.4 การทำนายล่วงหน้า (Prediction)

การทำนายล่วงหน้าเป็นเทคนิคที่เหมือนกับการจัดหมวดหมู่ แต่มีความแตกต่างตรงที่ว่าการทำนายล่วงหน้านั้น รายการข้อมูลที่ถูกแยกเพื่อจัดลำดับนั้น เกิดขึ้นตามลักษณะการทำนายพฤติกรรมในอนาคตหรือการทำนายค่าที่จะเกิดขึ้นในอนาคต ข้อมูลในอดีตจะถูกสร้างเป็นต้นแบบ (Model) ขึ้นมาเพื่อทำนายหรืออธิบายสิ่งที่จะเกิดขึ้นในอนาคต ตัวอย่างเช่นการทำนายว่าหน้าร้อนในปีหน้าอุณหภูมิสูงสุดจะเป็นกี่องศาเซลเซียส (Degree Celsius) หรือปริมาณน้ำในแม่น้ำจะเพิ่มขึ้นเป็นเท่าใดหากฝนตกหนักขึ้น 10% เป็นต้น โดยวิธีการหนึ่งที่ใช้ในการทำนายล่วงหน้า คือ สมการถดถอย (Regression) หลักการทำงานจะใช้ความรู้ทางสถิติมาช่วยในการสร้างสมการขึ้นมาจากข้อมูลที่มีอยู่ ผลลัพธ์ที่ได้จะอยู่ในรูปของสมการเพื่อใช้ในการทำนาย

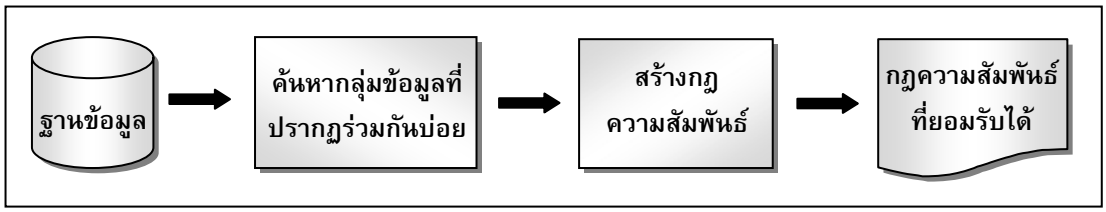
2.5.5 การสรุป (Summarization)

การสรุป คือ การนำข้อมูลที่มีอยู่ในฐานข้อมูลมาสรุปให้อยู่ในรูปแบบที่สามารถเข้าใจถึงลักษณะของข้อมูลให้ง่ายขึ้นโดยใช้ความรู้ทางสถิติเข้ามาผสมผสานกับกราฟิก (Graphic) เช่น กราฟที่อยู่ในรูปของสามมิติเพื่อให้สามารถเห็นลักษณะ แนวโน้ม หรือการกระจายตัวของข้อมูลได้อย่างชัดเจน บางครั้งอาจใช้เทคนิคการสรุปเพื่อตรวจสอบความถูกต้องหรือการสูญหายของข้อมูลก่อนการค้นหาความรู้ในฐานข้อมูล โดยวิธีการที่ใช้ในการสรุปนั้นยกตัวอย่างเช่น วิธีการ Exploratory Data Analysis (EDA)

2.6 การค้นหากฎความสัมพันธ์ (Association Rules)

การค้นหากฎความสัมพันธ์ของข้อมูลในฐานข้อมูล (Rygielski *et al*., 2002; Agrawal *et al*., 1993; Berberidis *et al*., 2005) ได้มีการพัฒนาขึ้นครั้งแรกโดยนักวิจัยจากศูนย์วิจัย IBM (International Business Machines Corporation) ประเทศสหรัฐอเมริกา มีจุดประสงค์เพื่อการค้นหาความสัมพันธ์ที่น่าสนใจซึ่งซ่อนอยู่ในข้อมูล โดยการนำไปใช้ที่เห็นได้ชัดเจนสำหรับการค้นหาความสัมพันธ์ คือ การค้นหากฎความสัมพันธ์ของการซื้อสินค้าของลูกค้าว่าจะซื้อสินค้าใดบ้างร่วมกันในรถเข็นในซูเปอร์มาร์เก็ต (Market Basket Analysis) เพื่อทำความเข้าใจถึงพฤติกรรมการซื้อสินค้าของลูกค้า ยกตัวอย่างเช่น เมื่อลูกค้าซื้อนมแล้วจะซื้อขนมปังด้วย เป็นต้น การค้นหากฎความสัมพันธ์นั้นมีขั้นตอนการทำงานที่สำคัญที่สุด คือ การ

ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยซึ่งจะใช้เวลาในการทำงานนานที่สุด ดังนั้นในการเลือกขั้นตอนวิธีจึงต้องเลือกขั้นตอนวิธีที่เหมาะสมกับลักษณะของข้อมูลในฐานข้อมูล เพื่อลดระยะเวลาและเนื้อที่หน่วยความจำในการทำงาน โดยทั่วไปการค้นหากฎความสัมพันธ์สามารถแสดงดังภาพประกอบ 2.3 แบ่งการทำงานออกเป็น 2 ขั้นตอน คือ ขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด โดยหาได้จากกลุ่มข้อมูลในฐานข้อมูลที่เกิดขึ้นร่วมกันในแต่ละรายการข้อมูล โดยกลุ่มข้อมูลเหล่านั้นจะต้องมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำที่ผู้ใช้กำหนดจึงจะถือว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย และขั้นตอนการนำกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยมาสร้างกฎความสัมพันธ์ โดยกฎความสัมพันธ์จะเป็นที่ยอมรับได้หากตรวจสอบค่าความเชื่อมั่นของกฎนั้นแล้วมีค่ามากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำที่ผู้ใช้กำหนด



ภาพประกอบ 2.3 กระบวนการค้นหากฎความสัมพันธ์

2.7 การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets)

การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเป็นขั้นตอนที่สำคัญสำหรับการค้นหากฎความสัมพันธ์ เนื่องจากต้องค้นหากลุ่มข้อมูลที่อาจเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพื่อนำไปสร้างกฎความสัมพันธ์เป็นจำนวนมาก ซึ่งขั้นตอนนี้จะใช้เวลานานและเนื้อที่ในหน่วยความจำนวนมาก จึงทำให้ต้องมีการเลือกขั้นตอนวิธีที่เหมาะสมกับลักษณะข้อมูลในฐานข้อมูล ทรัพยากรของเครื่องคอมพิวเตอร์ที่มีอยู่ หรือผลลัพธ์ของข้อมูลที่ต้องการจากการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

2.8 ตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

จากฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 แต่ละตัวอักษรภาษาอังกฤษหมายถึง ชิ้นข้อมูลที่ใช้แทนชื่อสินค้าที่ลูกค้าซื้อในแต่ละรายการข้อมูลซึ่งมีทั้งหมด 5 รายการข้อมูล การค้นหากฎความสัมพันธ์จะเริ่มจากการนับความถี่ของชิ้นข้อมูลแต่ละชิ้นในฐานข้อมูลเพื่อหาค่าสนับสนุนว่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำหรือไม่ และเพื่อหากลุ่มข้อมูลที่

ปรากฏร่วมกันบ่อยต่อไป ยกตัวอย่างเช่น มีการซื้อสินค้า C ทั้งหมด 4 รายการข้อมูล กล่าวได้ว่ามีจำนวนความถี่เท่ากับ 4 ซึ่งหากกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (40%) สามารถคำนวณค่าสนับสนุนได้จากจำนวนความถี่ของสินค้า C หารด้วยจำนวนของรายการข้อมูลทั้งหมดในฐานข้อมูล ดังสมการที่ (2.3)

$$\text{ค่าสนับสนุน } C = \frac{4}{5} = 0.8 \qquad (2.3)$$

จะได้ว่า สินค้า C เป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย เนื่องจากมีค่าสนับสนุนเป็น 0.8 (80%) ซึ่งมากกว่าค่าสนับสนุนขั้นต่ำที่กำหนดไว้ คือ 0.4 (40%)

ตารางที่ 2.1 ฐานข้อมูลการซื้อสินค้า

รหัสรายการ (TID)	ชิ้นข้อมูล (Items)
001	C D E F G I
002	A C D E L
003	A B D E G
004	A C D H
005	A C D J

2.9 ตัวอย่างการสร้างกฎความสัมพันธ์

การสร้างกฎความสัมพันธ์นั้น จำนวนของชิ้นข้อมูลที่ใช้จะต้องประกอบด้วย 2 ชิ้นข้อมูลขึ้นไปในการสร้างกฎความสัมพันธ์ ยกตัวอย่างเช่น สินค้า A และ D ที่ซื้อร่วมกันมีทั้งหมด 4 รายการข้อมูล กล่าวได้ว่ามีจำนวนความถี่เท่ากับ 4 จะได้ว่า สินค้า A และ D เป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย สามารถนำมาสร้างกฎความสัมพันธ์ได้ ยกตัวอย่างเช่น หากลูกค้าซื้อสินค้า A แล้วจะซื้อสินค้า D ด้วย ($A \rightarrow D$) หรือหากลูกค้าซื้อสินค้า D แล้วจะซื้อสินค้า A ด้วย ($D \rightarrow A$) โดยคำนวณค่าความเชื่อมั่นได้ดังสมการ (2.4) และ (2.5) ตามลำดับ

$$\text{ค่าความเชื่อมั่นของกฎ } A \rightarrow D = \frac{4}{4} = 1.0 \qquad (2.4)$$

$$\text{ค่าความเชื่อมั่นของกฎ } D \rightarrow A = \frac{4}{5} = 0.8 \qquad (2.5)$$

แต่กฎดังกล่าวที่สร้างขึ้นจะเป็นกฎความสัมพันธ์ที่สามารถยอมรับได้ก็ต่อเมื่อมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำที่กำหนดไว้ หากกำหนดค่าความเชื่อมั่นขั้นต่ำเป็น 0.5 (50%) ซึ่งกฎ $A \rightarrow D$ และกฎ $D \rightarrow A$ มีค่าความเชื่อมั่นไม่น้อยกว่าค่าความเชื่อมั่นขั้นต่ำที่กำหนดไว้ นั่นคือ 1.0 และ 0.8 ตามลำดับ ดังนั้นจะได้กฎความสัมพันธ์ที่สามารถยอมรับได้ว่า

หากลูกค้าซื้อสินค้า A แล้วมีโอกาสที่จะซื้อสินค้า D ด้วยมีความเชื่อมั่นเป็น 100% หรือหากลูกค้าซื้อสินค้า D แล้วมีโอกาสที่จะซื้อสินค้า A ด้วยมีความเชื่อมั่นเป็น 80%

บทที่ 3

ขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

ขั้นตอนวิธีที่ใช้สำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในการทำเหมืองข้อมูลนั้นมีอยู่หลากหลายวิธี โดยแต่ละขั้นตอนวิธีมีวิธีการในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่แตกต่างกันไปตามวัตถุประสงค์ของการสร้างขั้นตอนวิธีนั้นขึ้นมา ยกตัวอย่างเช่น เพื่อลดจำนวนของการอ่านข้อมูลจากฐานข้อมูล เพื่อลดระยะเวลาในขั้นตอนการค้นหากลุ่มข้อมูล หรือเพื่อลดเนื้อที่หน่วยความจำที่ใช้ในระหว่างการประมวลผล เป็นต้น โดยในหัวข้อนี้จะอธิบายถึงลักษณะการทำงานของขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ได้รับความนิยมและเป็นที่รู้จัก รวมถึงขั้นตอนวิธีที่มีส่วนเกี่ยวข้องสำหรับงานวิจัยในวิทยานิพนธ์เล่มนี้ ซึ่งจะประกอบด้วยขั้นตอนวิธี Apriori ขั้นตอนวิธี FP-Growth ขั้นตอนวิธี BitTableFI และขั้นตอนวิธีสุดท้ายคือ ขั้นตอนวิธี Index-BitTable

3.1 ขั้นตอนวิธี Apriori (Apriori Algorithm)

ขั้นตอนวิธี Apriori (Agrwal and Srikant, 1993) เป็นขั้นตอนวิธีพื้นฐานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยและสร้างกฎความสัมพันธ์ เป็นขั้นตอนวิธีที่ได้รับการยอมรับและได้รับความนิยมเป็นอย่างมาก อีกทั้งขั้นตอนวิธี Apriori ยังเป็นขั้นตอนวิธีที่มีอิทธิพลต่อการศึกษาและพัฒนาขั้นตอนวิธีอื่นๆ

3.1.1 หลักการทำงานของขั้นตอนวิธี Apriori

หลักการทำงานของขั้นตอนวิธี Apriori โดยหลักๆ แล้วจะประกอบด้วยขั้นตอนการทำงานทั้งหมด 2 ขั้นตอนด้วยกัน ขั้นตอนแรกคือการสร้างกลุ่มข้อมูลท้าชิง และขั้นตอนที่สองคือการทดสอบกลุ่มข้อมูลท้าชิงเหล่านั้นว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ แสดงดังภาพประกอบ 3.1 โดยการทำงานของขั้นตอนวิธี Apriori สามารถอธิบายได้ดังนี้

1) อ่านชิ้นข้อมูลจากฐานข้อมูลครั้งแรกเพื่อนับค่าความถี่ของแต่ละชิ้นข้อมูลที่ปรากฏทั้งหมดในฐานข้อมูล

2) ตรวจสอบค่าความถี่ของแต่ละชิ้นข้อมูล เพื่อคำนวณค่าสนับสนุนโดยหากชิ้นข้อมูลนั้นๆ มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำก็จะถือว่า

เป็น กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล 1 ชิ้นข้อมูล (L_1: Frequent 1-Itemsets)

3) นำ L_1 ที่ได้มาสร้างกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล 2 ชิ้นข้อมูล (C_2: Candidate 2-Itemsets)

4) อ่านชิ้นข้อมูลจากฐานข้อมูลอีกครั้งเพื่อนับค่าความถี่ของ C_2 และตัด C_2 ที่มีค่าสนับสนุนน้อยกว่าค่าสนับสนุนขั้นต่ำ โดยหาก C_2 มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำก็จะกลายเป็น L_2

5) ทำในหัวข้อที่ 3) และ 4) ซ้ำจนกว่าไม่สามารถสร้าง C_k จาก L_{k-1} ได้ เมื่อ k คือขนาดของชิ้นข้อมูล จึงสิ้นสุดการสร้างกลุ่มข้อมูลท้าชิงและจบการทำงานของขั้นตอนวิธี Apriori ทำให้ได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด

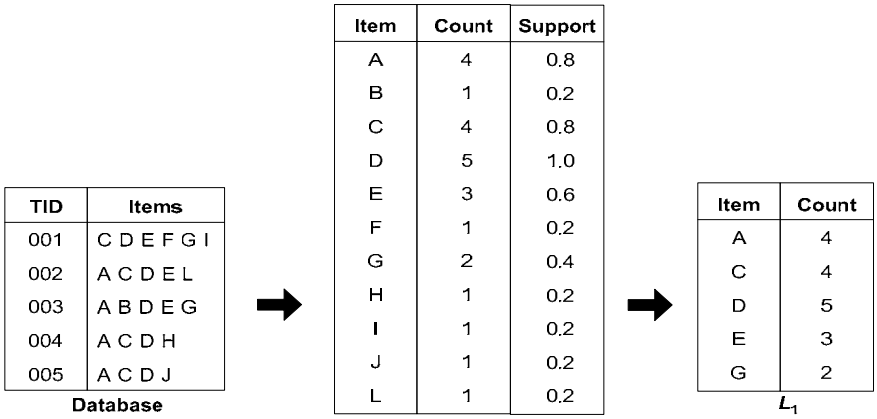
```
1   L1 = {large 1-itemsets};
2   For (k=2; Lk-1≠∅; k++) do begin
3       Ck = apriori-gen(Lk-1);    // New candidates
4       Forall transactions t ∈ D do begin
5           Ct = subset(Ck , t);    // Cadidates contained in t
6           Forall candidates c ∈ Ct do
7               c.count++;
8           End
9       L1 = { c ∈ Ck | c.count ≥ minsup};
10  End
11  Answer = Uk Lk;
```

ภาพประกอบ 3.1 ขั้นตอนการทำงานของขั้นตอนวิธี Apriori (Agrawal and Srikant, 1993)

3.1.2 ตัวอย่างการทำงานของขั้นตอนวิธี Apriori

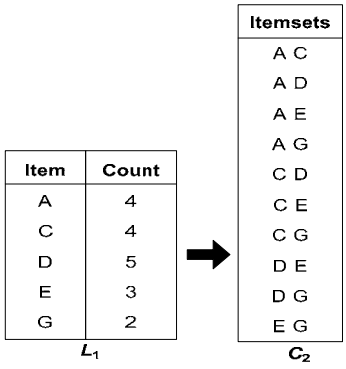
เพื่อให้เข้าใจถึงกระบวนการทำงานของขั้นตอนวิธี Apriori ได้อย่างชัดเจนขึ้น จึงขอยกตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธีนี้โดยใช้ฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 ประกอบการอธิบาย พร้อมทั้งกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (มีค่าความถี่อย่างน้อยเท่ากับ 2 รายการข้อมูลปรากฏในฐานข้อมูล) โดยตัวอย่างการทำงานของขั้นตอนวิธี Apriori สามารถอธิบายได้ดังนี้

1) อ่านชิ้นข้อมูลจากฐานข้อมูลเพื่อนับค่าความถี่ของแต่ละชิ้นข้อมูล และคำนวณค่าสนับสนุนจากค่าความถี่ที่ได้ พร้อมทั้งตัดชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำที่กำหนดไว้ ซึ่งผลลัพธ์ที่ได้จะเรียกว่า L_1 แสดงดังภาพประกอบ 3.2



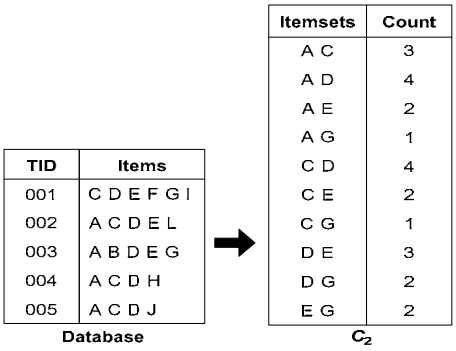
ภาพประกอบ 3.2 การอ่านข้อมูลจากฐานข้อมูลเพื่อค้นหา L_1 ของขั้นตอนวิธี Apriori

2) นำ L_1 ที่ได้ไปสร้างกลุ่มข้อมูลท้าชิง C_2 โดยวิธีการสร้าง C_2 นั้นจะจับคู่ร่วมกัน (Join) ของชิ้นข้อมูลทุกตัวที่เป็น L_1 ทุกกรณีที่เป็นไปได้ที่ประกอบด้วยชิ้นข้อมูล 2 ชิ้นข้อมูล ดังภาพประกอบ 3.3



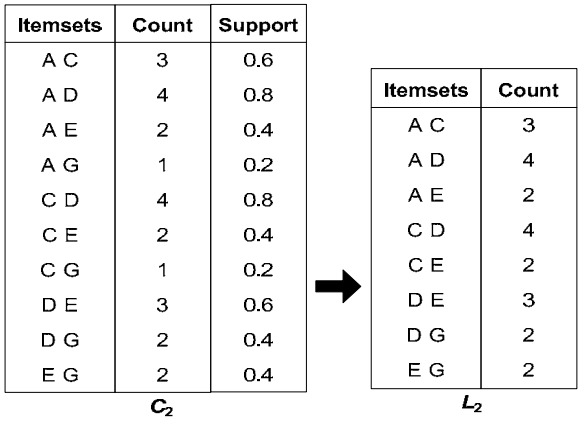
ภาพประกอบ 3.3 การสร้างกลุ่มข้อมูลท้าชิง C_2 ของขั้นตอนวิธี Apriori

3) หลังจากสร้างกลุ่มข้อมูลท้าชิง C_2 แล้วนั้น ขั้นตอนต่อไปคืออ่านชิ้นข้อมูลจากฐานข้อมูลเพื่อนับค่าความถี่ของ C_2 แสดงดังภาพประกอบ 3.4



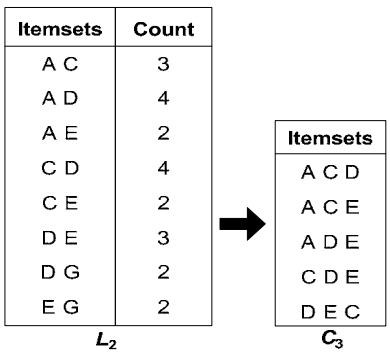
ภาพประกอบ 3.4 การนับค่าความถี่ของข้อมูลท้าชิง C_2 ของขั้นตอนวิธี Apriori

4) นำค่าความถี่ของ C_2 ที่ได้ไปคำนวณหาค่าสนับสนุนเพื่อใช้ในการตัดสินใจให้เป็นกลุ่มข้อมูล L_2 ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 3.5



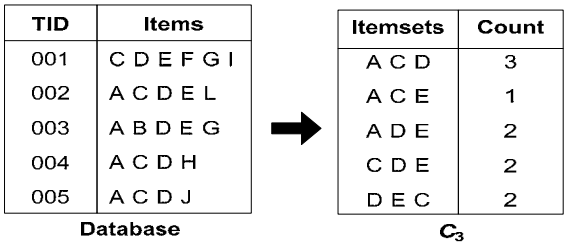
ภาพประกอบ 3.5 ผลลัพธ์การค้นหากลุ่มข้อมูล L_2 ของขั้นตอนวิธี Apriori

5) นำกลุ่มข้อมูล L_2 ที่ได้ไปสร้างกลุ่มข้อมูลท้าชิง C_3 โดยวิธีการสร้างจะจับคู่ร่วมกันของกลุ่มข้อมูลทุกตัวที่เป็น L_2 ที่มีชิ้นข้อมูลตัวแรกเหมือนกัน ซึ่งผลลัพธ์ของการจับคู่จะประกอบด้วยชิ้นข้อมูล 3 ชิ้นข้อมูล ดังภาพประกอบ 3.6



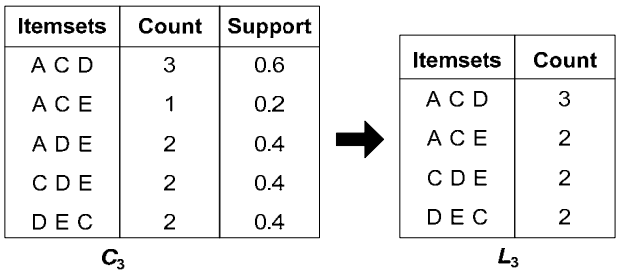
ภาพประกอบ 3.6 การสร้างกลุ่มข้อมูลท้าชิง C_3 ของขั้นตอนวิธี Apriori

6) หลังจากสร้างกลุ่มข้อมูลท้าชิง C_3 แล้วนั้น ขั้นตอนต่อไป คือ อ่านชิ้นข้อมูลจากฐานข้อมูลเพื่อนับค่าความถี่ของ C_3 แสดงดังภาพประกอบ 3.7



ภาพประกอบ 3.7 การนับค่าความถี่ของกลุ่มข้อมูลท้าชิง C_3 ของขั้นตอนวิธี Apriori

7) คำนวณหาค่าสนับสนุนจากค่าความถี่ที่ได้ พร้อมทั้งตัดกลุ่มข้อมูลท้าชิงC_3 ที่ไม่ผ่านค่าสนับสนุนขั้นต่ำ ซึ่งผลลัพธ์ที่ได้จะเรียกว่า L_3 แสดงดังภาพประกอบ 3.8



ภาพประกอบ 3.8 ผลลัพธ์การค้นหากลุ่มข้อมูล L_2 ของขั้นตอนวิธี Apriori

8) นำ L_3 ที่ได้ไปสร้าง C_4 โดยวิธีการสร้าง C_4 นั้นจะจับคู่ร่วมกันของกลุ่มข้อมูลทุกตัวที่เป็น L_3 กับตัวมันเองที่มีชิ้นข้อมูลสองตัวแรกเหมือนกัน ซึ่งผลลัพธ์ของการจับคู่จะประกอบด้วยชิ้นข้อมูล 4 ชิ้นข้อมูล แต่จากภาพปรกอบ 3.8 จะเห็นได้ว่าเมื่อได้ L_3 แล้วไม่สามารถสร้าง C_4 จาก L_3 ได้ เนื่องจากไม่มีกลุ่มข้อมูลใดเลยใน L_3 ที่มีชิ้นข้อมูล 3 ชิ้นข้อมูลแรกเหมือนกัน จึงหยุดการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ดังนั้นผลลัพธ์ที่ได้ทั้งหมดแสดงดังตาราง 3.1 โดยตัวเลขที่อยู่หลังเครื่องหมาย ":" ในตารางหมายถึงค่าความถี่ของกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นๆ

ตารางที่ 3.1 ผลลัพธ์การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Apriori

Level	Frequent Itemsets
L_1	A:4, C:4, D:5, E:3, G:2
L_2	AC:3, AD:4, AE:2, CD:4, CE:2, DE:3, DG:2, EG:2
L_3	ACD:3, ADE:2, CDE:2, DEG:2

3.1.3 ข้อดีของขั้นตอนวิธี Apriori

ขั้นตอนวิธี Apriori สามารถทำงานได้ดีหากกำหนดค่าสนับสนุนขั้นต่ำมีค่ามากๆ มีขนาดของฐานข้อมูลเล็ก และมีจำนวนของชิ้นข้อมูล L_1 น้อย อีกทั้งกระบวนการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอน Apriori นั้นมีลักษณะที่ง่ายและไม่ซับซ้อน

3.1.4 ข้อเสียของขั้นตอนวิธี Apriori

การทำงานของขั้นตอนวิธี Apriori นั้นต้องอ่านข้อมูลจากฐานข้อมูลหลายครั้งเพื่อใช้ตรวจสอบกลุ่มข้อมูลท้าชิงซึ่งอาจก่อให้เกิดปัญหาคอขวด (Bottle Neck Problem) ขึ้นได้ และในระหว่างการประมวลผลต้องใช้เนื้อที่ในหน่วยความจำเป็นจำนวนมากสำหรับการสร้างกลุ่มข้อมูลท้าชิง และยังใช้เวลาในการประมวลผลนานหากข้อมูลในฐานข้อมูลมีอัตราส่วนของจำนวนชิ้นข้อมูลปรากฏในรายการข้อมูลมาก และขนาดของฐานข้อมูลมีขนาดใหญ่

3.2 ขั้นตอนวิธี FP-Growth (FP-Growth Algorithm)

เนื่องจากขั้นตอนวิธี Apriori มีการอ่านข้อมูลหลายครั้งและยังต้องสร้างกลุ่มข้อมูลท้าชิงจำนวนมากจึงทำให้มีการคิดค้นวิธีการเพื่อแก้ไขข้อบกพร่องนี้ Han และคณะ (Han *et al*., 2000) ได้พัฒนาขั้นตอนวิธีใหม่ขึ้นมาเพื่อลดจำนวนของการอ่านข้อมูลจากฐานข้อมูลพร้อมทั้งนำเสนอโครงสร้างข้อมูลแบบใหม่ขึ้นมาที่มีชื่อว่า FP-Tree โดยใช้ชื่อว่าขั้นตอนวิธี FP-Growth เป็นขั้นตอนวิธีที่อ่านข้อมูลจากฐานข้อมูลเพียง 2 ครั้งและไม่มีการสร้างกลุ่มข้อมูลท้าชิง เพื่อลดระยะเวลาในการประมวลผลให้สามารถทำงานได้เร็วขึ้น

3.2.1 หลักการทำงานของขั้นตอนวิธี FP-Growth

หลักการทำงานของขั้นตอนวิธี FP-Growth แสดงดังภาพประกอบ 3.9 ซึ่งขั้นตอนวิธี FP-Growth เป็นขั้นตอนวิธีที่มีลักษณะการค้นหากลุ่มข้อมูลที่ปรากฏบ่อยแบบการเติบโตอย่างเป็นรูปแบบ (Pattern Growth) โดยการทำงานของขั้นตอนวิธี FP-Growth สามารถอธิบายหลักการทำงานได้ดังนี้

1) อ่านข้อมูลจากฐานข้อมูลครั้งแรกเพื่อนับค่าความถี่ของแต่ละชิ้นข้อมูล แล้วนำชิ้นข้อมูลที่ไม่น้อยกว่าค่าสนับสนุนขั้นต่ำ (L_1) มาเรียงลำดับตามค่าความถี่ของแต่ละชิ้นข้อมูลจากมากไปหาน้อยแล้วนำมาสร้างตาราง Header (Header Table)

2) อ่านข้อมูลจากฐานข้อมูลครั้งที่สองเพื่อสร้างต้นไม้ FP-Tree โดยอ่านข้อมูลจากฐานข้อมูลทีละรายการข้อมูล จากนั้นตัดชิ้นข้อมูลในรายการข้อมูลนั้นที่ไม่ปรากฏอยู่ในตาราง Header ทิ้งไป แล้วเรียงชิ้นข้อมูลที่เหลือตามลำดับในตาราง Header แล้วนำชิ้นข้อมูลดังกล่าวไปสร้างโหนด (Node Tree) เพิ่มเข้าไปในต้นไม้ FP-Tree แล้วเชื่อมแต่ละโหนดที่เป็นชิ้นข้อมูลเดียวกันเพิ่มเข้าไปกับตาราง Header

3) สร้าง Conditional pattern base และสร้าง Conditional FP-Tree ของแต่ละชิ้นข้อมูล เพื่อใช้ในขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยการพิจารณาจะเริ่มจากชิ้นข้อมูลล่างสุดจนถึงชิ้นข้อมูลที่อยู่บนสุดในตาราง Header ซึ่ง Conditional pattern base หมายถึงเซตของชิ้นข้อมูลที่เกิดขึ้นพร้อมกับชิ้นข้อมูลที่กำลังพิจารณาในแต่ละเส้นทาง (Path Tree) และกำหนดให้ทุกชิ้นข้อมูลมีค่าความถี่เท่ากับค่าความถี่ของชิ้นข้อมูลที่กำลังพิจารณาจากต้นไม้ FP-Tree หลังจากนั้นสร้างต้นไม้ FP-Tree บน Conditional pattern base นี้ เรียกว่า Conditional FP-Tree ซึ่งเกิดจากการนำค่าความถี่ของแต่ละชิ้นข้อมูลในทุกเส้นทางมารวมกัน และเลือกเฉพาะชิ้นข้อมูลที่ผ่านค่าสนับสนุนขั้นต่ำจาก Conditional FP-Tree เพื่อนำไปสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยต่อไป

4) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากการสร้าง Conditional pattern base และสร้าง Conditional FP-Tree ของแต่ละชิ้นข้อมูล โดยใช้หลักการทำงานแบบแบ่งแยกแล้วเอาชนะ (Divide and Conquer)

Input: FP-tree constructed based on Algorithm 1, using *DB* and a minimum support threshold ξ.

Output: The complete set of frequent patterns.

Method: Call FP-tree (FP-tree, null).

Procedure FPF-tree (FP-tree, α)

{

1 **IF** Tree contains a single path *P*

2 **Then for each** combination (denoted as β) of the nodes in the path *P* **do**

3 generate pattern β U α with support = minimum support of node in β;

4 **Else for each** a_i in the header of Tree **do** **{**

5 generate pattern $\beta = a_i$ U α with support = a_i.support;

6 construct β's conditional pattern base and then β 's conditional FP-tree Tree_β;

7 **If** $\text{Tree}_\beta \neq \varnothing$

8 **Then** call FP-Tree (Tree_β, β) **}**

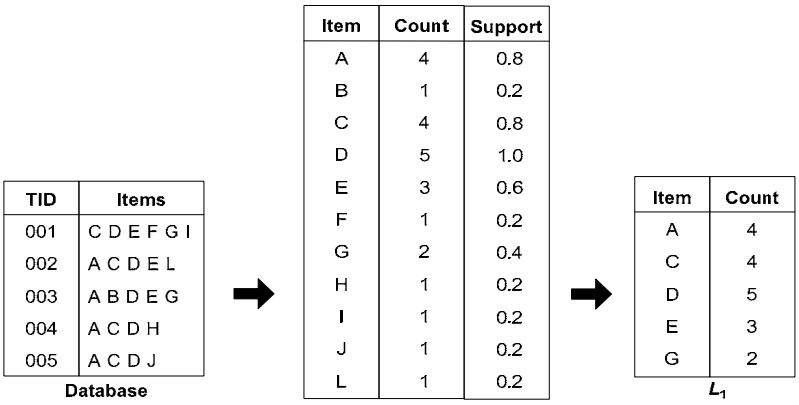
}

ภาพประกอบ 3.9 ขั้นตอนการทำงานของขั้นตอนวิธี FP-Growth (Han *et al*., 2000)

3.2.2 ตัวอย่างการทำงานของขั้นตอนวิธี FP-Growth

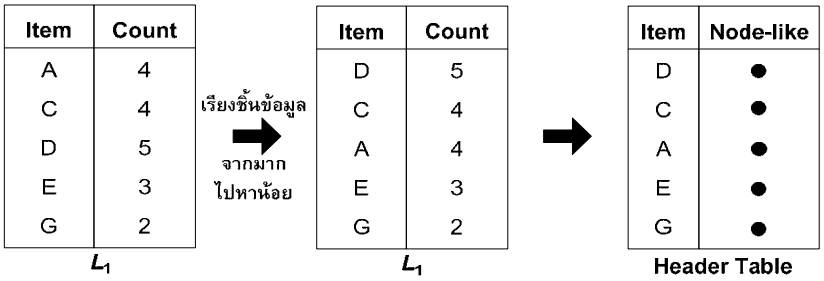
เพื่อให้เข้าใจถึงกระบวนการทำงานของขั้นตอนวิธี FP-Growth ได้อย่างชัดเจนขึ้น จึงขอยกตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธีนี้โดยใช้ฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 ประกอบการอธิบาย พร้อมทั้งกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (มีค่าความถี่อย่างน้อยเท่ากับ 2 รายการข้อมูลปรากฏในฐานข้อมูล) ดังนี้

1) อ่านข้อมูลจากฐานข้อมูลครั้งแรกเพื่อนับค่าความถี่ของแต่ละชิ้นข้อมูล และคำนวณค่าความถี่เพื่อกำจัดชิ้นข้อมูลที่ปรากฏไม่บ่อย ซึ่งผลลัพธ์ที่ได้คือ L_1 ดังภาพประกอบ 3.10



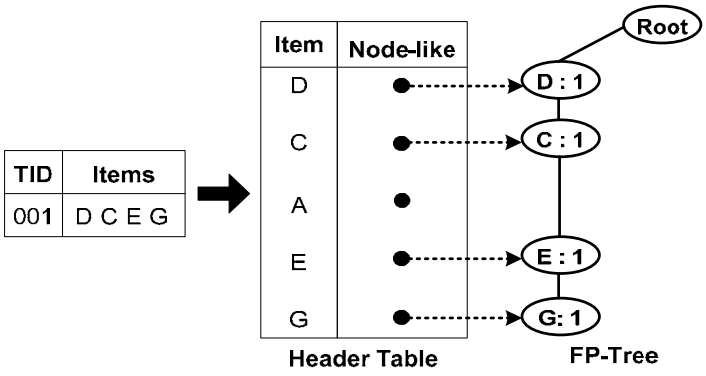
ภาพประกอบ 3.10 การอ่านข้อมูลจากฐานข้อมูลเพื่อค้นหา L_1 ของขั้นตอนวิธี FP-Growth

2) จัดเรียง L_1 ใหม่ตามค่าความถี่ของแต่ละชิ้นข้อมูลจากมากไปหาน้อยเพื่อสร้างตาราง Header ดังภาพประกอบ 3.11

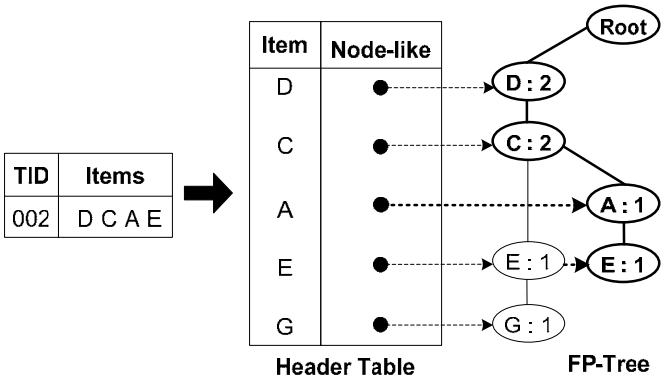


ภาพประกอบ 3.11 การสร้างตาราง Header ของขั้นตอนวิธี FP-Growth

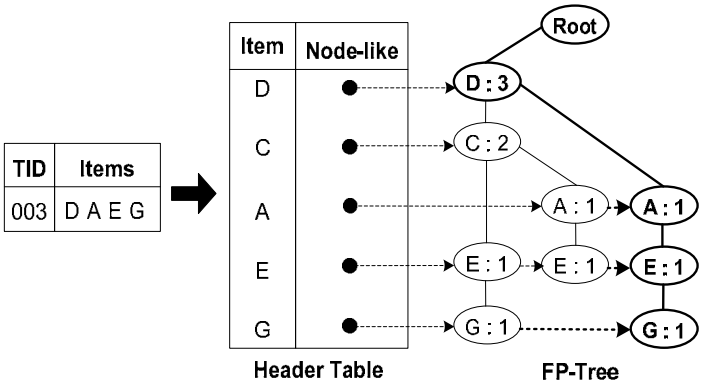
3) อ่านข้อมูลจากฐานข้อมูลครั้งที่สอง โดยเริ่มอ่านชิ้นข้อมูลทั้งหมดจากรายการข้อมูลแรกในฐานข้อมูลซึ่งประกอบด้วยชิ้นข้อมูล C D E F G และ I จากนั้นตัดชิ้นข้อมูลที่ไม่ปรากฏในตาราง Header ออกไป แล้วเรียงลำดับชิ้นข้อมูลที่เหลือใหม่ตามลำดับในตาราง Header จะได้ลำดับของชิ้นข้อมูลของรายการข้อมูลแรกคือ D C E และ G จากนั้นนำชิ้นข้อมูลที่ได้ไปสร้างโหนดของชิ้นข้อมูลเพิ่มเข้าไปในต้นไม้ FP-Tree แล้วเชื่อมแต่ละโหนดที่เพิ่มเข้าไปกับตาราง Header ผลลัพธ์แสดงดังภาพประกอบ 3.12 โดยตัวเลขที่อยู่หลังเครื่องหมาย “:” ในแต่ละโหนดหมายถึงค่าความถี่ของกลุ่มข้อมูล และทำตามขั้นตอนข้างต้นนี้กับทุกรายการข้อมูลในฐานข้อมูล จะสามารถแสดงผลลัพธ์ของการเพิ่มโหนดชิ้นข้อมูลของแต่ละรายการข้อมูลเข้าไปในต้นไม้ FP-Tree ดังภาพประกอบที่ 3.12 ถึง 3.16



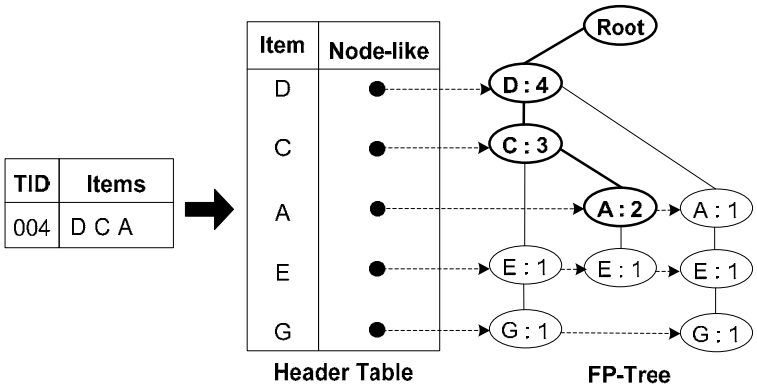
ภาพประกอบ 3.12 การอ่านรายการข้อมูลแรกจากฐานข้อมูลของขั้นตอนวิธี FP-Growth



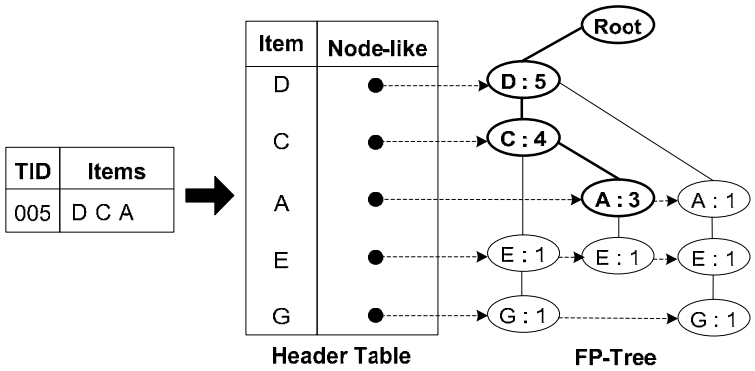
ภาพประกอบ 3.13 การอ่านรายการข้อมูลที่สองจากฐานข้อมูลของขั้นตอนวิธี FP-Growth



ภาพประกอบ 3.14 การอ่านรายการข้อมูลที่สามจากฐานข้อมูลของขั้นตอนวิธี FP-Growth

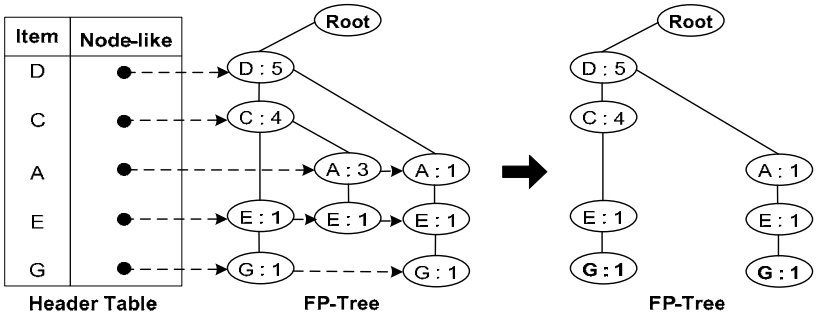


ภาพประกอบ 3.15 การอ่านรายการข้อมูลที่สี่จากฐานข้อมูลของขั้นตอนวิธี FP-Growth



ภาพประกอบ 3.16 การอ่านรายการข้อมูลที่ห้าจากฐานข้อมูลของขั้นตอนวิธี FP-Growth

4) สร้าง Conditional pattern base และสร้าง Conditional FP-Tree โดยเริ่มพิจารณาจากชิ้นข้อมูลสุดท้ายในตาราง Header Table นั่นคือชิ้นข้อมูล G ซึ่งจากภาพประกอบ 3.17 จะเห็นได้ว่าต้นไม้ FP-Tree มีโหนด G ปรากฏอยู่ 2 เส้นทางได้แก่ เส้นทาง D C E มีค่าความถี่เป็น 1 (เกิดร่วมกับชิ้นข้อมูล G จำนวน 1 ครั้ง) และเส้นทาง D A E มีค่าความถี่เป็น 1 ดังนั้นจะได้ Conditional pattern base ของชิ้นข้อมูล G คือ {(DCE:1), (DAE:1)} ซึ่งจากทั้ง 2 เส้นทางจะเห็นว่ามีชิ้นข้อมูล D และ E ปรากฏร่วมกันกับชิ้นข้อมูล G เหมือนกันทั้ง 2 เส้นทาง ดังนั้นจะสร้าง Conditional FP-Tree ได้เป็น {(DE:2)} | G และทำเช่นนี้กับทุกชิ้นข้อมูลในตาราง Header จะสามารถหา Conditional pattern base และสร้าง Conditional FP-Tree ของทุกชิ้นข้อมูลได้ดังแสดงในตารางที่ 3.2



ภาพประกอบ 3.17 การสร้างตาราง Header ของขั้นตอนวิธี FP-Growth

ตารางที่ 3.2 ค่า Conditional pattern base และค่า Conditional FP-Tree ของขั้นตอนวิธี FP-Growth

Items	Conditional Pattern base	Conditional FP-Tree
G	{(DCE:1), (DAE:1)}	{(DE:2)} | G
E	{(DC:1), (DCA:1), (DA:1)}	{(D:3), (DC:2), (DA:2)} | E
A	{(DC:3), (D:1)}	{(D:4), (DC:3)} | A
C	{(D:4)}	{(D:4)} | C
D	∅	∅

5) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากการสร้าง Conditional pattern base และสร้าง Conditional FP-Tree โดยเริ่มพิจารณาจากชิ้นข้อมูล G ซึ่งจากตารางที่ 3.2 จะเห็นได้ว่าชิ้นข้อมูล G มี Conditional pattern base เป็น {(DCE:1), (DAE:1)} และ Conditional FP-Tree เป็น {(DE:2)} | G ทำให้การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของชิ้นข้อมูล G แบ่งการทำงานของ {(DE:2)} | G เป็น 2 ส่วนคือ EG:2 และ DG:2 โดยส่วนแรก EG:2 จะแยกได้อีกเป็น {(D:2)} | EG ซึ่งสุดท้ายจะได้เป็น DEG:2 และส่วนที่สอง DG:2 ไม่สามารถแบ่งได้อีกดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของชิ้นข้อมูล G เป็น G:2 EG:2 DG:2 และ DEG:2 ผลลัพธ์ของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมดแสดงดังตารางที่ 3.3

ตารางที่ 3.3 ผลลัพธ์ของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Apriori

Items	Frequent Itemsets
G	G:2, EG:2, DG:2, DEG:2
E	E:3, DE:3, CE:2, AE:2, DCE:2, DAE:2
A	A:4, DA:4, CA:3, DCA:3
C	C:4, DC:4
D	D:5

3.2.3 ข้อดีของขั้นตอนวิธี FP-Growth

ขั้นตอนวิธี FP-Growth ช่วยลดจำนวนการอ่านข้อมูลจากฐานข้อมูลสำหรับการค้นหาข้อมูลที่ปรากฏร่วมกันบ่อยเหลือเพียง 2 ครั้ง และกระบวนการทำงานใช้หลักการทำงานแบบพลวัตร (Dynamic Programming) ทำให้การทำงานมีประสิทธิภาพ เหมาะสมกับฐานข้อมูลที่มีขนาดเล็กและขนาดใหญ่ มีจำนวนชิ้นข้อมูลในฐานข้อมูลน้อย และลักษณะข้อมูลที่เหมาะสมต้องมีความหนาแน่นของข้อมูลสูง คืออัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏอยู่ในรายการข้อมูลมีมาก การทำงานสามารถทำงานได้ดีหากกำหนดค่าสนับสนุนขั้นต่ำมีค่ามากๆ เพราะจะใช้เวลาในการท่องไปยังแต่ละโหนดสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้เร็ว และลดการใช้เนื้อที่ในการสร้างต้นไม้ FP-Tree สำหรับจัดเก็บข้อมูล

3.2.4 ข้อเสียของขั้นตอนวิธี FP-Growth

หากฐานข้อมูลที่ใช้ในการค้นหากลุ่มข้อมลที่ปรากฏร่วมกันบ่อยมีจำนวนชิ้นข้อมูลในฐานข้อมูลมากแล้ว การทำงานของขั้นตอนวิธี FP-Growth จะต้องใช้เนื้อที่หน่วยความจำเป็นจำนวนมาก เปลืองเนื้อที่ในการจัดเก็บข้อมูลในระหว่างการประมวลผล เนื่องจากต้นไม้ FP-Tree ที่สร้างขึ้นจะมีขนาดใหญ่ ซึ่งเกิดจากที่ต้องสร้างโหนดแทนชิ้นข้อมูลในฐานข้อมูลเป็นจำนวนมาก อีกทั้งใช้เวลาในการท่องไปยังโหนดที่ต้องการนาน

3.3 ขั้นตอนวิธี BitTableFI (BitTableFI Algorithm)

ในงานทางด้านการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นเวลาที่ใช้ในการประมวลผลและเวลาในการเข้าถึงฐานข้อมูลนั้นเป็นสิ่งสำคัญที่นักวิจัยส่วนใหญ่สนใจและศึกษา ซึ่งจากปัญหาข้างต้นได้มีนักวิจัยคิดค้นขั้นตอนวิธีใหม่ขึ้นมาที่ช่วยในการลดระยะเวลาในการประมวลผลโดยใช้หลักการทำงานของการดำเนินการระดับบิต และลดระยะเวลาในการเข้าถึงข้อมูลโดยอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียว ยกตัวอย่างเช่นงานวิจัยของขั้นตอนวิธี BitTableFI (Dong and Han, 2007)

3.3.1 หลักการทำงานของขั้นตอนวิธี BitTableFI

ขั้นตอนในการทำงานของขั้นตอนวิธี BitTableFI ใช้หลักการทำงานคล้ายคลึงกับขั้นตอนวิธี Apriori คือสร้างกลุ่มข้อมูลท้าชิงและตรวจสอบค่าความถี่เพื่อค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ซึ่งการทำงานของขั้นตอนวิธี BitTableFI เป็นไปตามวิธีการที่แสดงดังภาพประกอบ 3.18 แต่ขั้นตอนวิธี BitTableFI จะใช้เวลาน้อยกว่าขั้นตอนวิธี Apriori ในขั้นตอนของการสร้างกลุ่มข้อมูลท้าชิงและนับค่าความถี่ของกลุ่มข้อมูลท้าชิงเหล่านั้น

```
BitTableFI(Database D, int MinSup)
{
1    L1 = {frequent 1-itemsets};
2    BitTable[] db = database_bittable_init(D,L1)
3    L2 = genl2froml1(L1)
4    int k = 2;
5    While (Lk, NotEmpty())
6        k++;
7        Ck = quick_candidateitemsets_generation(Lk-1);
8        Lk = quick_candidateitemsets_support_count(Ck, db, Minsup);
9    End While
}
```

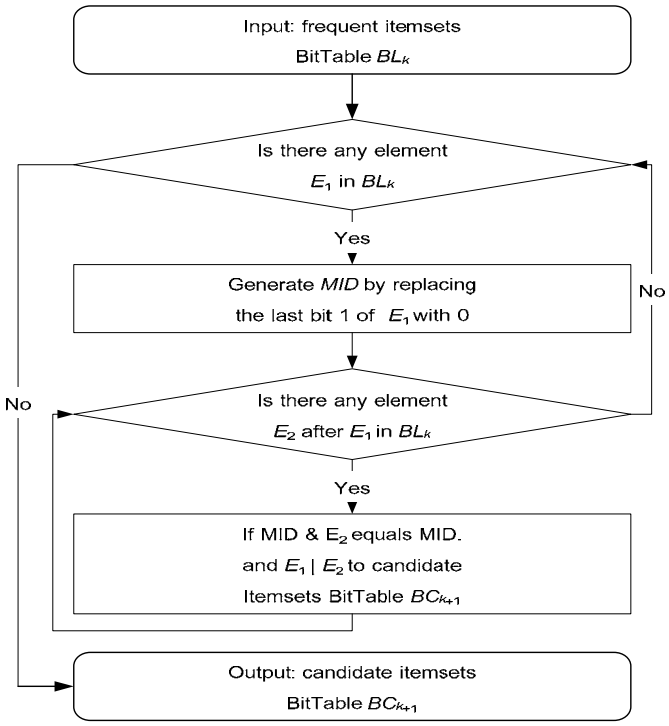
ภาพประกอบ 3.18 ขั้นตอนการทำงานของขั้นตอนวิธี BitTableFI (Dong and Han, 2007)

โดยสามารถอธิบายการทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี BitTableFI ได้ดังนี้

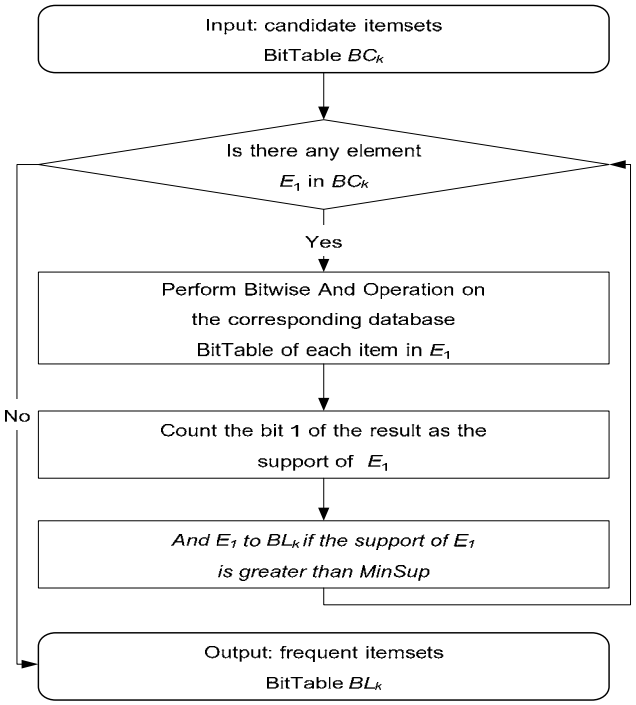
1) สร้างตาราง BitTable เพื่อใช้ในการสร้างกลุ่มข้อมูลท้าชิงและนับค่าความถี่ของกลุ่มข้อมูลท้าชิงเหล่านั้น การทำงานในขั้นตอนนี้เริ่มจากกำหนดให้แต่ละรายการข้อมูลแทนอยู่ในรูปแบบบิต โดยมีจำนวนบิตเท่ากับจำนวนของชิ้นข้อมูลที่ปรากฏในฐานข้อมูล หากในแต่ละรายการข้อมูลที่อ่านเข้ามาจากฐานข้อมูลมีชิ้นข้อมูลใดแล้วที่ตำแหน่งบิตนั้นในรายการข้อมูลนั้นจะแทนค่าบิตเป็น 1 แต่หากไม่ปรากฏชิ้นข้อมูลนั้นในรายการข้อมูลตรงตำแหน่งบิตนั้นจะแทนค่าบิตเป็น 0 แล้วจากตาราง BitTable จะสามารถหาค่า BitTable ของแต่ละชิ้นข้อมูลจากตำแหน่งบิตของชิ้นข้อมูลนั้นในแต่ละรายการข้อมูลเรียกว่า เซตของบิต (Elements) โดยที่ค่า BitTable คือค่าตัวเลขที่ได้จากเซตของบิตที่แทนค่าชิ้นข้อมูลนั้นในแต่ละรายการข้อมูล

2) สร้างกลุ่มข้อมูลท้าชิง โดยวิธีการสร้างกลุ่มข้อมูลท้าชิงของขั้นตอนวิธี BitTableFI จะมีลักษณะคล้ายกับขั้นตอนวิธี Apriori คือ สร้างกลุ่มข้อมูลท้าชิงที่ขนาดชิ้นข้อมูล k ชิ้นข้อมูล (C_k) จากกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดชิ้นข้อมูล k-1 ชิ้นข้อมูล (L_{k-1}) แต่วิธีการในการสร้างกลุ่มข้อมูลท้าชิงและการนับค่าความถี่นั้นจะมีวิธีการที่แตกต่างจากขั้นตอนวิธี Apriori การทำงานในขั้นตอนนี้เริ่มจาก ข้อมูลนำเข้าที่ใช้ในการสร้าง C_k คือเซตของค่า BitTable ซึ่งแต่ละชิ้นข้อมูลในตาราง BitTable ที่ผ่านการตรวจสอบค่าสนับสนุนขั้นต่ำแล้วจะเรียกว่า BL_k (Frequent k-Itemsets BitTable) ขั้นตอนต่อไป คือ การสร้าง MID (Middle Variable) โดยวิธีการสร้าง MID จะเริ่มจาก E_1 (เซตของบิตแรกของชิ้นข้อมูลในตาราง BitTable) นำมาแทนค่าบิตหลังสุดของ E_1 ที่มีค่าเป็น 1 ด้วย 0 หลังจากนั้นนำค่า MID มาดำเนินการระดับบิต AND กับ E_2 ในตาราง BitTabel หากผลลัพธ์ที่ได้จากการดำเนินการระดับบิต AND มีค่าเท่ากับค่า MID แสดงว่า E_1 และ E_2 มีสมาชิกที่เป็น L_{k-1} เหมือนกันดังนั้นจะสามารถสร้างกลุ่มข้อมูลท้าชิงจาก E_1 และ E_2 ได้โดยใช้การดำเนินการระดับบิต OR กับค่า E_1 และ E_2 ซึ่งผลลัพธ์ที่ได้เรียกว่า BC_k (Candidate k-Itemsets BitTable) โดยขั้นตอนการทำงานในส่วนนี้สารถแสดงดังภาพประกอบ 3.19

3) การนับค่าความถี่ของกลุ่มข้อมูลท้าชิงเพื่อใช้สำหรับตรวจสอบว่ากลุ่มข้อมูลใดเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นทำได้โดย การนับค่าบิตของเซตของบิตของชิ้นข้อมูลแต่ละตำแหน่งที่มีค่าบิตเป็น 1 โดยขั้นตอนการทำงานในการนับค่าความถี่นี้สามารถแสดงดังภาพประกอบ 3.20



ภาพประกอบ 3.19 ขั้นตอนการสร้างกลุ่มข้อมูลท้าชิงของขั้นตอนวิธี BitTableFI (Dong and Han, 2007)



ภาพประกอบ 3.20 ขั้นตอนการนับค่าความถี่ของขั้นตอนวิธี BitTableFI (Dong and Han, 2007)

3.3.2 ตัวอย่างการทำงานของขั้นตอนวิธี BitTableFI

เพื่อให้เข้าใจถึงกระบวนการทำงานของขั้นตอนวิธี BitTableFI ได้อย่างชัดเจนขึ้น จึงขอยกตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธีนี้โดยใช้ฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 ประกอบการอธิบาย พร้อมทั้งกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (มีค่าความถี่อย่างน้อยเท่ากับ 2 รายการข้อมูลปรากฏในฐานข้อมูล) ดังนี้

1) อ่านรายการข้อมูลแรกในฐานข้อมูล ซึ่งประกอบด้วยชิ้นข้อมูล C D E F G และ I สามารถแทนค่าในรูปแบบเซตของบิตได้เป็น {001111101000} เมื่อกำหนดให้บิตแรกแทนชิ้นข้อมูล A บิตที่สองแทนชิ้นข้อมูล B ตามลำดับจนถึงบิตสุดท้ายแทนชิ้นข้อมูล L และเมื่อแทนค่าทุกๆ รายการข้อมูลในฐานข้อมูลแล้ว สามารถแสดงดังภาพประกอบ 3.21 ขั้นตอนต่อไปหาค่า BitTable ของแต่ละชิ้นข้อมูล โดยหาได้จากตำแหน่งบิตของชิ้นข้อมูลนั้นในแต่ละรายการข้อมูลมารวมกันเป็นเซตของบิตของชิ้นข้อมูลนั้น ยกตัวอย่างเช่น ชิ้นข้อมูล A ที่แสดงในภาพประกอบ 3.21 มีเซตของบิตเป็น {01111} ซึ่งได้จากตำแหน่งบิตแรกของแต่ละรายการข้อมูลในตาราง BitTable ดังนั้นจะได้ค่า BitTable ของชิ้นข้อมูล A ในเลขฐานสิบเป็น 15 และเมื่อทำเช่นนี้กับทุกเซตของบิตของทุกชิ้นข้อมูลจะได้เซตค่า BitTable ของแต่ละชิ้นข้อมูลเป็น {{01111} {00100} {11011} {11111} {11100} {10000} {10100} {00010} {10000} {00001} {00100} {01000}} ซึ่งมีค่าในเลขฐานสิบเป็น {15 4 27 31 28 16 20 2 16 1 4 8} แสดงดังภาพประกอบ 3.22

TID	Items
001	C D E F G I
002	A C D E L
003	A B D E G
004	A C D H
005	A C D J

Database

TID	A	B	C	D	E	F	G	H	I	J	K	L
001	0	0	1	1	1	1	1	0	1	0	0	0
002	1	0	1	1	1	0	0	0	0	0	0	1
003	1	1	0	1	1	0	1	0	0	0	1	0
004	1	0	1	1	0	0	0	1	0	0	0	0
005	1	0	1	1	0	0	0	0	0	1	0	0

BitTable

ภาพประกอบ 3.21 การสร้างตาราง BitTableFI ของขั้นตอนวิธี BitTableFI

TID	A	B	C	D	E	F	G	H	I	J	K	L
001	0	0	1	1	1	1	1	0	1	0	0	0
002	1	0	1	1	1	0	0	0	0	0	0	1
003	1	1	0	1	1	0	1	0	0	0	1	0
004	1	0	1	1	0	0	0	1	0	0	0	0
005	1	0	1	1	0	0	0	0	0	1	0	0
BitTable	**15**	**4**	**27**	**31**	**28**	**16**	**20**	**2**	**16**	**1**	**4**	**8**

BitTable

ภาพประกอบ 3.22 การหาค่า BitTable ของขั้นตอนวิธี BitTableFI

2) นับค่าความถี่ของแต่ละชิ้นข้อมูล โดยนับตำแหน่งบิตที่มีค่าเป็น 1 ในเซตของบิตของแต่ละชิ้นข้อมูล ยกตัวอย่างเช่น ชิ้นข้อมูล A มีเซตของบิตเป็น {01111} จะสามารถหาค่าความถี่ของชิ้นข้อมูล A ได้เป็น 4 ซึ่งเกิดจากการนับตำแหน่งบิตที่มีค่าเป็น 1 ในเซตของบิตของชิ้นข้อมูล A ที่มีทั้งหมด 4 ตำแหน่ง และเมื่อทำเช่นนี้กับทุกชิ้นข้อมูลในตาราง BitTable จะสามารถหาค่าความถี่ของแต่ละชิ้นข้อมูลได้แสดงดังภาพประกอบ 3.23

TID	A	B	C	D	E	F	G	H	I	J	K	L
001	0	0	1	1	1	1	1	0	1	0	0	0
002	1	0	1	1	1	0	0	0	0	0	0	1
003	1	1	0	1	1	0	1	0	0	0	1	0
004	1	0	1	1	0	0	0	1	0	0	0	0
005	1	0	1	1	0	0	0	0	0	1	0	0
BitTable	15	4	27	31	28	16	20	2	16	1	4	8

BitTable

Item	Count
A	4
B	1
C	4
D	5
E	3
F	1
G	2
H	1
I	1
J	1
L	1

ภาพประกอบ 3.23 การนับค่าความถี่ของแต่ละชิ้นข้อมูลของขั้นตอนวิธี BitTableFI

3) เมื่อได้ค่าความถี่ของทุกชิ้นข้อมูลแล้วก็นำมาคำนวณค่าสนับสนุนเพื่อใช้ในการพิจารณาว่าชิ้นข้อมูลใดเป็น L_1 ผลลัพธ์ที่ได้หลังจากตัดชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำ จะได้ L_1 เป็น {A C D E G} แสดงดังภาพประกอบ 3.24

Item	Count	Support
A	4	0.8
B	1	0.2
C	4	0.8
D	5	1.0
E	3	0.6
F	1	0.2
G	2	0.4
H	1	0.2
I	1	0.2
J	1	0.2
L	1	0.2

Item	Count
A	4
C	4
D	5
E	3
G	2

L_1

ภาพประกอบ 3.24 ผลลัพธ์ของการค้นหากลุ่มข้อมูล L_1 ของขั้นตอนวิธี BitTableFI

4) แก้ไขตาราง BitTable ให้มีขนาดเล็กลงด้วยการแก้ไขตำแหน่งบิตใหม่ให้กับ L_1 ในตาราง BitTable เพื่อลดขนาดของตาราง BitTable โดยการตัดชิ้น

ข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำออกไปและแทนที่บิตนั้นด้วยชิ้นข้อมูลถัดไปที่เป็น L_1 ผลลัพธ์ที่ได้จากการทำงานในขั้นตอนนี้ แสดงดังภาพประกอบ 3.25 ซึ่งจะว่าเรียกว่า BL_1

TID	A	C	D	E	G
001	0	1	1	1	1
002	1	1	1	1	0
003	1	0	1	1	1
004	1	1	1	0	0
005	1	1	1	0	0
BitTable	**15**	**27**	**31**	**28**	**20**

BitTable

ภาพประกอบ 3.25 ตาราง BitTable หลังแก้ไขตำแหน่งบิตของขั้นตอนวิธี BitTableFI

5) เนื่องจาก L_1 ประกอบไปด้วยชิ้นข้อมูลทั้งหมด 5 ชิ้นข้อมูล คือ {A C D E G} ทำให้ต้องใช้จำนวนของงบิตในการแทนค่าในรูปแบบบิตสำหรับแต่ละเซตของบิตทั้งหมด 5 ตำแหน่ง ดังนั้นจะสามารถแทนแต่ละชิ้นข้อมูลในแต่ละตำแหน่งได้เป็นดังนี้ ชิ้นข้อมูล A แทนที่ตำแหน่งบิตแรก ชิ้นข้อมูล C แทนที่ตำแหน่งบิตสอง ชิ้นข้อมูล D แทนที่ตำแหน่งบิตที่สาม ชิ้นข้อมูล E แทนที่ตำแหน่งบิตที่สี่ และชิ้นข้อมูล G แทนที่ตำแหน่งบิตที่ห้า และจาก L_1 สามารถสร้าง C_2 ได้เป็น {{A C} {A D} {A E} {C D} {C E} {D E} {D G} {E G}} ทำให้สามารถแทนค่า BC_2 ของแต่ละ C_2 ให้อยู่ในรูปแบบบิตได้เป็น {{11000} {10100} {10010} {01100} {01010} {00110} {00101} {00011}} แสดงดังภาพประกอบ 3.26

Item	Count	Bitsets
A	4	10000
C	4	01000
D	5	00100
E	3	00010
G	2	00001

L_1

➡

Itemsets	BitSets
{A C}	11000
{A D}	10100
{A E}	10010
{A G}	10001
{C D}	01100
{C E}	01010
{C G}	01001
{D E}	00110
{D G}	00101
{E G}	00011

BC_2

ภาพประกอบ 3.26 การแทนค่า BC_2 ของขั้นตอนวิธี BitTableFI

6) นับค่าความถี่ของ BC_2 โดยใช้การดำเนินการระดับบิต AND กับค่า BitTable ยกตัวอย่างเช่น เซตของบิตแรก (E_1) ในตาราง BC_2 คือกลุ่มข้อมูลท้าชิง {A C} มีเซตของบิตเป็น {11000} การหาค่าความถี่ของ {A C} จะเริ่มจากนำค่า BitTable ของชิ้นข้อมูล A คือ {01111} มาดำเนินการระดับบิต AND กับค่า BitTable ของชิ้นข้อมูล C คือ

{11011} จะได้ผลลัพธ์เป็น {01011} ดังนั้นค่าความถี่ของกลุ่มข้อมูลท้าชิง {A C} จะเท่ากับ 3 ซึ่งเกิดจากผลลัพธ์ที่ได้มีตำแหน่งบิตที่มีค่าเป็น 1 ทั้งหมด 3 ตำแหน่ง ดังนั้นเมื่อหาค่าความถี่ให้กับทุกๆ กลุ่มข้อมูลท้าชิง และหลังตัดกลุ่มข้อมูลท้าชิงที่ไม่ผ่านค่าสนับสนุนขั้นต่ำแล้วจะได้ BL_2 แสดงดังภาพประกอบ 3.27

Itemsets	BitSets	Count
{A C}	11000	3
{A D}	10100	4
{A E}	10010	2
{A G}	10001	1
{C D}	01100	4
{C E}	01010	2
{C G}	01001	1
{D E}	00110	3
{D G}	00101	2
{E G}	00011	2

BC_2

Itemsets	BitSets	Count
{A C}	11000	3
{A D}	10100	4
{A E}	10010	2
{C D}	01100	4
{C E}	01010	2
{D E}	00110	3
{D G}	00101	2
{E G}	00011	2

BL_2

ภาพประกอบ 3.27 การนับค่าความถี่ของ BC_2 และสร้าง BL_2 ของขั้นตอนวิธี BitTableFI

7) นำข้อมูลจากเซต BL_2 มาสร้าง BC_3 โดยขั้นตอนการสร้าง BC_3 ขอยกตัวอย่าง E_1 ของ BL_2 คือ {A C} มีเซตของบิตเป็น {11000} นำมาสร้าง MID ได้เป็น {10000} ซึ่งค่า MID ของ E_1 ที่ได้เกิดจากการแทนตำแหน่งบิตสุดท้ายที่เป็น 1 ด้วย 0 จากนั้นนำค่า MID มาดำเนินการระดับบิต AND กับเซตของบิตที่อยู่ถัดจากเซตของบิตที่กำลังพิจารณาอยู่ในเซต BL_2 ทั้งหมด ซึ่งในที่นี้เริ่มจาก E_2 จะได้เป็น {10000} & {10100} ผลลัพธ์ที่ได้คือ {10000} ซึ่งผลลัพธ์ที่ได้มีค่าเท่ากับค่า MID ของ E_1 ดังนั้นแสดงว่า E_1 และ E_2 มีชิ้นข้อมูลตัวแรกเหมือนกัน และสามารถสร้างกลุ่มข้อมูลท้าชิงได้จากการนำ E_1 มาดำเนินการระดับบิต OR กับ E_2 จะได้เป็น {11000} | {10100} ผลลัพธ์ที่ได้คือ {11100} ดังนั้นจะได้กลุ่มข้อมูลท้าชิงเป็น {A C D} และเมื่อทำเช่นนี้กับทุกเหตุการณ์ใน BL_2 จะได้ผลลัพธ์เป็นกลุ่มข้อมูลท้าชิงทั้งหมดเป็น {A C D} {A C E} {A D E} {C D E} และ {D E G} แสดงดังภาพประกอบ 3.28

i	Itemsets	BitSets
1	{A C}	11000
2	{A D}	10100
3	{A E}	10010
4	{C D}	01100
5	{C E}	01010
6	{D E}	00110
7	{D G}	00101
8	{E G}	00011

BL_2

i	$BL_2[i]$	MID	MID&$BL_2[i+1]$	$BL_2[i] \mid BL_2[i+1]$	Itemsets
1	11000	10000	10000	11100	{A C D}
2	10100	10000	10000	11010	{A C E}
3	10010	10000	10000	10110	{A D E}
4	01100	01000	01000	01110	{C D E}
5	01010	01000	00000	-	-
6	00110	00100	00100	00111	{D E G}
7	00101	00100	00000	-	-
8	00011	00010	-	-	-

ภาพประกอบ 3.28 การสร้าง BC_3 ของขั้นตอนวิธี BitTableFI

8) นับค่าความถี่ของ BC_3 โดยใช้การดำเนินการระดับบิต AND กับค่า BitTable ยกตัวอย่างเช่น E_1 ในตาราง BC_3 คือกลุ่มข้อมูลท้าชิง {A C D} มีเซตของบิตเป็น {11100} การทำงานในการค้นหาค่าความถี่ของ {A C D} จะเริ่มจากนำค่า BitTable ของชิ้นข้อมูล A คือ {01111} มาดำเนินการระดับบิต AND กับค่า BitTable ของชิ้นข้อมูล C คือ {11011} และชิ้นข้อมูล D คือ {11111} จะได้ผลลัพธ์เป็น {01011} ดังนั้นค่าความถี่ของกลุ่มข้อมูลท้าชิง {A C D} จะเท่ากับ 3 ซึ่งเกิดจากผลลัพธ์ที่ได้มีตำแหน่งที่บิตมีค่าเป็น 1 ทั้งหมด 3 ตำแหน่ง ผลลัพธ์ที่ได้แสดงดังภาพประกอบที่ 3.29 ดังนั้นเมื่อหาค่าความถี่ให้กับทุกกลุ่มข้อมูลท้าชิงแล้วและเมื่อตัดกลุ่มข้อมูลท้าชิงที่ไม่ผ่านค่าสนับสนุนขั้นต่ำแล้วจะได้ BL_3 แสดงดังภาพประกอบ 3.30

A		C		D		Count
0		1		1		0
1		1		1		1
1	&	0	&	1	=	0
1		1		1		1
1		1		1		1
15		27		31		11

ภาพประกอบ 3.29 การนับค่าความถี่ของ BC_3 ของขั้นตอนวิธี BitTableFI

Itemsets	BItsets	Count
{A C D}	11100	3
{A C E}	11010	1
{A D E}	10110	2
{C D E}	01110	2
{D E G}	00111	2

BC_3

Itemsets	BItsets	Count
{A C D}	11100	3
{A D E}	10110	2
{C D E}	01110	2
{D E G}	00111	2

BL_3

ภาพประกอบ 3.30 การสร้าง BL_3 ของขั้นตอนวิธี BitTableFI

9) นำ BL_3 ที่ได้ไปสร้าง BC_4 โดยวิธีการสร้าง BC_4 นั้นจะทำเช่นเดียวกันกับในหัวข้อที่ 7) แต่เนื่องจากเมื่อตรวจสอบแล้วไม่สามารถสร้าง BC_4 จาก BL_3 ได้ จึงหยุดการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับขั้นตอนวิธี BitTableFI ผลลัพธ์ที่ได้ทั้งหมดแสดงดังตาราง 3.3

ตารางที่ 3.4 ผลลัพธ์ของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี BitTableFI

Level	Frequent Itemsets
L_1	A:4, C:4, D:5, E:3, G:2
L_2	AC:3, AD:4, AE:2, CD:4, CE:2, DE:3, DG:2, EG:2
L_3	ACD:3, ADE:2, CDE:2, DEG:2

3.3.3 ข้อดีของขั้นตอนวิธี BitTableFI

ขั้นตอนวิธี BitTableFI มีลักษณะเด่นที่นำเสนอการจัดเก็บข้อมูลในรูปแบบใหม่ โดยเก็บอยู่ในรูปแบบบิต ซึ่งช่วยลดเนื้อที่หน่วยความจำที่ใช้ในระหว่างการประมวลผล ทำให้เหมาะสำหรับการทำงานกับฐานข้อมูลที่มีขนาดใหญ่ อีกทั้งยังช่วยลดระยะเวลาการทำงานให้เร็วขึ้นโดยใช้การดำเนินการระดับบิตในการสร้างกลุ่มข้อมูลท้าชิง และการนับค่าความถี่ของกลุ่มข้อมูลท้าชิงเหล่านั้น

3.3.4 ข้อเสียของขั้นตอนวิธี BitTableFI

ขั้นตอนวิธี BitTableFI มีลักษณะการทำงานคล้ายกับการทำงานของขั้นตอนวิธี Apriori คือต้องมีการสร้างกลุ่มข้อมูลท้าชิงและนับค่าความถี่ของกลุ่มข้อมูลท้าชิงเหล่านั้น ทำให้หากต้องสร้างกลุ่มข้อมูลเหล่านั้นเป็นจำนวนมากแล้วย่อมทำให้เสียเวลาในการทำงานมากขึ้นเช่นกัน จึงทำให้วิธีการนี้เหมาะสำหรับฐานข้อมูลขนาดใหญ่ที่มีรายการข้อมูลซึ่งประกอบไปด้วยชิ้นข้อมูลน้อยๆ เพราะจะช่วยลดระยะเวลาในการสร้างกลุ่มข้อมูลท้าชิงและนับค่าความถี่ของกลุ่มข้อมูลท้าชิงน้อยลง

3.4 ขั้นตอนวิธี Index-BitTableFI (Index-BitTableFI Algorithm)

ขั้นตอนวิธี Index-BitTableFI (Song *et al*., 2008) พัฒนาขึ้นมาจากขั้นตอนวิธี BitTableFI ที่ใช้ตาราง BitTable ในการทำงานเช่นเดียวกัน ซึ่งเป็นโครงสร้างข้อมูลที่มีประสิทธิภาพในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ช่วยลดเนื้อที่ในการสร้างการเปรียบเทียบกลุ่มข้อมูลและการนับความถี่ของกลุ่มข้อมูล แต่การทำงานของขั้นตอนวิธี BitTableFI หากกลุ่มข้อมูลที่มีสมาชิกในกลุ่มมากหรือมีการกำหนดค่าสนับสนุนขั้นต่ำมีค่าน้อยมาก ๆ จะทำให้เกิดปัญหาได้ คือเสียเวลาในการเปรียบเทียบกลุ่มข้อมูลและทดสอบ เพราะต้องสร้างกลุ่มข้อมูลจำนวนมากในการเปรียบเทียบ ทำให้เกิดความซับซ้อนในการนับจำนวนความถี่ ด้วยปัญหาดังกล่าวทำให้งานวิจัยนี้ได้นำเสนอขั้นตอนวิธีนี้ขึ้นมา

3.4.1 หลักการทำงานของขั้นตอนวิธี Index-BitTableFI

หลักการทำงานของขั้นตอนวิธี Index-BitTableFI ใช้ตาราง BitTable ในรูปแบบการทำงานทั้งในแนวนอน (Horizontally) และแนวตั้ง (Vertically) ใช้การค้นหาแบบแนวกว้างก่อน (Breadth-first Search) เพื่อระบุชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย และใช้การค้นหาแบบแนวลึกก่อน (Depth-first search) เพื่อสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยอื่น ๆ ทั้งหมด และเพื่อให้เข้าใจถึงหลักการทำงานของขั้นตอนวิธี Index-BitTableFI จึงขอกล่าวถึงบทนิยามและทฤษฎีบทที่ได้เขียนไว้ในงานวิจัยของขั้นตอนวิธี Index-BitTableFI ดังนี้

<u>บทนิยาม 3.4.1</u> Index Array คือ อาร์เรย์ที่มีขนาด m_1 เมื่อ m_1 คือจำนวนของชิ้นข้อมูล L_1 ทั้งหมดในตาราง BitTable ซึ่งสมาชิกแต่ละตัวของอาเรย์ถูกจัดเก็บอยู่ในรูปของคู่อันดับ (Item, Subsume) เมื่อ Item คือชิ้นข้อมูลที่เป็น L_1 และ Subsume คือกลุ่มข้อมูลที่ปรากฏทุกรายการข้อมูลในฐานข้อมูลที่มี Item นั้นปรากฏอยู่ เราจะเรียก Item นี้ว่า Index และ Subsume ของ Item นั้นว่า Subsume Index

<u>ทฤษฎีบท 3.4.1</u> ให้ Index[*j*].item เป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีค่าสนับสนุนเท่ากับค่าสนับสนุนขั้นต่ำ โดยที่ตำแหน่งของ Index[*j*].item คือชิ้นข้อมูลที่อยู่ในตำแหน่ง *j* ซึ่งเกิดขึ้นหลังจาก *i* เมื่อ *i* คือชิ้นข้อมูลที่อยู่ในตำแหน่งก่อนหน้า Index[*j*].item ในตาราง BitTable ดังนั้นเมื่อ Index[*j*].item ยูเนียนกับ *i* ก็จะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเช่นกัน

<u>ทฤษฎีบท 3.4.2</u> ให้ Item คือชิ้นข้อมูลและ Subsume(Item) = $a_1, a_2, \ldots, a_m$ เมื่อ *m* คือจำนวนชิ้นข้อมูลใน Subsume และ *a* คือชิ้นข้อมูลที่เป็นสมาชิกของ Subsume หากสร้างกลุ่มข้อมูลจากชิ้นข้อมูลใน Subsume โดยที่กลุ่มข้อมูลที่สร้างขึ้นเป็นเพาเวอร์เซตของ Subsume ยกเว้นเซตว่าง แล้วค่าความถี่ของกลุ่มข้อมูลที่สร้างขึ้นทั้งหมดนั้นจะเท่ากับ

ค่าความถี่ของ Item โดยการทำงานของขั้นตอนวิธี Index-BitTableFI มีการทำงานหลักๆ สามารถอธิบายได้ดังนี้

1) สร้างตาราง BitTable โดยขั้นตอนการสร้างจะอ่านข้อมูลจากฐานข้อมูลเพื่อตรวจสอบชิ้นข้อมูลใดเป็น L_1 แล้วตัดชิ้นข้อมูลที่ไม่เป็น L_1 ออกไป หลังจากนั้นเรียงข้อมูลที่อ่านเข้าจากฐานข้อมูลนั้นใหม่ในแต่ละรายการข้อมูล โดยเรียงชิ้นข้อมูลจากชิ้นข้อมูลที่มีค่าความถี่น้อยไปยังชิ้นข้อมูลที่มีค่าความถี่มาก

2) สร้าง Index Array เพื่อนำไปใช้ในขั้นตอนของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดย Index Array คือคู่อันดับของชิ้นข้อมูลและ Subsume เก็บอยู่ในรูปของ (Item, Subsume) ดังบทนิยามที่ 3.4.1 โดยขั้นตอนของการสร้าง Index Array สามารถอธิบายได้ดังภาพประกอบที่ 3.31

Input: database *D*, min_sup

Output: index array

1 Scan database ***D*** once. Delete infrequent items;
2 Sort frequent single items in support ascending order as $a_1, a_2, ..., a_m$;
3 **For** each element index[*j*] of index array **do**
4 index[*j*].item = a_j;
5 Represent the database ***D*** with BitTable;
6 **For** each element index[*j*] in index array **do**
7 index[*j*].subsume = ∅;
8 candidate = $\bigcap_{t \in g(\text{index}[j].\text{item})} t$;
9 **For** each $i > j$ **do**
10 **If** the value of the i^{th} bit in candidate is set **then**
11 index[*j*].subsume ⟵ index[*i*].item;
12 **End If**
13 **End For**
14 **End For**
15 **Write Out** the index array;

ภาพประกอบ 3.31 ขั้นตอนการสร้าง Index Array ของขั้นตอนวิธี BitTableFI (Song *et al*., 2008)

3) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย วิธีการค้นหาจะใช้ข้อมูลที่ได้จากการสร้าง Index Array และใช้การยูเนียนสำหรับค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย การทำงานเริ่มจากการสร้างต้นไม้เพื่อใช้ในจัดเก็บกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยเริ่มจากบันทึก L_1 ที่ได้จากการสร้าง Index Array ซึ่งการเข้าถึงจะใช้การค้นหาแบบแนวกว้างก่อน จากนั้นจะสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นร่วมกันกับสมาชิกใน Index Array โดยเริ่มจากชิ้นข้อมูลแรกในตาราง BitTable ซึ่งก็คือ Index แรกนั่นเอง กลุ่มข้อมูลที่สร้างจะใช้การยูเนียนกันระหว่าง Index กับเพาเวอร์เซตของ Subsume Index นั้น และหาก Index ดังกล่าวมีค่าความถี่มากกว่าค่าสนับสนุนขั้นต่ำ ก็จะทำงานในส่วนของการสร้างการเปรียบเทียบกลุ่มข้อมูลกับชิ้นข้อมูลอื่นที่ปรากฏร่วมกันบ่อยที่อยู่ในลำดับหลัง Index ที่กำลังพิจารณาในตาราง BitTable โดยการทำงานในการจัดเก็บกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจะใช้การค้นหาแบบแนวลึกก่อน โดยการทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Index-BitTableFI แสดงดังภาพประกอบ 3.32 ถึง 3.33

```
Input: index array, min_sup
Output: frequent itemsets
1   For each element index[j] of index array do
2     Write Out index[j].item and its support;
3     If index[j].subsume == ∅ then
4       If (sup(index[j].item) > min_sup) then
5         Depth_First(index[j].item, t(index[j].item));        // t(index[j].item) is the
set of frequent single items that after index[j].item, according to support ascending
order
6       End If
7     Else
8       For each element s-item ⊆ index[j].subsume do
9         Write Out index[j].item U s-item and its support;  // the support of any
index[j].item U s-item equals to support of index[j].item
10      End For
```

ภาพประกอบ 3.32 ขั้นตอนการทำงานของขั้นตอนวิธี BitTableFI (Song *et al*., 2008)

```
11        If (sup(index[j].item) > min_sup) then
12          tail ⟵ t(index[j].item) \ items in index[j].subsume;  // delete items
included by index[j].subsume from t(index[j].item)
13          Depth_First(index[j].item, tail);
14          For each element s-item ⊆ index[j].subsume do
15              Depth_First(index[j] U s-item.item, tail);
16          End For
17        End If
18    End Else
19    End If
20  End For
Procedure Depth_First (itemset, tail)
21  If tail == ∅ then retun;
22  For each i ∈ tail do
23    f-itemset ⟵ itemset U i;
24    If (sup(f-itemset) ≥ min_sup) then
25      Write Out f-itemset and its support;
26      tail ⟵ tail \ l;
27      Depth_First (f-itemset, tail);
28    End If
29  End For
```

ภาพประกอบ 3.33 ขั้นตอนการทำงานของขั้นตอนวิธี BitTableFI (ต่อ)

3.4.2 ตัวอย่างการทำงานของขั้นตอนวิธี Index-BitTableFI

เพื่อให้เข้าใจถึงกระบวนการทำงานของขั้นตอนวิธี Index-BitTableFI ได้อย่างชัดเจนขึ้น จึงขอยกตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธีนี้โดยใช้ฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 ประกอบการอธิบาย พร้อมทั้งกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (มีค่าความถี่อย่างน้อยเท่ากับ 2 รายการข้อมูลปรากฏในฐานข้อมูล) ดังนี้

1) อ่านรายการข้อมูลแรกในฐานข้อมูล และนับค่าความถี่ของแต่ละชิ้นข้อมูล ตัดชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำ ผลลัพธ์ที่ได้ L_1 คือ A C D E และ G หลังจากนั้นนำชิ้นข้อมูลที่ได้มาเรียงค่าความถี่จากมากไปหาน้อย จะสามารถเรียงชิ้นข้อมูลได้

เป็น G E A C และ D จากนั้นเรียงลำดับชิ้นข้อมูลในฐานข้อมูลใหม่ตามลำดับของชิ้นข้อมูล แสดงดังภาพประกอบ 3.34

TID	Items
001	C D E F G I
002	A C D E L
003	A B D E G
004	A C D H
005	A C D J

Database

TID	Items
001	G E C D
002	E A C D
003	G A E D
004	A C D
005	A C D

Database

ภาพประกอบ 3.34 การเรียงลำดับชิ้นข้อมูลในฐานข้อมูลของขั้นตอนวิธี Index-BitTableFI

2) นำชิ้นข้อมูลที่ผ่านการจัดเรียงใหม่แล้วนั้นมาสร้างตาราง BitTable ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 3.35

TID	Items
001	G E C D
002	E A C D
003	G E A D
004	A C D
005	A C D

Database

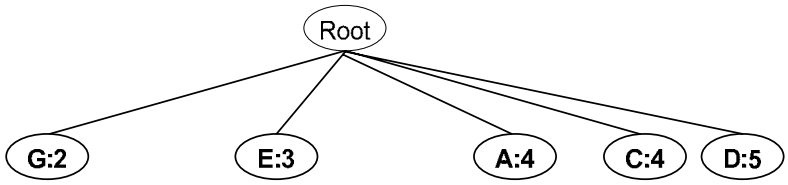
TID	G	E	A	C	D
001	1	1	0	1	1
002	0	1	1	1	1
003	1	1	1	0	1
004	0	0	1	1	1
005	0	0	1	1	1

BitTable

ภาพประกอบ 3.35 การสร้างตาราง BitTable ของขั้นตอนวิธี Index-BitTableFI

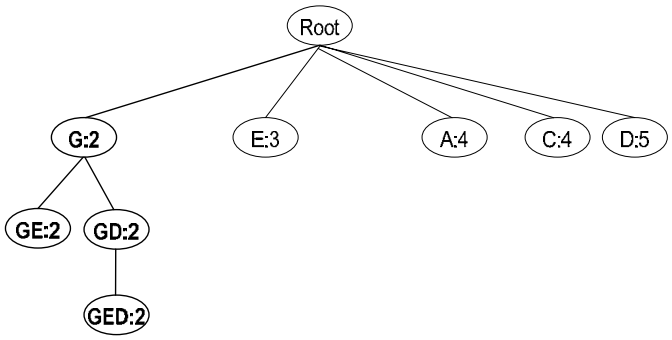
3) สร้าง Index Array โดยเริ่มจากชิ้นข้อมูลแรกของ L_1 คือ ชิ้นข้อมูล G ซึ่งเมื่อตรวจสอบตำแหน่งบิตของชิ้นข้อมูล G ในตาราง BitTable ที่มีค่าบิตเป็น 1 มีทั้งหมด 2 รายการข้อมูล คือรายการข้อมูลแรกและรายการข้อมูลที่สามมีค่าเป็น 11011 และ 11101 ตามลำดับ หลังจากนั้นนำทั้งสองรายการข้อมูลมาอินเตอร์เซ็คชั่นกัน ผลลัพธ์ที่คือ 11001 ดังนั้นจะได้ Subsume ของชิ้นข้อมูล G ประกอบด้วยชิ้นข้อมูล E และชิ้นข้อมูล D ซึ่งได้จากตำแหน่งบิตที่มีค่าเป็น 1 ของผลการอินเตอร์เซ็คชั่น ทำให้สามารถเขียน Index Array ของ G ให้อยู่ในรูปของคู่อันดับได้เป็น (G,ED) และเมื่อตรวจสอบชิ้นข้อมูล L_1 ที่เหลือทุกตัวจะสามารถสร้าง Index Array ทั้งหมดได้เป็น (G,ED) (E,D) (A,D) (C,D) และ (D, ∅)

4) สร้างต้นไม้เพื่อใช้ในการจัดเก็บกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยเริ่มบันทึกชิ้นข้อมูล L_1 ทั้งหมดแสดงดังภาพประกอบ 3.36



ภาพประกอบ 3.36 การบันทึกชิ้นข้อมูล L_1 ในต้นไม้ของขั้นตอนวิธี Index-BitTableFI

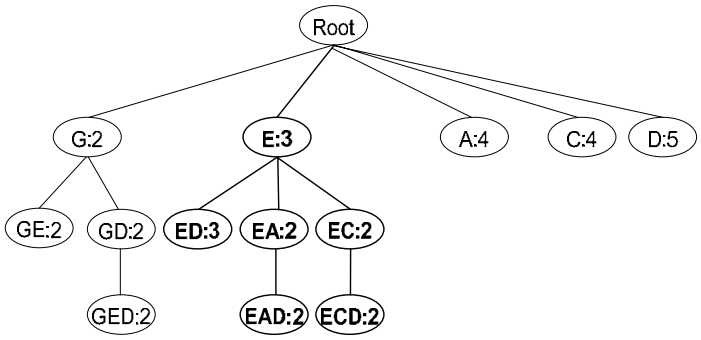
5) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นทั้งหมด โดยเริ่มจาก Index แรกของ Index Array คือชิ้นข้อมูล G การทำงานเริ่มจากตรวจสอบ Subsume(G) ว่าเป็นเซตว่างหรือไม่ ซึ่งปรากฏว่าชิ้นข้อมูล G ไม่เป็นเซตว่าง โดย Subsume(G) มีสมาชิกประกอบด้วยชิ้นข้อมูล E และ D ดังนั้นสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้จากชิ้นข้อมูล G โดยนำ G ยูเนียนกับเพาเวอร์เซตของ Subsume(G) ทุกกรณีที่เป็นไปได้ยกเว้นเซตว่าง ตามทฤษฎีบทที่ 3.4.1 จะได้เป็น GE GD และ GED โดยมีค่าความถี่เท่ากับ 2 เนื่องจากเป็นไปตามทฤษฎีบทที่ 3.4.2 ดังนั้นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่สามารถสร้างได้จาก Index Array ของชิ้นข้อมูล G มีทั้งหมดดังนี้ G:2 GE:2 GD:2 และ GED:2 หลังจากนั้นบันทึกกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ได้นั้นในต้นไม้แสดงดังภาพประกอบ 3.37



ภาพประกอบ 3.37 การบันทึกกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นร่วมกับชิ้นข้อมูล G ในต้นไม้ของขั้นตอนวิธี Index-BitTableFI

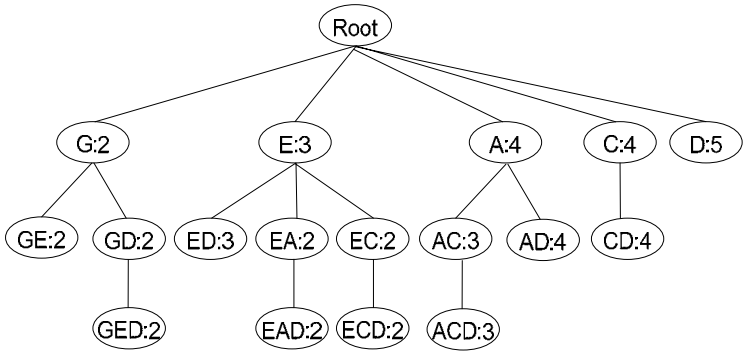
6) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นจาก Index ถัดไปของ Index Array คือชิ้นข้อมูล E โดย Subsume(E) มีสมาชิกประกอบด้วยชิ้นข้อมูลเพียงตัวเดียวคือ D ดังนั้นสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้จากชิ้นข้อมูล E โดยนำ E ยูเนียนกับ D จะได้เป็น ED โดยมีค่าความถี่เท่ากับ 3 แต่เนื่องจากค่าความถี่ของชิ้นข้อมูล E มีค่ามากกว่าค่าความถี่ขั้นต่ำที่กำหนดไว้ จึงต้องมีการตรวจสอบกับชิ้นข้อมูลอ่านที่อยู่ถัดจากชิ้นข้อมูล E ในตาราง BitTable ที่ไม่ใช่ชิ้นข้อมูลที่เป็น Subsume(E) ในที่นี้จะมีชิ้นข้อมูล A และชิ้นข้อมูล C โดยใช้หลักการเดียวกันคือใช้การยูเนียน E กับชิ้นข้อมูลเหล่านั้นทำให้สามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้เป็น EA EC และ EAC โดยที่ EA และ EC มีค่าความถี่

เท่ากับ 2 แต่เนื่องจาก EAC มีค่าความถี่เท่ากับ 1 จึงไม่เป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย หลังจากนั้นนำ ED ที่ได้จากการนำ E ยูเนียนกับ D ข้างต้นมายูเนียนกับชิ้นข้อมูล A และชิ้นข้อมูล C เช่นกันซึ่งผลลัพธ์ที่ได้คือ EAD และ ECD มีค่าความถี่เท่ากับ 2 ดังนั้นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่สามารถสร้างได้จาก Index Array ของชิ้นข้อมูล E มีทั้งหมดดังนี้ E:3 ED:3 EA:2 EC:2 EAD:2 ECD:2 หลังจากนั้นบันทึกกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ได้นั้นในต้นไม้แสดงดังภาพประกอบ 3.38



ภาพประกอบ 3.38 การบันทึกกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นร่วมกับชิ้นข้อมูล E ในต้นไม้ของขั้นตอนวิธี Index-BitTableFI

7) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นจาก Index ที่อยู่ถัดจาก Index ของชิ้นข้อมูล E ของ Index Array คือ Index ของชิ้นข้อมูล A ผลลัพธ์ที่ได้ประกอบไปด้วย A:4 AD:4 AC:3 และ ACD:3 ต่อมา Index ของชิ้นข้อมูล C ผลลัพธ์ที่ได้ประกอบไปด้วย C:4 และ CD:4 และสุดท้าย Index ของชิ้นข้อมูล D ผลลัพธ์ที่ได้คือ D:5 โดยกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่สร้างขึ้นได้เหล่านี้บันทึกเข้าไปในต้นไม้แสดงดังภาพประกอบ 3.39



ภาพประกอบ 3.39 การบันทึกกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมดในต้นไม้ของขั้นตอนวิธี Index-BitTableFI

3.4.3 ข้อดีของขั้นตอนวิธี Index-BitTableFI

การทำงานของขั้นตอนวิธี Index-BitTableFI เป็นขั้นตอนวิธีที่ช่วยลดจำนวนของการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยการใช้ Index array ทำให้ช่วยลดค่าใช้จ่ายในเรื่องของหน่วยความจำและเวลาที่ใช้ในการทำงาน อีกทั้งยังสามารถทำงานได้อย่างมีประสิทธิภาพกับฐานข้อมูลที่มีการกำหนดค่าสนับสนุนขั้นต่ำที่มีค่าน้อย ๆ และอัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏในรายการข้อมูลมีสูง

3.4.4 ข้อเสียของขั้นตอนวิธี Index-BitTableFI

ถึงแม้ว่าขั้นตอนวิธี Index-BitTableFI จะช่วยลดระยะเวลาในขั้นตอนของการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย แต่ในขั้นตอนของการจัดเก็บข้อมูลและตรวจสอบกลุ่มข้อมูลที่เป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในต้นไม้นั้นจะใช้เวลานาน เนื่องจากต้องใช้การค้นหาแบบแนวลึกก่อนเพื่อบันทึกการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยอื่น ๆ ที่เกิดขึ้นร่วมกับ Index ของแต่ละชิ้นข้อมูลนั้นๆ ซึ่งหากมีจำนวนของกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เกิดขึ้นร่วมกับ Index นั้นมาก จะทำให้ต้องใช้เวลานานสำหรับการค้นหาเพื่อบันทึกกลุ่มข้อมูลเหล่านั้นเข้าไปในต้นไม้

บทที่ 4

ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน

การเลือกขั้นตอนวิธีที่มีประสิทธิภาพเพื่อใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นเป็นเรื่องที่สำคัญอย่างยิ่งในการทำเหมืองข้อมูลสำหรับการค้นหากฎความสัมพันธ์ของข้อมูลในฐานข้อมูล ซึ่งโดยทั่วไปแล้วขั้นตอนวิธีที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นได้พัฒนาขึ้นเพื่อตอบสนองวัตถุประสงค์ที่แตกต่างกันออกไป เพื่อให้การทำงานมีประสิทธิภาพมากที่สุด ดังแสดงในตารางที่ 4.1 ยกตัวอย่างเช่น ขั้นตอนวิธี FP-Growth ลดระยะเวลาในการสร้างกลุ่มข้อมูลท้าชิงโดยการหลีกเหลี่ยงการสร้างกลุ่มข้อมูลท้าชิง ขั้นตอนวิธี ITL-Mine ลดจำนวนของอ่านข้อมูลจากฐานข้อมูลเพื่อให้การทำงานเร็วขึ้น ขั้นตอนวิธี BitTableFI ลดเนื้อที่ในหน่วยความจำในระหว่างการประมวลผลโดยเก็บข้อมูลอยู่ในรูปแบบบิต เป็นต้น แต่อีกวิธีการหนึ่งที่สามารถทำให้ขั้นตอนวิธีการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นเป็นขั้นตอนวิธีที่มีประสิทธิภาพได้ นั่นคือ การรวมรายการข้อมูลที่ซ้ำกันก่อนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย เพื่อลดระยะเวลาการทำงานและลดเนื้อที่ในหน่วยความจำ

วิทยานิพนธ์นี้จึงได้นำเสนอขั้นตอนวิธีที่มีประสิทธิภาพสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยใช้ชื่อว่า “AMFIST: ขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน” (AMFIST: An Algorithm for Mining Frequent Itemsets based on the Similarities of Transactions) โดยหลีกเหลี่ยงการสร้างกลุ่มข้อมูลท้าชิง การอ่านข้อมูลจากฐานข้อมูลเพื่อใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพียงครั้งเดียว ใช้การจัดเก็บข้อมูลในรูปแบบบิต และใช้การดำเนินการระดับบิตในการค้นหาค่าความถี่และสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย สุดท้ายใช้วิธีการรวมรายการข้อมูลที่ซ้ำกันหลังจากตัดกลุ่มข้อมูลที่ไม่ปรากฏร่วมกันบ่อย เพื่อให้สามารถทำงานได้อย่างรวดเร็วและลดเนื้อที่ในการจัดเก็บข้อมูลในหน่วยความจำระหว่างการประมวลผล

ตารางที่ 4.1 เปรียบเทียบคุณสมบัติของแต่ละขั้นตอนวิธี

ขั้นตอนวิธี / คุณสมบัติ	Apriori (1993)	Close (1999)	FP-Growth (2000)	H-Mine (2001)	ITL-Mine (2002)	CT-ITL (2003)	CT-PRO (2004)	BitTableFI (2007)	Inex-BitTableFI (2008)
การสร้างกลุ่มข้อมูลท้าชิง (Generate Candidate itemsets)	✓	✓	×	×	×	×	×	✓	✓
ขนาดของฐานข้อมูลที่เหมาะสม	เล็ก	ใหญ่	เล็ก	ใหญ่	เล็ก	ใหญ่	ใหญ่	ใหญ่	ใหญ่
จำนวนครั้งของการอ่านฐานข้อมูล	>2	>2	2	1	1	1	1	1	1
ค่าสนับสนุนขั้นต่ำที่เหมาะสม	มาก	มาก	น้อย	น้อย	น้อย	น้อย	น้อย	มาก	น้อย
จำนวนชิ้นข้อมูลในฐานข้อมูลที่เหมาะสม	น้อย	น้อย	น้อย	น้อย	น้อย	มาก	มาก	น้อย	มาก
อัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏอยู่ในรายการข้อมูลที่เหมาะสม	น้อย	มาก	มาก	น้อย	น้อย	มาก	มาก	มาก	มาก

4.1 หลักการทำงานขั้นตอนวิธี AMFIST (AMFIST Algorithm)

ขั้นตอนวิธี AMFIST หรือขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน คือขั้นตอนวิธีที่สร้างขึ้นเพื่อเป็นทางเลือกหนึ่งสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยการทำงานของขั้นตอนวิธี AMFIST ได้คำนึงถึงการหลีกเหลี่ยงการสร้างกลุ่มข้อมูลท้าชิงเพื่อให้สามารถลดระยะเวลาและเนื้อที่หน่วยความจำที่ต้องสูญเสียไปในระหว่างการสร้างกลุ่มข้อมูลท้าชิง การอ่านข้อมูลจาก

ฐานข้อมูลเพียงครั้งเดียวเพื่อลดระยะเวลาการทำงานของอุปกรณ์นำเข้าข้อมูล (Input Device) จัดเก็บข้อมูลที่อ่านจากฐานข้อมูลมาเก็บอยู่ในรูปแบบบิตเพื่อลดเนื้อที่ในหน่วยความจำ ใช้การดำเนินการระดับบิตในการนับความถี่และสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพื่อให้สามารถทำงานได้อย่างรวดเร็ว สุดท้ายรองรับรายการข้อมูลที่คล้ายคลึงกัน คือ การรวมรายการข้อมูลที่ซ้ำกันหลังจากกำจัดชิ้นข้อมูลที่ปรากฏร่วมกันไม่บ่อยเพื่อลดจำนวนของรายการข้อมูลก่อนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทำให้สามารถลดระยะเวลาในการทำงานโดยรวมได้อีกทางหนึ่ง ทำให้ขั้นตอนวิธี AMFIST เป็นขั้นตอนวิธีที่มีประสิทธิภาพ

การทำงานของขั้นตอนวิธี AMFIST โดยหลักๆ ประกอบด้วยขั้นตอนการทำงานทั้งหมด 2 ขั้นตอนด้วยกัน ขั้นตอนแรกคือการสร้างตาราง ItemTable และตาราง BitTable และขั้นตอนที่สองคือการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากตาราง BitTable และตาราง ItemTable ที่สร้างขึ้นในขั้นตอนแรก

4.1.1 การสร้างตาราง ItemTable และตาราง BitTable

การทำงานในขั้นตอนนี้จะเริ่มจากการอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียวในการสร้างตาราง ItemTable และตาราง BitTable เพื่อนำตารางทั้งสองนี้มาใช้ในการค้นหาค่าความถี่ของกลุ่มข้อมูลและสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในขั้นตอนของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยขั้นตอนการสร้างตาราง ItemTable และตาราง BitTable แสดงดังภาพประกอบ 4.1 ซึ่งสามารถอธิบายการทำงานได้ดังนี้

1) อ่านข้อมูลจากฐานข้อมูลทีละรายการข้อมูล เพื่อบันทึกแต่ละชิ้นข้อมูลในรายการข้อมูลนั้นลงในตาราง ItemTable และตาราง Temp-BitTable โดยการทำงานสำหรับการสร้างตาราง ItemTable นั้นจะเริ่มจากการบันทึกชิ้นข้อมูลในตาราง ItemTable และให้ค่าความถี่ของชิ้นข้อมูลนั้นมีค่าเป็น 1 แต่หากมีการอ่านชิ้นข้อมูลดังกล่าวเข้ามาจากฐานข้อมูลอีกก็จะเพิ่มแค่ค่าความถี่ให้กับชิ้นข้อมูลนั้นอีก 1 ส่วนการทำงานสำหรับการสร้างตาราง BitTable นั้นจะกำหนดค่าบิตมีค่าเป็น 1 ที่ตำแหน่งของ Temp-BitTable[*t*][*k*] เมื่อ *t* คือ ลำดับของรายการข้อมูลที่อ่านมาจากฐานข้อมูล และ *k* คือ ลำดับของชิ้นข้อมูลนั้นในตาราง ItemTable

2) คำนวณค่าสนับสนุนของแต่ละชิ้นข้อมูลในตาราง ItemTable หลังจากสิ้นสุดการอ่านข้อมูลจากฐานข้อมูล ซึ่งหากชิ้นข้อมูลใดมีค่าสนับสนุนน้อยกว่าค่าสนับสนุนขั้นต่ำที่กำหนดไว้ ก็จะกำจัดชิ้นข้อมูลนั้นออกจากตาราง ItemTable และตาราง Temp-BitTable

3) เรียงลำดับชิ้นข้อมูลในตาราง ItemTable ใหม่ตามค่าความถี่ของชิ้นข้อมูลจากมากไปหาน้อย และให้ค่าดัชนีแก่แต่ละชิ้นข้อมูลจากน้อยไปหามาก หลังจากนั้นเรียงลำดับชิ้นข้อมูลแต่ละรายการข้อมูลในตาราง Temp-BitTable ใหม่ตามลำดับ

ของดัชนีของชิ้นข้อมูลในตาราง ItemTable และขั้นตอนสุดท้ายรวมรายการข้อมูลที่ซ้ำกันในตาราง Temp-BitTable สร้างเป็นตารางใหม่ชื่อ BitTable

Input: Database *D*, min_sup

Output: ItemTable, BitTable

1 **For** each transaction $t \in D$

2 **For** each item $i \in t$

3 **If** $i \in$ ItemTable

4 Increment count of item *i* in ItemTable

5 **Else**

6 Insert *i* in ItemTable with count =1

7 **End If**

8 Temp-BitTable[*t*][*k*] = 1 //*k* is the k^{th} item of item *i* in ItemTable

9 **End For**

10 **End For**

11 Delete infrequent items in ItemTable and Temp-BitTable

12 Sort frequent items in ItemTable in descending order and assign an index for each item

13 Sort in ascending order to rearrange the items in BitTable by index

14 Aggregate transactions which have similar transactions in Temp-BitTable to BitTable

ภาพประกอบ 4.1 ขั้นตอนการสร้างตาราง ItemTable และตาราง BitTable ของขั้นตอนวิธี AMFIST

4.1.2 การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

หลังจากการสร้างตาราง ItemTable และตาราง BitTable แล้วนั้น ขั้นตอนต่อไปคือการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ซึ่งขั้นตอนของการทำงานจะต้องอาศัยตารางทั้งสองข้างต้นในการค้นหาความถี่ของกลุ่มข้อมูลและสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เป็นไปได้ทั้งหมด โดยขั้นตอนการทำงานแสดงดังภาพประกอบ 4.2 ถึง 4.3 และสามารถอธิบายรายละเอียดการทำงานได้ดังนี้

1) ชิ้นข้อมูลที่ปรากฏในตาราง ItemTable คือชิ้นข้อมูลที่ผ่านการตรวจสอบค่าสนับสนุนกับค่าสนับสนุนขั้นต่ำแล้ว ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล 1 ชิ้นข้อมูล (Frequent 1-Itemsets) จากตาราง ItemTable

2) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เหลือ โดยเริ่มพิจารณาจากดัชนีที่สองจนถึงดัชนีสุดท้ายในตาราง ItemTable ซึ่งการทำงานของแต่ละดัชนีจะตรวจสอบว่า ที่ตำแหน่งบิตของดัชนีดังกล่าวมีค่าเป็น 1 ที่รายการข้อมูลใดบ้างในตาราง BitTable แล้วนำรายการข้อมูลเหล่านั้นบันทึกลงในตาราง BitTableIndex เพื่อใช้ในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยต่อไป

3) ใช้การดำเนินการระดับบิต AND กับแต่ละรายการข้อมูลในตาราง BitTableIndex เพื่อสร้างกลุ่มข้อมูลที่มีความเป็นไปได้ที่จะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่แฝงอยู่ในรายการข้อมูลเหล่านั้น และสามารถหาค่าความถี่ของรายการข้อมูลที่ดำเนินการระดับบิต AND นั้นได้โดยการนำค่าความถี่ของแต่ละรายการข้อมูลที่ดำเนินการระดับบิต AND มารวมกันเป็นค่าความถี่ของกลุ่มข้อมูลที่สร้างขึ้น สาเหตุที่ต้องรวมค่าความถี่ของแต่ละรายการข้อมูลนั้นเนื่องจาก กลุ่มข้อมูลที่สร้างขึ้นเป็นกลุ่มข้อมูลที่เกิดในทุกๆ รายการข้อมูล จากนั้นเพิ่มกลุ่มข้อมูลดังกล่าวลงในตาราง BitTableIndex และใช้การดำเนินการระดับบิต XOR เพื่อตรวจสอบรายการข้อมูลว่ามีรายการข้อมูลใดบ้างที่ซ้ำกันในตาราง BitTableIndex โดยเลือกรายการข้อมูลที่มีค่าความถี่มากที่สุด แล้วตัดรายการข้อมูลที่มีค่าความถี่ที่น้อยกว่าที่เหลือออกไป

4) ตรวจสอบค่าสนับสนุนของแต่ละรายการข้อมูลจากค่าความถี่ในตาราง BitTableIndex ซึ่งถ้าหากค่าสนับสนุนที่คำนวณได้ของแต่ละรายการข้อมูลมีค่าน้อยกว่าค่าสนับสนุนขั้นต่ำก็จะกำจัดรายการข้อมูลนั้นออกไปจากตาราง BitTableIndex

5) ใช้การดำเนินการระดับบิต OR กับบิตทุกตัวของแต่ละรายการข้อมูลที่มีค่าเป็น 1 ในตาราง BitTableIndex เพื่อสร้างกลุ่มข้อมูลทุกกรณีที่มีความเป็นไปได้ที่จะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากบิตทุกตัวเหล่านั้น และกลุ่มข้อมูลที่สร้างขึ้นเหล่านั้นจะมีค่าความถี่เท่ากับค่าความถี่ของรายการข้อมูลที่กำลังพิจารณา เนื่องจากกลุ่มข้อมูลที่สร้างขึ้นนั้นเกิดจากรายการข้อมูลดังกล่าวทำให้มีค่าความถี่เท่ากับรายการข้อมูลนั้น เพิ่มกลุ่มข้อมูลดังกล่าวในตาราง BitTableIndex

6) ผลลัพธ์ที่ได้ทั้งหมดหลังจากเพิ่มกลุ่มข้อมูลที่ดำเนินการระดับบิต OR ในตาราง BitTableIndex แล้วนั้น คือกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมดที่ค้นหาได้จากดัชนีดังกล่าว

```
Input: ItemTable, BitTable, min_sup
Output: Frequent Itemsets
1   Write each item in ItemTable and their count  // frequent 1-itemsets
2   For each index i ∈ ItemTable from the 2th Index to the last Index
3     For each transaction t ∈ BitTable
4         If (ith bit of t =1)
5             Insert bit 1st to ith of t in BitTableIndex and their count
6         End If
7     End For
8     Aggregate transactions which have similar transactions in BitTableIndex
9     MiningFrequentItemsets(i, BitTableIndex, ItemTable, min_sup)
10  End For

Procedure MiningFrequentItemsets (i, BitTableIndex, ItemTable, min_sup)
11  If i = 2
12      Write 1st and 2nd items in ItemTable and its count
13  Else If (the number of transactions in BitTableIndex=1)and(its support≥min_sup)
14          CreatingFrequentItemsets(i, SetOfBits, ItemTable, min_sup) //SetOfBit is
                all of bits of this transaction in BitTableIndex
15      Else If (the number of transactions in BitTableIndex ≥ 2)
16          ANDTrans = ∩t ⊆ P(BitTableIndex) //not include the transaction itself and ∅
17          Insert ANDTrans in new transaction of BitTableIndex
18          Delete transactions in BitTableIndex which have support < min_sup
19          Delete transactions which have same SetOfBits in BitTableIndex
                select the maximum count in those transactions and stored in
                BitTableIndex
20          For each transaction t ∈ BitTableIndex
21              CreatingFrequentItemsets(i, SetOfBits, ItemTable, min_sup)
22          End For
```

ภาพประกอบ 4.2 ขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย
ของขั้นตอนวิธี AMFIST

```
23          Delete transactions which have same SetOfBits in BitTableIndex
               select the maximum count in those transactions and stored in
               BitTableIndex
24       End Else
25       For each transaction t ∈ BitTableIndex
26          Write all frequent itemsets in BitTableIndex
27       End For
28    End Else
29    End If

Procedure CreatingFrequentItemsets(i, SetOfBits, ItemTable, min_sup)
30    ORBits = ∪_{i ⊆ P(SetOfBits)}  //not include the bit itself and ∅
31    If (ORBit.sup ≥ min_sup)
32       Insert ORBits in new transaction of BitTableIndex
33       Delete transactions which have same SetOfBits in BitTableIndex select
         the maximum count in those transactions and stored in BitTableIndex
34    End If
```

ภาพประกอบ 4.3 ขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย
ของขั้นตอนวิธี AMFIST (ต่อ)

4.2 ตัวอย่างการทำงานของขั้นตอนวิธี AMFIST

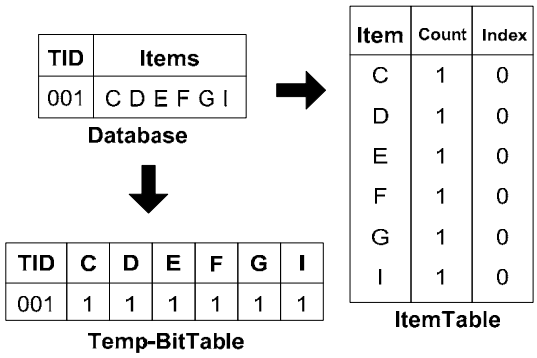
เพื่อให้เข้าใจถึงกระบวนการทำงานของขั้นตอนวิธี AMFIST ได้อย่างชัดเจนขึ้น จึงขอยกตัวอย่างการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธีนี้ โดยใช้ฐานข้อมูลการซื้อสินค้าในตารางที่ 2.1 ประกอบการอธิบาย พร้อมทั้งกำหนดค่าสนับสนุนขั้นต่ำเป็น 0.4 (มีค่าความถี่อย่างน้อยเท่ากับ 2 รายการข้อมูลปรากฏในฐานข้อมูล) โดยตัวอย่างการทำงานของขั้นตอนวิธี AMFIST มีขั้นตอนการทำงานดังนี้

4.2.1 การสร้างตาราง ItemTable และตาราง BitTable

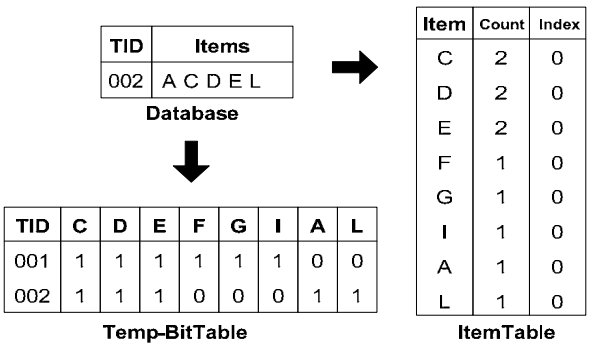
การทำงานในขั้นตอนนี้จะเริ่มจากการอ่านข้อมูลจากฐานข้อมูลเพียงหนึ่งครั้งในการสร้างตาราง ItemTable และตาราง BitTable เพื่อนำตารางทั้งสองนี้มาใช้ในการค้นหา

ค่าความถี่ของกลุ่มข้อมูลและสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ซึ่งสามารถอธิบายการทำงานได้ดังนี้

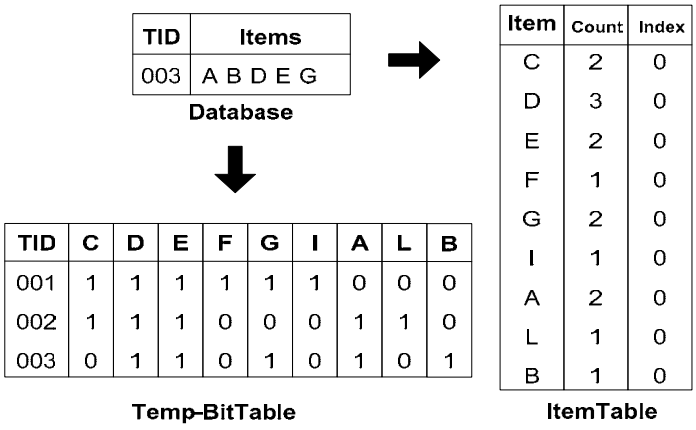
1) อ่านชิ้นข้อมูลจากฐานข้อมูลทีละรายการข้อมูล โดยเริ่มจากรายการข้อมูลแรกซึ่งประกอบด้วยชิ้นข้อมูล C D E F G และชิ้นข้อมูล I นำชิ้นข้อมูลเหล่านั้นบันทึกลงในตาราง ItemTable พร้อมทั้งให้ค่าความถี่แก่ชิ้นข้อมูลเหล่านั้นเท่ากับ 1 และกำหนดค่าดัชนีเป็น 0 หลังจากนั้นบันทึกค่าบิตในตาราง Temp-BitTable โดยให้มีค่าบิตเป็น 1 ที่ตำแหน่งแถวแรก ณ ตำแหน่งบิตที่มีชิ้นข้อมูลนั้น ซึ่งตำแหน่งบิตของแต่ละชิ้นข้อมูลจะขึ้นอยู่กับลำดับของชิ้นข้อมูลในตาราง ItemTable ยกตัวอย่างเช่น ชิ้นข้อมูล C ซึ่งเป็นชิ้นข้อมูลแรกที่อ่านจากฐานข้อมูล ดังนั้นชิ้นข้อมูล C จะมีลำดับในตาราง ItemTable เป็นลำดับที่หนึ่ง ทำให้มีตำแหน่งบิตในตาราง Temp-BitTable เป็นตำแหน่งบิตแรกเช่นกัน จึงทำให้สามารถให้ค่าบิตแก่ชิ้นข้อมูล C เพื่อบันทึกในตาราง Temp-BitTable โดยให้แถวแรกและตำแหน่งบิตที่หนึ่งมีค่าบิตเป็น 1 เพื่อแทนชิ้นข้อมูล C ที่อ่านจากรายการข้อมูลแรกของฐานข้อมูล และเมื่อบันทึกทุกชิ้นข้อมูลที่อ่านจากรายการข้อมูลแรกในตาราง ItemTable และตาราง Temp-BitTable จะสามารถแสดงผลลัพธ์ที่ได้ดังภาพประกอบ 4.4 และเมื่อทำเช่นนี้กับทุกรายการข้อมูลในฐานข้อมูล จะสามารถแสดงผลลัพธ์ที่ได้ทั้งหมดดังภาพประกอบ 4.4 ถึง 4.8



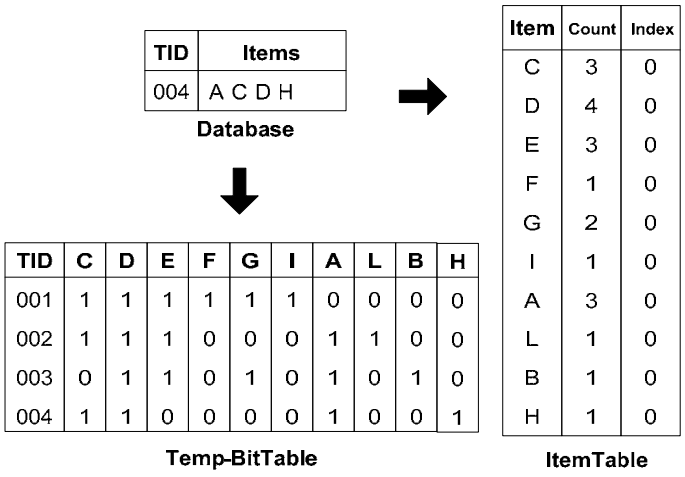
ภาพประกอบ 4.4 การอ่านรายการข้อมูลแรกจากฐานข้อมูลของขั้นตอนวิธี AMFIST



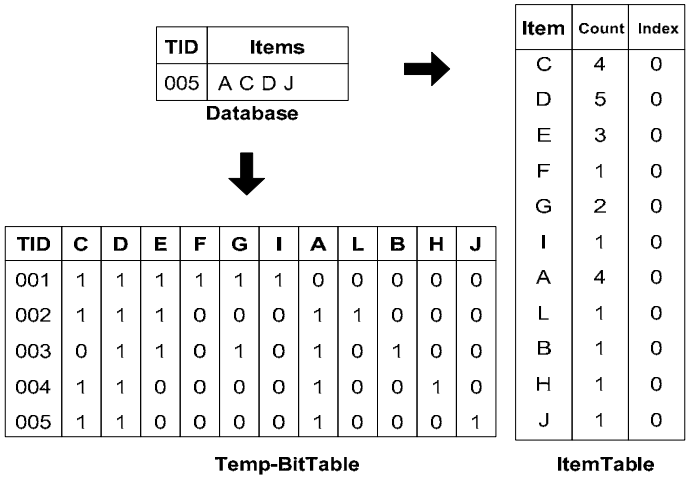
ภาพประกอบ 4.5 การอ่านรายการข้อมูลที่สองจากฐานข้อมูลของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.6 การอ่านรายการข้อมูลที่สามจากฐานข้อมูลของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.7 การอ่านรายการข้อมูลที่สี่จากฐานข้อมูลของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.8 การอ่านรายการข้อมูลที่ห้าจากฐานข้อมูลของขั้นตอนวิธี AMFIST

2) คำนวณค่าสนับสนุนจากค่าความถี่ของแต่ละชิ้นข้อมูลในตาราง ItemTable ซึ่งหากชิ้นข้อมูลใดมีค่าสนับสนุนน้อยกว่าค่าสนับสนุนขั้นต่ำที่กำหนดไว้ ก็ตัดชิ้นข้อมูลนั้นออกจากตาราง ItemTable และตาราง Temp-BitTable ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 4.9

TID	C	D	E	G	A
001	1	1	1	1	0
002	1	1	1	0	1
003	0	1	1	1	1
004	1	1	0	0	1
005	1	1	0	0	1

Temp-BitTable

Item	Count	Index
C	4	0
D	5	0
E	3	0
G	2	0
A	4	0

ItemTable

ภาพประกอบ 4.9 ตาราง ItemTable และตาราง Temp-BitTable หลังตัดชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำของขั้นตอนวิธี AMFIST

3) เรียงชิ้นข้อมูลในตาราง ItemTable ใหม่โดยเรียงชิ้นข้อมูลจากค่าความถี่มากไปหาค่าความถี่น้อย พร้อมทั้งให้ค่าดัชนีแก่แต่ละชิ้นข้อมูล หลังจากนั้นแก้ไขตำแหน่งบิตของชิ้นข้อมูลในตาราง Temp-BitTable ใหม่โดยเรียงลำดับบิตตามค่าดัชนีของชิ้นข้อมูลในตาราง ItemTable และรวมรายการข้อมูลในตาราง Temp-BitTable ที่มีชิ้นข้อมูลปรากฏในรายการข้อมูลซ้ำกันนั้นเข้าด้วยกัน และเพิ่มค่าความถี่ของชิ้นข้อมูลตามจำนวนของรายการข้อมูลที่นำมารวมกันสร้างเป็นตารางใหม่ชื่อ ตาราง BitTable ซึ่งจากภาพประกอบ 4.9 จะเห็นได้ว่ามีรายการข้อมูลที่สี่และรายการข้อมูลที่ห้ามีเซตของบิตที่ซ้ำกัน ดังนั้นจึงนำ 2 รายการข้อมูลนี้มารวมกันและเพิ่มค่าความถี่เป็น 2 สามารถแสดงผลลัพธ์ที่ได้ดังภาพประกอบ 4.10

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	1	1	1	0	1
3	1	0	1	1	1	1
4	1	1	1	0	0	2

BitTable

Item	Count	Index
D	5	1
C	4	2
A	4	3
E	3	4
G	2	5

ItemTable

ภาพประกอบ 4.10 ตาราง ItemTable และตาราง BitTable หลังรวมรายการข้อมูลที่ซ้ำกันของขั้นตอนวิธี AMFIST

4.2.2 การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

หลังจากการสร้างตาราง ItemTable และตาราง BitTable แล้วนั้น ขั้นตอนต่อไปคือการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยการทำงานสามารถอธิบายรายละเอียดได้ดังนี้

1) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล 1 ชิ้นข้อมูลจากตาราง ItemTable เนื่องจากชิ้นข้อมูลทั้งหมดในตาราง ItemTable ได้ผ่านการตรวจสอบค่าสนับสนุนแล้ว ซึ่งมีค่าไม่น้อยกว่าค่าสนับสนุนขั้นต่ำที่ได้กำหนดไว้ ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล 1 ชิ้นข้อมูลดังนี้ D(5) C(4) A(4) E(3) และ G(2) โดยค่าในวงเล็บหลังชิ้นข้อมูลคือค่าความถี่ของชิ้นข้อมูลเหล่านั้น

2) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่เหลือ โดยเริ่มต้นจากชิ้นข้อมูล C หรือดัชนีที่สองในตาราง ItemTable ซึ่งต้องตรวจสอบว่ามีรายการข้อมูลใดบ้างในตาราง BitTable ที่ตำแหน่งบิตที่สองมีค่าเป็น 1 จากผลการตรวจสอบพบว่ามีทั้งหมด 3 รายการข้อมูล คือรายการข้อมูลที่หนึ่ง ที่สอง และรายการข้อมูลที่สี่ ซึ่งจะเห็นว่าทั้ง 3 รายการข้อมูลดังกล่าวมีเซตของบิตที่เหมือนกันคือ {11} ดังนั้นจึงนำทั้ง 3 รายการข้อมูลมารวมกันเป็นรายการข้อมูลเดียวและเพิ่มค่าความถี่แก่รายการข้อมูลนี้เป็น 4 ซึ่งเกิดจากการรวมค่าความถี่ของทั้ง 3 รายการข้อมูลบันทึกลงไปในตาราง BitTableIndex โดยการทำงานสามารถแสดงดังภาพประกอบ 4.11 ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่สองคือ DC(2)

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	1	1	1	0	1
3	1	0	1	1	1	1
4	1	1	1	0	0	2

BitTable

t	D	C	Count
1	1	1	4

BitTableIndex

ภาพประกอบ 4.11 ตาราง BitTableIndex ของดัชนีที่สองหลังรวมรายการข้อมูลที่ได้จากตาราง BitTable ของขั้นตอนวิธี AMFIST

4) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากชิ้นข้อมูล A หรือดัชนีที่สามในตาราง ItemTable โดยตรวจสอบรายการข้อมูลในตาราง BitTable ที่ตำแหน่งบิตที่สามที่มีค่าบิตเป็น 1 ซึ่งผลการตรวจสอบพบว่ามีทั้งหมด 3 รายการข้อมูล คือรายการข้อมูลที่สอง ที่สาม และรายการข้อมูลที่สี่ โดยมีเซตของบิตเป็น {111} {101} และ {111} ตามลำดับ ซึ่งจะเห็นว่ารายการข้อมูลที่สองและรายการข้อมูลที่สี่มีเซตของบิตที่เหมือนกันคือ {111} ดังนั้นจึง

นำทั้ง 2 รายการข้อมูลมารวมกันพร้อมทั้งนำค่าความถี่มารวมกันด้วย จะได้ว่า {111} มีค่าความถี่เป็น 3 ทำให้ในตาราง BitTableIndex มีรายการข้อมูลเป็น {111} มีค่าความถี่เป็น 3 และ {101} มีค่าความถี่เป็น 1 แสดงดังภาพประกอบ 4.12 จากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากรายการข้อมูลในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต AND กับรายการข้อมูลในตาราง BitTableIndex ผลลัพธ์ที่ได้เพิ่มเข้าไปในตาราง BitTableIndex แสดงดังภาพประกอบ 4.13 ขั้นตอนต่อไปตรวจสอบว่ามีรายการข้อมูลใดบ้างที่เหมือนกัน ซึ่งพบว่าเซตของบิตของรายการข้อมูลที่สองเหมือนกับรายการข้อมูลที่สาม ดังนั้นจึงตัดรายการข้อมูลที่ซ้ำกันออกให้เหลือเพียงรายการข้อมูลเดียวและเลือกค่าความถี่ของรายการข้อมูลที่มีค่าความถี่มากที่สุดเป็นค่าความถี่ของรายการข้อมูลที่ซ้ำกันนั้น ดังนั้นในตาราง BitTableIndex จะมีเซตของบิตของรายการข้อมูลเหลือเป็น {111} มีค่าความถี่เป็น 3 และ {101} มีค่าความถี่เป็น 4 การทำงานแสดงดังภาพประกอบ 4.14 หลังจากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากเซตของบิตของแต่ละรายการข้อมูลในตาราง BitTableIndex โดยพิจารณาบิตที่สามในแต่ละรายการข้อมูล ใช้การดำเนินการระดับบิต OR กับแต่ละบิตที่เหลือที่มีค่าบิตเป็น 1 ผลลัพธ์ที่ได้เพิ่มในตาราง BitTableIndex ซึ่งพบว่าในรายการข้อมูลที่หนึ่งสามารถสร้างกลุ่มข้อมูลได้เป็น {101} มีค่าความถี่เป็น 3 และ {011} มีค่าความถี่เป็น 3 ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 4.15 หลังจากนั้นตรวจสอบว่ามีรายการข้อมูลใดบ้างที่ซ้ำกันในตาราง BitTableIndex ซึ่งปรากฏว่ารายการข้อมูลที่สองและที่สามมีเซตของบิตที่ซ้ำกัน จึงตัดรายการข้อมูลที่มีค่าความถี่น้อยกว่าออกไป ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 4.16 ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่สามคือ DCA(3) DA(4) และ CA(3)

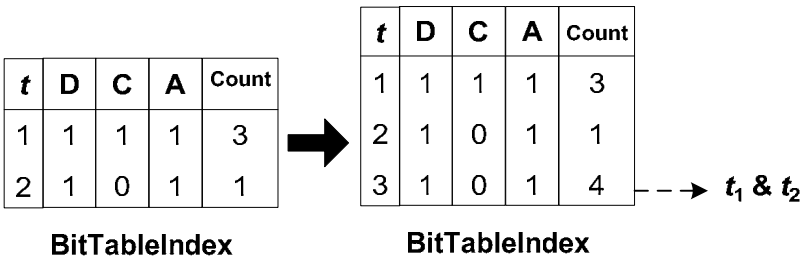
t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	1	1	1	0	1
3	1	0	1	1	1	1
4	1	1	1	0	0	2

BitTable

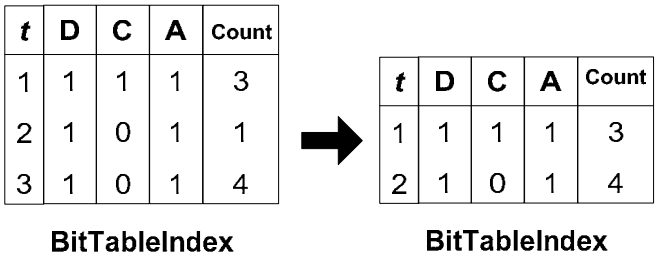
t	D	C	A	Count
1	1	1	1	3
2	1	0	1	1

BitTableIndex

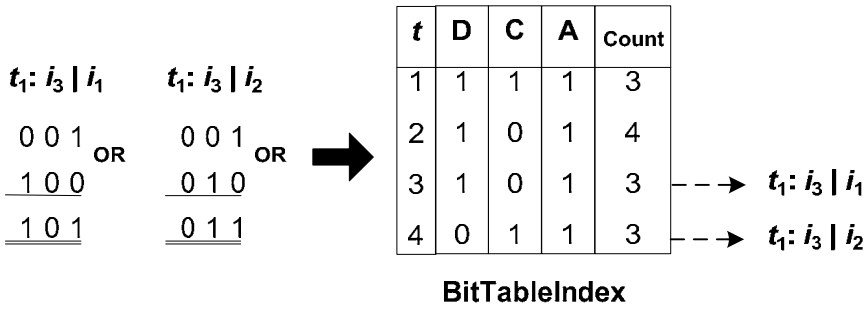
ภาพประกอบ 4.12 ตาราง BitTableIndex ของดัชนีที่สามหลังรวมรายการข้อมูลที่ซ้ำกันจากตาราง BitTable ของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.13 ตาราง BitTableIndex ของดัชนีที่สามหลังดำเนินการระดับบิต AND ของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.14 ตาราง BitTableIndex ของดัชนีที่สามหลังรวมรายการข้อมูลที่ซ้ำกันจากการดำเนินการระดับบิต AND ของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.15 ตาราง BitTableIndex ของดัชนีที่สามหลังดำเนินการระดับบิต OR ของขั้นตอนวิธี AMFIST

t	D	C	A	Count
1	1	1	1	3
2	1	0	1	4
3	1	0	1	3
4	0	1	1	3

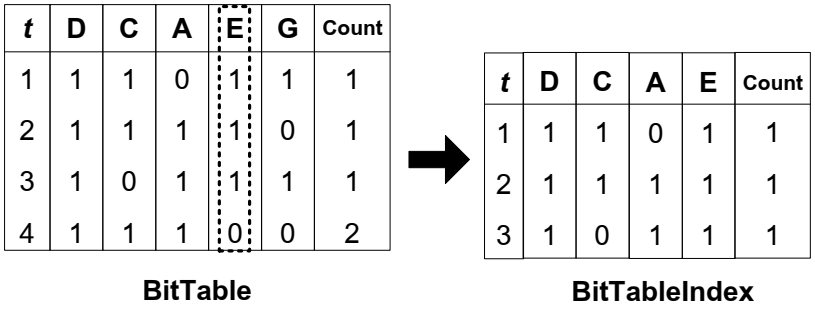
BitTableIndex

t	D	C	A	Count
1	1	1	1	3
2	1	0	1	4
3	0	1	1	3

BitTableIndex

ภาพประกอบ 4.16 ตาราง BitTableIndex ของดัชนีที่สามหลังรวมรายการข้อมูลที่ซ้ำกัน จากการดำเนินการระดับบิต OR ของขั้นตอนวิธี AMFIST

5) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากชิ้นข้อมูล E หรือดัชนีที่สี่ในตาราง ItemTable โดยตรวจสอบรายการข้อมูลในตาราง BitTable ที่ตำแหน่งบิตที่สี่ที่มีค่าบิตเป็น 1 ซึ่งผลการตรวจสอบพบว่ามีทั้งหมด 3 รายการข้อมูล คือรายการข้อมูลที่หนึ่ง ที่สอง และรายการข้อมูลที่สาม โดยมีเซตของบิตเป็น {1101} {1110} และ {1011} ตามลำดับ แสดงดังภาพประกอบ 4.17 จากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากรายการข้อมูลในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต AND กับรายการข้อมูลในตาราง BitTableIndex ผลลัพธ์ที่ได้เพิ่มเข้าไปในตาราง BitTableIndex แสดงดังภาพประกอบ 4.18 ขั้นตอนต่อไปตรวจสอบว่ามีรายการข้อมูลใดบ้างที่ซ้ำกัน ซึ่งพบว่าเซตของบิตที่เหมือนกันถึง 3 รายการข้อมูล คือรายการข้อมูลที่หนึ่งซ้ำกับรายการข้อมูลที่สี่ รายการข้อมูลที่สามซ้ำกับรายการข้อมูลที่หก รายการข้อมูลที่ห้าซ้ำกับรายการข้อมูลที่เจ็ด ดังนั้นจึงตัดรายการข้อมูลที่ซ้ำกันออกให้เหลือเพียงรายการข้อมูลเดียวและเลือกค่าความถี่ของรายการข้อมูลที่มีค่าความถี่มากที่สุดเป็นค่าความถี่ของรายการข้อมูลที่ซ้ำกันนั้น พร้อมทั้งตัดรายการข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำที่กำหนดไว้ การทำงานในส่วนนี้แสดงดังภาพประกอบ 4.19 หลังจากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากเซตของบิตของแต่ละรายการข้อมูลในตาราง โดยพิจารณาบิตที่สี่ในแต่ละรายการข้อมูล ใช้การดำเนินการระดับบิต OR กับแต่ละบิตที่เหลือที่มีค่าบิตเป็น 1 ผลลัพธ์ที่ได้เพิ่มในตาราง BitTableIndex ซึ่งพบว่าในรายการข้อมูลที่หนึ่งสามารถสร้างกลุ่มข้อมูลได้เป็น {1001} มีค่าความถี่เป็น 2 และ {0101} มีค่าความถี่เป็น 2 ส่วนรายการข้อมูลที่สองสามารถสร้างกลุ่มข้อมูลได้เป็น {1001} มีค่าความถี่เป็น 2 และ {0011} มีค่าความถี่เป็น 2 ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 4.20 หลังจากนั้นตรวจสอบว่ามีรายการข้อมูลใดบ้างที่ซ้ำกัน ซึ่งปรากฏว่ารายการข้อมูลที่สาม ที่สี่ และรายการข้อมูลที่หกมีเซตของบิตที่ซ้ำกันจึงตัดรายการข้อมูลที่มีค่าความถี่น้อยกว่าออกไป ผลลัพธ์ที่ได้แสดงดังภาพประกอบ 4.21 ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่สี่คือ DCE(2) DAE(2) DE(3) CE(2) และ AE(2)



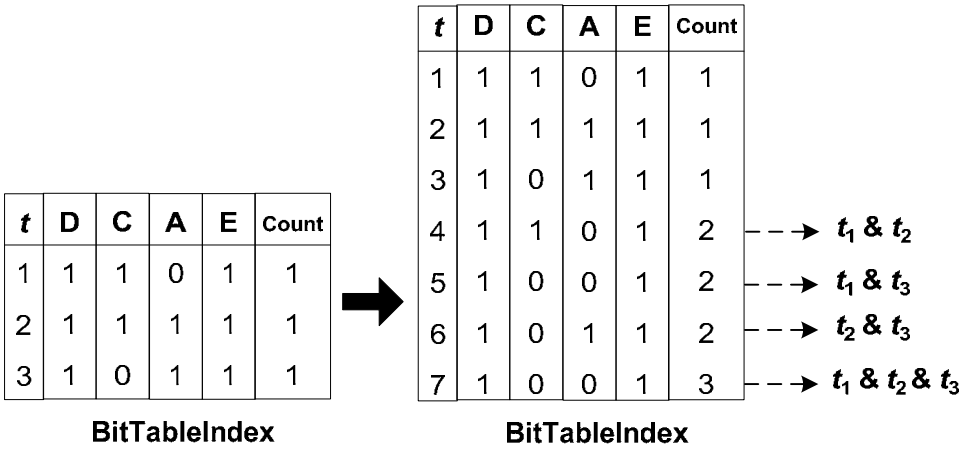
t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	1	1	1	0	1
3	1	0	1	1	1	1
4	1	1	1	0	0	2

BitTable

t	D	C	A	E	Count
1	1	1	0	1	1
2	1	1	1	1	1
3	1	0	1	1	1

BitTableIndex

ภาพประกอบ 4.17 ตาราง BitTableIndex ของดัชนีที่สี่ที่ได้จากตาราง BitTable ของขั้นตอนวิธี AMFIST



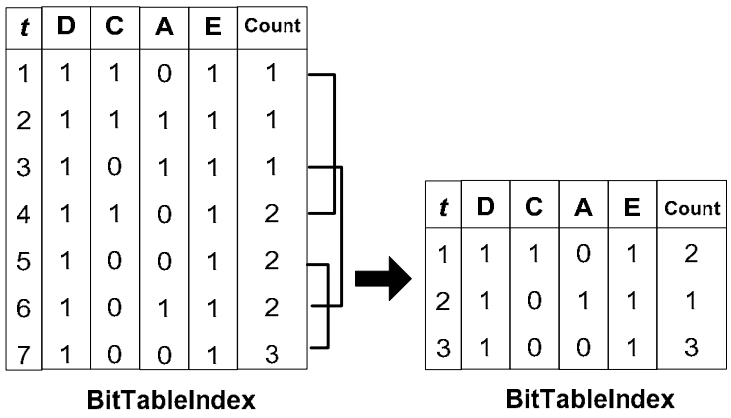
t	D	C	A	E	Count
1	1	1	0	1	1
2	1	1	1	1	1
3	1	0	1	1	1

BitTableIndex

t	D	C	A	E	Count	
1	1	1	0	1	1	
2	1	1	1	1	1	
3	1	0	1	1	1	
4	1	1	0	1	2	t_1 & t_2
5	1	0	0	1	2	t_1 & t_3
6	1	0	1	1	2	t_2 & t_3
7	1	0	0	1	3	t_1 & t_2 & t_3

BitTableIndex

ภาพประกอบ 4.18 ตาราง BitTableIndex ของดัชนีที่สี่หลังดำเนินการระดับบิต AND ของขั้นตอนวิธี AMFIST



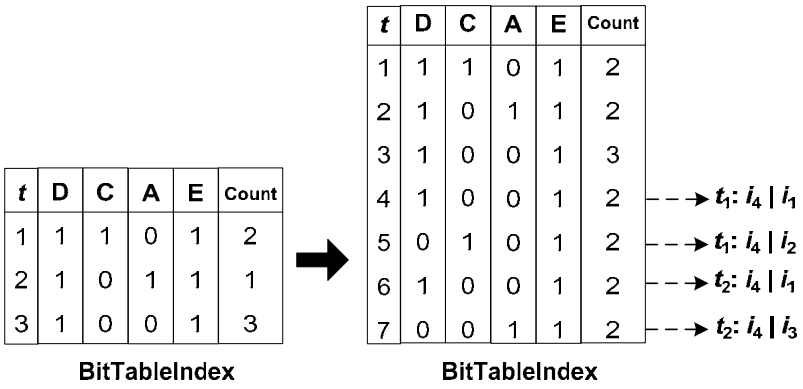
t	D	C	A	E	Count
1	1	1	0	1	1
2	1	1	1	1	1
3	1	0	1	1	1
4	1	1	0	1	2
5	1	0	0	1	2
6	1	0	1	1	2
7	1	0	0	1	3

BitTableIndex

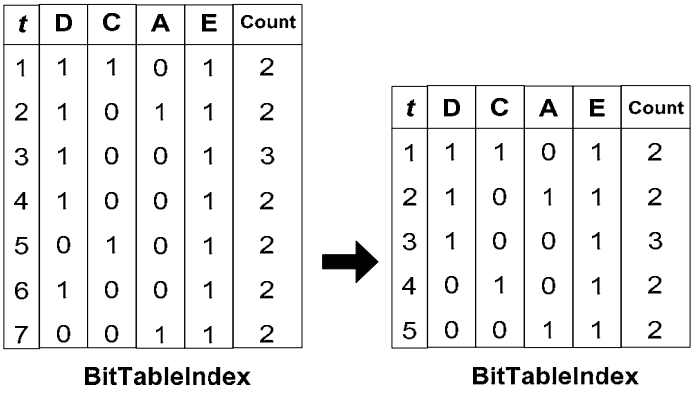
t	D	C	A	E	Count
1	1	1	0	1	2
2	1	0	1	1	1
3	1	0	0	1	3

BitTableIndex

ภาพประกอบ 4.19 ตาราง BitTableIndex ของดัชนีที่สี่หลังรวมรายการข้อมูลที่ซ้ำกันจากการดำเนินการระดับบิต AND ของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.20 ตาราง BitTableIndex ของดัชนีที่สี่หลังการดำเนินการระดับบิต OR ของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.21 ตาราง BitTableIndex ของดัชนีที่สี่หลังรวมรายการข้อมูลที่ซ้ำกันจากการดำเนินการระดับบิต OR ของขั้นตอนวิธี AMFIST

6) ค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากชิ้นข้อมูล G หรือดัชนีที่ห้าในตาราง ItemTable โดยตรวจสอบรายการข้อมูลในตาราง BitTable ที่ตำแหน่งบิตที่ห้าที่มีค่าบิตเป็น 1 ซึ่งผลการตรวจสอบพบว่ามีทั้งหมด 2 รายการข้อมูล คือรายการข้อมูลที่หนึ่ง และรายการข้อมูลที่สาม โดยมีเซตของบิตเป็น {11011} และ {10111} แสดงดังภาพประกอบ 4.22 จากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากรายการข้อมูลในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต AND รายการข้อมูลที่หนึ่งกับรายการข้อมูลที่สอง ผลลัพธ์ที่ได้คือ {10011} มีค่าความถี่เป็น 2 แสดงดังภาพประกอบ 4.23 และตัดรายการข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำที่กำหนดไว้ แสดงดังภาพประกอบ 4.24 หลังจากนั้นสร้างกลุ่มข้อมูลที่คาดว่าจะเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากเซตของบิตของรายการข้อมูลในตาราง BitTableIndex โดยพิจารณาบิตที่ห้าในแต่ละรายการข้อมูล ใช้การ

ดำเนินการระดับบิต OR กับแต่ละบิตที่เหลือที่มีค่าบิตเป็น 1 ผลลัพธ์ที่ได้เพิ่มในตาราง BitTableIndex ซึ่งพบว่าในรายการข้อมูลที่หนึ่งสามารถสร้างกลุ่มข้อมูลได้เป็น {10001} มีค่าความถี่เป็น 2 และ {00011} มีค่าความถี่เป็น 2 การทำงานในส่วนนี้แสดงดังภาพประกอบ 4.25 หลังจากนั้นตรวจสอบว่ามีรายการข้อมูลใดบ้างที่ซ้ำกัน ซึ่งปรากฏว่าไม่มีรายการข้อมูลใดซ้ำกัน ดังนั้นจะได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่ห้าคือ DEG(2) DG(2) และ EG(2) และสามารถสรุปกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ค้นหาได้ดังตารางที่ 4.2

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	1	1	1	0	1
3	1	0	1	1	1	1
4	1	1	1	0	0	2

BitTable

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	0	1	1	1	1

BitTableIndex

ภาพประกอบ 4.22 ตาราง BitTableIndex ของดัชนีที่ห้าที่ได้จากตาราง BitTable ของขั้นตอนวิธี AMFIST

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	0	1	1	1	1

BitTableIndex

t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	0	1	1	1	1
3	1	0	0	1	1	2

t_1 & t_2

BitTableIndex

ภาพประกอบ 4.23 ตาราง BitTableIndex ของดัชนีที่ห้าหลังการดำเนินการระดับบิต AND ของขั้นตอนวิธี AMFIST

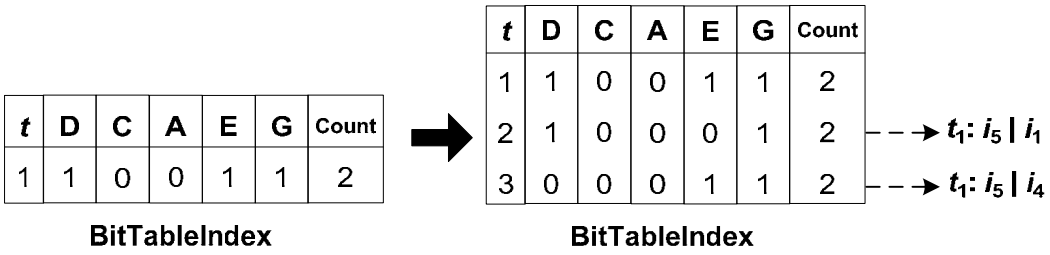
t	D	C	A	E	G	Count
1	1	1	0	1	1	1
2	1	0	1	1	1	1
3	1	0	0	1	1	2

BitTableIndex

t	D	C	A	E	G	Count
1	1	0	0	1	1	2

BitTableIndex

ภาพประกอบ 4.24 ตาราง BitTableIndex ของดัชนีที่ห้าหลังตัดรายการข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำของขั้นตอนวิธี AMFIST



ภาพประกอบ 4.25 ตาราง BitTableIndex ของดัชนีที่ห้าหลังการดำเนินการระดับบิต OR ของขั้นตอนวิธี AMFIST

ตารางที่ 4.2 ผลลัพธ์การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AM-FIST

Index	Frequent Itemsets
L_1	D(5), C(4), A(4), E(3), G(2)
2	DC(2)
3	DCA(3), DA(4), CA(3)
4	DCE(2), DAE(2), DE(3), CE(2), AE(2)
5	DEG(2), DG(2), EG(2)

4.3 ข้อดีของขั้นตอนวิธี AMFIST

ขั้นตอนวิธี AMFIST สามารถทำงานได้ดีทั้งฐานข้อมูลที่มีขนาดเล็กและขนาดใหญ่เนื่องจากการเก็บข้อมูลในระหว่างการประมวลผลอยู่ในรูปแบบบิต และมีการรวมรายการข้อมูลที่ซ้ำกันทำให้ลดจำนวนของรายการข้อมูลและยังช่วยลดจำนวนเนื้อที่ในหน่วยความจำ อีกทั้งยังใช้เวลาที่ในการทำงานน้อยลงเนื่องจากใช้การดำเนินการระดับบิตในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยซึ่งทำให้การทำงานในการค้นหากลุ่มข้อมูลและนับค่าความถี่นั้นเป็นไปอย่างรวดเร็ว

4.4 ข้อเสียของขั้นตอนวิธี AMFIST

ขั้นตอนวิธี AMFIST ถูกออกแบบมาเพื่อรองรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีรายการข้อมูลคล้ายคลึงกันจำนวนมากในฐานข้อมูล ซึ่งหากฐานข้อมูลที่นำมาค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นมีจำนวนของรายการข้อมูลที่คล้ายคลึงกันมีน้อย ก็จะไม่เหมาะสมหากใช้ขั้นตอนวิธี AMFIST

บทที่ 5

การวิเคราะห์และผลการศึกษา

สำหรับบทนี้จะกล่าวถึงการวัดประสิทธิภาพการทำงานของขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกันหรือขั้นตอนวิธี AMFIST (AMFIST Algorithm) ที่คิดค้นขึ้น เปรียบเทียบกับขั้นตอนวิธี Apriori ซึ่งเป็นขั้นตอนวิธีที่ได้รับความนิยมและเป็นที่ยอมรับสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ขั้นตอนวิธี Index-BitTableFI ซึ่งเป็นขั้นตอนวิธีที่มีลักษณะการจัดเก็บข้อมูลในลักษณะเช่นเดียวกันกับขั้นตอนวิธี AMFIST และผู้วิจัยได้สร้างอีกขั้นตอนวิธีหนึ่งขึ้น เพื่อใช้ในการวัดประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST โดยประยุกต์การทำงานระหว่างขั้นตอนวิธี Apriori กับขั้นตอนวิธี AMFIST ซึ่งใช้ชื่อว่า ขั้นตอนวิธี Modified-Apriori

การทำงานของขั้นตอนวิธี Modified-Apriori จะมีลักษณะการทำงานที่เหมือนกับขั้นตอนวิธี AMFIST ในส่วนของการสร้างตาราง BitTable และตาราง ItemTable เพื่อลดจำนวนการอ่านข้อมูลจากฐานข้อมูลเหลือเพียงครั้งเดียวและลดเนื้อที่หน่วยความจำในการจัดเก็บข้อมูลในระหว่างการประมวลผล แต่จะยังคงใช้หลักการทำงานของขั้นตอนวิธี Apriori ในส่วนของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ที่มีการสร้างกลุ่มข้อมูลท้าชิงและทดสอบกลุ่มข้อมูลท้าชิงเหล่านั้นว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่

การวัดประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST จะแบ่งออกเป็น 2 ส่วนคือ การวิเคราะห์ประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST ด้วยค่า Big-O ซึ่งจะเป็นการวิเคราะห์เวลาการทำงานในกรณีที่ดีที่สุด (Best Case) กรณีเฉลี่ย (Average Case) และกรณีที่แย่ที่สุด (Worst Case) ของขั้นตอนวิธี AMFIST ขั้นตอนวิธี Apriori ขั้นตอนวิธี Modified-Apriori และขั้นตอนวิธี Index-BitTableFI อีกทั้งยังมีการทดสอบระยะเวลาที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori

5.1 นิยามคำศัพท์และสัญลักษณ์ที่เกี่ยวข้อง

ก่อนการวิเคราะห์ประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST ด้วยค่า Big-O ผู้วิจัยขอให้นิยาม คำศัพท์ และสัญลักษณ์ที่เกี่ยวข้องในการวัดประสิทธิภาพ ดังนี้

1) กำหนดให้ เซตของบิต คือกลุ่มของชิ้นข้อมูลในรายการข้อมูลที่อยู่ในรูปแบบของบิต

2) กำหนดให้ a คือจำนวนของชิ้นข้อมูลที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ (Frequent 1-Itemsets)

3) กำหนดให้ b คือจำนวนรายการข้อมูลในตาราง BitTable

4) กำหนดให้ i คือดัชนีในตาราง ItemTable ที่กำลังพิจารณา

5) กำหนดให้ j คือกลุ่มข้อมูลที่มีขนาดของชิ้นข้อมูล j ชิ้นข้อมูล (จำนวนของชิ้นข้อมูลในกลุ่มข้อมูล)

6) กำหนดให้ l_k คือจำนวนบิต k บิตที่มีตำแหน่งบิตมีค่าของบิตเป็น 1 ในเซตของบิตที่ l

7) กำหนดให้ c_j คือจำนวนของกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล j ชิ้นข้อมูล (Candidate j-Itemsets)

8) กำหนดให้ d_j คือจำนวนของกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล j ชิ้นข้อมูล (Frequent j-Itemsets)

9) กำหนดให้ e_j คือจำนวนของสมาชิกของ Subsume ใน Index Array ที่มีสมาชิก e ตัวสำหรับดัชนี i

10) กำหนดให้ m คือจำนวนชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูล

11) กำหนดให้ n คือจำนวนรายการข้อมูลทั้งหมดในฐานข้อมูล

12) กำหนดให้ p คือผลรวมของจำนวนชิ้นข้อมูลที่มีอยู่ในฐานข้อมูล

13) กำหนดให้ q_i คือจำนวนรายการข้อมูลในตาราง BitTable ที่มีดัชนี i ปรากฏอยู่ในรายการข้อมูลนั้น

14) กำหนดให้ r_i คือจำนวนรายการข้อมูลที่ได้หลังการรวมรายการข้อมูลที่ซ้ำกันในตาราง BitTableIndex ของดัชนี i

15) กำหนดให้ $x!$ (x Factorial) สำหรับจำนวนเต็ม $x \geq 0$ (Bogart and Stein, 2002) โดย

$$0! = 1$$
$$x! = x(x\text{-}1)!$$

16) กำหนดให้ $C(x,y)$ แทนจำนวนในการเลือกของ y สิ่งจากของทั้งหมด x สิ่ง สำหรับ $0 \le y \le x$ (Bogart and Stein, 2002; Simovici and Djeraba, 2008) โดย

$$C(x,y) = \frac{x(x-1)\cdots(x-y+1)(x-y)\cdots 2.1}{y!(x-y)!}$$
$$= \frac{x!}{y!(x-y)!}$$

บางครั้งอาจเขียนแทนด้วยสัญลักษณ์ $C(n,r)$ ด้วย ${}^{x}C_{y}$ หรือ $\binom{x}{y}$ ซึ่งสัญลักษณ์ $\binom{x}{y}$ เป็นที่นิยมใช้กันมาก

17) กำหนดให้ $\sum_{i=1}^{y} x_i$ แทนผลรวมของ x จาก x_1 จนถึง x_y มีจำนวนทั้งหมด y ตัว จะได้

$$\sum_{i=1}^{y} x_i = x_1 + x_2 + x_3 + \cdots + x_y$$

18) กำหนดให้ $P(x)$ แทนเพาเวอร์เซตของ x หากจำนวนของสมาชิกของ x มีทั้งหมด y ตัว จะได้

$$P(x) = 2^y$$

5.2 การวิเคราะห์ประสิทธิภาพการทำงานของขั้นตอนวิธีด้วยค่า Big-O

การวัดประสิทธิภาพการทำงานของขั้นตอนวิธีด้วยค่า Big-O เป็นวิธีการหนึ่งที่ทำให้ทราบถึงประสิทธิภาพของขั้นตอนวิธี โดยในหัวข้อนี้จะกล่าวถึงการหาค่า Big-O ในกรณีที่ดีที่สุด กรณีเฉลี่ย และกรณีที่แย่ที่สุดของขั้นตอนวิธี AMFIST ขั้นตอนวิธี Apriori ขั้นตอนวิธี Modified-Apriori และขั้นตอนวิธี Index-BitTableFI ซึ่งแสดงได้ดังนี้

5.2.1 ค่า Big-O ของขั้นตอนวิธี AMFIST

การทำงานของขั้นตอนวิธี AMFIST จะเริ่มจากการอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียวในการสร้างตาราง ItemTable และตาราง BitTable เพื่อนำตารางทั้งสองนี้มาใช้ในการค้นหาค่าความถี่ของกลุ่มข้อมูลและสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในขั้นตอนของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยขั้นตอนวิธี AMFIST มีค่า Big-O ดังทฤษฎีบท 5.1 ถึง 5.3

<u>ทฤษฎีบท 5.1</u> ประสิทธิภาพการทำงานสำหรับ*กรณีที่ดีที่สุด*ของขั้นตอนวิธี AMFIST มีค่า Big-O เป็น $O(p)$

<u>พิสูจน์</u>

กรณีที่ดีที่สุดสำหรับการทำงานของขั้นตอนวิธี AMFIST จะเกิดขึ้นเมื่อไม่มีชิ้นข้อมูลใดมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ ทำให้การทำงานจะเกิดขึ้นเพียงขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable เท่านั้นโดยประกอบด้วยการทำงาน ดังนี้

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลลงในตาราง ItemTable เป็น m
- บันทึกทุกชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p
- ลบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำออกจากตาราง ItemTable และตาราง Temp-BitTable เป็น $2m$

และเนื่องจากทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันไม่บ่อยจึงไม่สามารถดำเนินการต่อได้ในขั้นตอนของการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้ ซึ่งค่า Big-O ในส่วนนี้สามารถแสดงดังสมการที่ (5.1)

$$\begin{aligned} \text{Big-O} &= O(m+p+2m) \\ &= O(p) \end{aligned} \tag{5.1}$$

ดังนั้นสามารถสรุปได้ว่า การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST ในกรณีที่ดีที่สุดมีค่า Big-O เป็น $O(p)$ ■

<u>ทฤษฎีบท 5.2</u> ประสิทธิภาพการทำงานสำหรับ*กรณีเฉลี่ย*ของขั้นตอนวิธี AMFIST มีค่า Big-O เป็น $O(a2^{r_i})$

<u>พิสูจน์</u>

กรณีเฉลี่ยสำหรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST จะประกอบด้วยการทำงาน 2 ขั้นตอนดังนี้

1) การทำงานในการสร้างตาราง ItemTable และตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

(1.1) การสร้างตาราง ItemTable

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลลงในตาราง ItemTable เป็น m
- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง ItemTable เป็น m

- เรียงลำดับชิ้นข้อมูลในตาราง ItemTable จากค่าความถี่มากไปหาค่าความถี่น้อยเป็น $a\log_2 a$

(1.2) การสร้างตาราง BitTable

- บันทึกทุกชิ้นข้อมูลจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p

- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง Temp-BitTable เป็น m

- เรียงรายการข้อมูลในตาราง Temp-BitTable เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $n\log_2 n$

- ตรวจสอบรายการข้อมูลในตาราง Temp-BitTable ที่ซ้ำกันเป็น n

- บันทึกรายการข้อมูลจากตาราง Temp-BitTable หลังตรวจสอบรายการข้อมูลที่ซ้ำกันลงในตาราง BitTabel เป็น b

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable มีค่า Big-O เป็น $O(p)$ ซึ่งได้จากสมการ (5.2)

$$\begin{aligned}\text{Big-O} &= O(m+m+a\log_2 a+p+m+n\log_2 n+n+b)\\ &= O(3m+a\log_2 a+p+n\log_2 a+n+b)\\ &= O(p) \qquad (5.2)\end{aligned}$$

2) การทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ประกอบด้วยการทำงานดังนี้

- ตรวจสอบรายการข้อมูลในตาราง BitTable ที่มีดัชนี i ในรายการข้อมูลนั้นเป็น b

- บันทึกรายการข้อมูลที่มีดัชนี i ลงในตาราง BitTableIndex เป็น q_i

- เรียงรายการข้อมูลในตาราง BitTableIndex เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $q_i \log_2 q_i$

- รวมรายการข้อมูลในตาราง BitTableIndex ที่ซ้ำกันเป็น r_i

- สร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยด้วยการดำเนินการระดับบิต AND กับรายการข้อมูลในตาราง BitTableIndex ของดัชนี i โดยไม่รวมตัวมันเองและเซตว่างเป็น $2^{r_i} - r_i - 1$

- สร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยด้วยการดำเนินการระดับบิต OR กับเซตของบิตของแต่ละรายการข้อมูลในตาราง BitTableIndex ของดัชนี i โดยไม่รวมตัวมันเองและเซตว่างเป็น $\sum_{l=1}^{r_i}(2^{l_k} - l_k - 1)$

การทำงานในส่วนที่สองนี้เป็นการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีเพียงตัวเดียว การดำเนินงานสามารถแสดงดังสมการที่ (5.3) และเมื่อสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีทั้งหมดในตาราง ItemTable ซึ่งมีทั้งหมด a ดัชนีทำให้การสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีทั้งหมดมีค่า Big-O เป็น $O(a2^{r_i})$

$$\begin{aligned}\text{Big-O} &= O\left(b + q_i + q_i \log_2 q_i + r_i + (2^{r_i} - r_i - 1) + \sum_{l=1}^{r_i}(2^{l_k} - l_k - 1)\right) \\ &= O\left(b + q_i + q_i \log_2 q_i + 2^{r_i} - 1) + \sum_{l=1}^{r_i}(2^{l_k} - l_k - 1)\right) \\ &= O(2^{r_i}) \qquad (5.3)\end{aligned}$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST ในกรณีเฉลี่ยมีค่า Big-O เป็น $O(a2^{r_i})$ ■

<u>ทฤษฎีบท 5.3</u> ประสิทธิภาพการทำงานสำหรับ*กรณีที่แย่ที่สุด*ของขั้นตอนวิธี AMFIST มีค่า Big-O เป็น $O(m2^{r_i})$

<u>พิสูจน์</u>

กรณีที่แย่ที่สุดสำหรับการทำงานของขั้นตอนวิธี AMFIST จะเกิดขึ้นเมื่อทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย และทุกรายการข้อมูลในฐานข้อมูลไม่มีรายการข้อมูลใดซ้ำกัน โดยการทำงานทั้งหมดสามารถแสดงได้ดังนี้

1) การทำงานในการสร้างตาราง ItemTable และตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

(1.1) การสร้างตาราง ItemTable

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูลลงในตาราง ItemTable เป็น m
- เรียงลำดับชิ้นข้อมูลในตาราง ItemTable จากค่าความถี่มากไปหาค่าความถี่น้อยเป็น $m\log_2 m$

(1.2) การสร้างตาราง BitTable

- บันทึกทุกชิ้นข้อมูลจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p
- เรียงรายการข้อมูลในตาราง Temp-BitTable เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $n\log_2 n$
- ตรวจสอบรายการข้อมูลในตาราง Temp-BitTable ที่ซ้ำกันเป็น n

- บันทึกรายการข้อมูลจากตาราง Temp-BitTable หลังตรวจสอบรายการข้อมูลที่ซ้ำกันลงในตาราง BitTabel เป็น n

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable มีค่า Big-O เป็น $O(p)$ ซึ่งได้จากสมการ (5.4)

$$\begin{aligned} \text{Big-O} &= O(m + m\log_2 m + p + n\log_2 n + n + n) \\ &= O(m + m\log_2 m + p + n\log_2 n + 2n) \\ &= O(p) \end{aligned} \tag{5.4}$$

2) การทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ประกอบด้วยการทำงานดังนี้

- ตรวจสอบรายการข้อมูลในตาราง BitTable ที่มีดัชนี i ในรายการข้อมูลนั้นเป็น n
- บันทึกรายการข้อมูลที่มีดัชนี i ลงในตาราง BitTableIndex เป็น q_i
- เรียงรายการข้อมูลในตาราง BitTableIndex เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $q_i \log_2 q_i$
- สร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยด้วยการดำเนินการระดับบิต AND กับรายการข้อมูลในตาราง BitTableIndex ของดัชนี i โดยไม่รวมตัวมันเองและเซตว่างเป็น $2^{r_i} - r_i - 1$
- สร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยด้วยการดำเนินการระดับบิต OR กับเซตของบิตของแต่ละรายการข้อมูลในตาราง BitTableIndex ของดัชนี i โดยไม่รวมตัวมันเองและเซตว่างเป็น $\sum_{l=1}^{r_i}(2^{l_k} - l_k - 1)$

การทำงานในส่วนที่สองนี้เป็นการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีเพียงตัวเดียว ซึ่งการดำเนินงานข้างต้นสามารถแสดงดังสมการที่ (5.5) และเมื่อสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีทั้งหมดในตาราง ItemTable ซึ่งมีทั้งหมด m ดัชนีทำให้การสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีทั้งหมดมีค่า Big-O เป็น $O(m2^{r_i})$

$$\begin{aligned} \text{Big-O} &= O\left(n + q_i + q_i \log_2 q_i + (2^{r_i} - r_i - 1) + \sum_{l=1}^{r_i}(2^{l_k} - l_k - 1)\right) \\ &= O(2^{r_i}) \end{aligned} \tag{5.5}$$

ดังนั้นสามากสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST ในกรณีแย่ที่สุดมีค่า Big-O เป็น $O(m2^{r_i})$ ■

5.2.2 ค่า Big-O ของขั้นตอนวิธี Apriori

หลักการทำงานของขั้นตอนวิธี Apriori โดยหลักๆ แล้วจะประกอบด้วยขั้นตอนการทำงานทั้งหมด 2 ขั้นตอนด้วยกัน ขั้นตอนแรกคือการสร้างกลุ่มข้อมูลท้าชิง และขั้นตอนที่สองคือการตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเหล่านั้นว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ โดยขั้นตอนวิธี Apriori มีค่า Big-O ดังทฤษฎีบท 5.4 ถึง 5.6

ทฤษฎีบท 5.4 ประสิทธิภาพการทำงานสำหรับ*กรณีที่ดีที่สุด*ของขั้นตอนวิธี Apriori มีค่า Big-O เป็น $O(p)$

พิสูจน์

กรณีที่ดีที่สุดสำหรับการทำงานของขั้นตอนวิธี Apriori จะเกิดขึ้นเมื่อไม่มีชิ้นข้อมูลใดมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ ทำให้การทำงานจะเกิดขึ้นเพียงขั้นตอนของการอ่านข้อมูลเพื่อตรวจสอบว่าชิ้นข้อมูลใดเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย โดยการทำงานสามารถอธิบายได้ดังนี้

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูลเพื่อค้นหาค่าความถี่ของชิ้นข้อมูล เป็น p
- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเป็น m

และเนื่องจากทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันไม่บ่อยจึงไม่สามารถดำเนินการต่อได้ในขั้นตอนของการสร้างกลุ่มข้อมูลท้าชิง จึงสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.6)

$$\begin{aligned} \text{Big-O} &= O(p+m) \\ &= O(p) \end{aligned} \tag{5.6}$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Apriori ในกรณีที่ดีที่สุดมีค่า Big-O เป็น $O(p)$ ■

ทฤษฎีบท 5.5 ประสิทธิภาพการทำงานสำหรับ*กรณีเฉลี่ย*ของขั้นตอนวิธี Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$

พิสูจน์

กรณีเฉลี่ยสำหรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Apriori จะประกอบด้วยการทำงาน 2 ขั้นตอนดังนี้

1) การทำงานในการสร้างกลุ่มข้อมูลท้าชิง ประกอบด้วยการทำงานดังนี้

- สร้างกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล 2 ชิ้นข้อมูลจากชิ้นข้อมูลที่ผ่านค่าสนับสนุนขั้นต่ำดังสมการ (5.4)

$$
\begin{aligned}
c_2 &= \binom{d_1}{2} \\
&= \frac{d_1!}{2!(d_1-2)!} \\
&= \frac{d_1(d_1-1)(d_1-2)!}{2(d_1-2)!} \\
&= \frac{d_1(d_1-1)}{2} \\
&= \frac{d_1^{\,2}-d_1}{2}
\end{aligned} \tag{5.7}
$$

จากสมการ (5.7) จะเป็นเพียงค่า Big-O ในการสร้างกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล 2 ชิ้นข้อมูลเท่านั้นซึ่งมีค่าเป็น $O(d_1^2)$ และซึ่งหากสร้างกลุ่มข้อมูลท้าชิงทั้งหมดที่มีขนาดของชิ้นข้อมูลตั้งแต่ 2 ชิ้นข้อมูลจนถึง a ชิ้นข้อมูลจะสามารถแสดงดังสมการ (5.8) สามารถสรุปได้ว่าเวลาที่ใช้ในการสร้างกลุ่มข้อมูลท้าชิงมีค่า Big-O เป็น $O(ad_{a-1}^a)$

$$
\begin{aligned}
\sum_{j=2}^{a} c_j &= \sum_{j=2}^{a} \binom{d_{j-1}}{j} \\
&= \sum_{j=2}^{a} \left(\frac{(d_{j-1})!}{j!(d_{j-1}-1)!} \right) \\
&= \sum_{j=2}^{a} \left(\frac{d_{j-1}(d_{j-1}-1)(d_{j-1}-2)\cdots(d_{j-1}-j+1)(d_{j-1}-j)!}{j!(d_{j-1}-j)!} \right) \\
&= \sum_{j=2}^{a} \left(\frac{d_{j-1}(d_{j-1}-1)(d_{j-1}-2)\cdots(d_{j-1}-j+1)}{j!} \right)
\end{aligned} \tag{5.8}
$$

2) การทำงานในการตรวจสอบกลุ่มข้อมูลท้าชิงว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ ประกอบด้วยการทำงานดังนี้

- อ่านรายการข้อมูลจากฐานข้อมูลเพื่อค้นหาค่าความถี่ของกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเป็น n
- ตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเพื่อตัดกลุ่มข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเป็น d_{a-1}^a

จากการดำเนินงานทั้งหมดในส่วนนี้ เมื่อตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นตั้งแต่กลุ่มข้อมูลท้าชิงที่มีขนาด 2 ชิ้นข้อมูลถึง a ชิ้นข้อมูลนั้น สามารถแสดงดังสมการ (5.9) ทำให้ในขั้นตอนของการตรวจสอบกลุ่มข้อมูลท้าชิงมีค่า Big-O เป็น $O(ad_{a-1}^a)$

$$\text{Big-O} = O\left(\sum_{j=2}^{a}(n + d_{j-1}^{j})\right)$$
$$= O\left(an + \sum_{j=2}^{a} d_{j-1}^{j}\right)$$
$$= O(an + ad_{a-1}^{a})$$
$$= O(ad_{a-1}^{a}) \quad (5.9)$$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.10)

$$\text{Big-O} = O(ad_{a-1}^{a} + ad_{a-1}^{a})$$
$$= O(ad_{a-1}^{a}) \quad (5.10)$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีเฉลี่ยของขั้นตอนวิธี Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$ ■

<u>ทฤษฎีบท 5.6</u> ประสิทธิภาพการทำงานสำหรับ*กรณีที่แย่ที่สุด*ของขั้นตอนวิธี Apriori มีค่า Big-Oเป็น $O(md_{m-1}^{m})$

<u>พิสูจน์</u>

กรณีที่แย่ที่สุดสำหรับการทำงานของขั้นตอนวิธี Apriori จะเกิดขึ้นเมื่อทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย และทุกรายการข้อมูลในฐานข้อมูลเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด โดยการทำงานทั้งหมดสามารถแสดงได้ดังนี้

1) การทำงานในการสร้างกลุ่มข้อมูลท้าชิง ประกอบด้วยการทำงานดังนี้

การสร้างกลุ่มข้อมูลท้าชิงทั้งหมดที่มีขนาดของชิ้นข้อมูลตั้งแต่ 2 ชิ้นข้อมูลจนถึง m ชิ้นข้อมูลมีลักษณะการทำงานเช่นเดียวกับสมการ (5.8) ทำให้สามารถสรุปได้ว่าเวลาที่ใช้ในการสร้างกลุ่มข้อมูลท้าชิงมีค่า Big-O เป็น $O(md_{m-1}^{m})$

2) การทำงานในการตรวจสอบกลุ่มข้อมูลท้าชิงว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ ประกอบด้วยการทำงานดังนี้

- อ่านชิ้นข้อมูลจากฐานข้อมูลเพื่อค้นหาค่าความถี่ของกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเป็น m
- ตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเพื่อตัดกลุ่มข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเป็น c_{m-1}^{m}

จากการดำเนินงานทั้งหมดในส่วนนี้ เมื่อตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นตั้งแต่กลุ่มข้อมูลท้าชิงที่มีขนาด 1 ชิ้นข้อมูลถึง m ชิ้นข้อมูลนั้น สามารถแสดงดังสมการ (5.11) ทำให้ในขั้นตอนของการตรวจสอบกลุ่มข้อมูลท้าชิงมีค่า Big-O เป็น $O(m)$

$$\text{Big-O} = O\left(\sum_{j=2}^{m}(m + d^{j}_{j-1})\right)$$
$$= O\left(m^2 + \sum_{j=2}^{m} d^{j}_{j-1}\right)$$
$$= O(m^2 + md^{m}_{m-1})$$
$$= O(md^{m}_{m-1}) \quad (5.11)$$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.12)

$$\text{Big-O} = O(md^{m}_{m-1} + md^{m}_{m-1})$$
$$= O(md^{m}_{m-1}) \quad (5.12)$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีที่แย่ที่สุดของขั้นตอนวิธี Apriori มีค่า Big-O เป็น $O(md^{m}_{m-1})$ ■

5.2.3 ค่า Big-O ของขั้นตอนวิธี Modified-Apriori

การทำงานของขั้นตอนวิธี Modified-Apriori จะมีลักษณะการทำงานที่เหมือนกับขั้นตอนวิธี AMFIST ในส่วนของการสร้างตาราง ItemTable และตาราง BitTable แต่จะยังคงใช้หลักการทำงานของขั้นตอนวิธี Apriori ในส่วนของการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ที่มีการสร้างกลุ่มข้อมูลท้าชิงและทดสอบกลุ่มข้อมูลท้าชิงนั้นว่าเป็นกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยหรือไม่ โดยขั้นตอนวิธี Modified-Apriori มีค่า Big-O ดังทฤษฎีบท 5.7 ถึง 5.9

ทฤษฎีบท 5.7 ประสิทธิภาพการทำงานสำหรับ*กรณีที่ดีที่สุด*ของขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(p)$

พิสูจน์

กรณีที่ดีที่สุดสำหรับการทำงานของขั้นตอนวิธี Modified-Apriori จะเกิดขึ้นเมื่อไม่มีชิ้นข้อมูลใดมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ ทำให้การทำงานจะเกิดขึ้นเพียงขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable โดยประกอบด้วยการทำงานดังนี้

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูลลงในตาราง ItemTable เป็น m
- บันทึกทุกชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p
- ลบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำออกจากตาราง ItemTable และตาราง Temp-BitTable เป็น $2m$

และเนื่องจากทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันไม่บ่อยจึงไม่สามารถดำเนินการต่อได้ในขั้นตอนของการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้ ซึ่งค่า Big-O ในส่วนนี้สามารถแสดงดังสมการที่ (5.13)

$$\begin{aligned} \text{Big-O} &= O(m+p+2m) \\ &= O(p) \end{aligned} \tag{5.13}$$

ดังนั้นสามารถสรุปได้ว่า การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Modified-Apriori ในกรณีที่ดีที่สุดมีค่า Big-O เป็น $O(p)$ ■

<u>ทฤษฎีบท 5.8</u> ประสิทธิภาพการทำงานสำหรับ*กรณีเฉลี่ย*สำหรับขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$

<u>พิสูจน์</u>

กรณีเฉลี่ยสำหรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Modified-Apriori จะประกอบด้วยการทำงาน 2 ขั้นตอนดังนี้

1) การทำงานในการสร้างตาราง ItemTable และตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

(1.1) การสร้างตาราง ItemTable

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูลลงในตาราง ItemTable เป็น m
- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง ItemTable เป็น m
- เรียงลำดับชิ้นข้อมูลในตาราง ItemTable จากค่าความถี่มากไปหาค่าความถี่น้อยเป็น $a\log_2 a$

(1.2) การสร้างตาราง BitTable

- บันทึกทุกชิ้นข้อมูลจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p
- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง Temp-BitTable เป็น m
- เรียงรายการข้อมูลในตาราง Temp-BitTable เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $n\log_2 n$
- รวมรายการข้อมูลในตาราง Temp-BitTable ที่ซ้ำกันเป็น n
- บันทึกรายการข้อมูลจากตาราง Temp-BitTable หลังรวมรายการข้อมูลที่ซ้ำกันลงในตาราง BitTabel เป็น b

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable สามารถแสดงได้ดังสมการ (5.2) มีค่า Big-O เป็น $O(p)$

2) ขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ประกอบด้วยการทำงานดังนี้

- สร้างกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล 2 ชิ้นข้อมูลจากชิ้นข้อมูลที่ผ่านค่าสนับสนุนขั้นต่ำ ดังสมการ (5.7) ซึ่งการสร้างกลุ่มข้อมูลท้าชิงทั้งหมดที่มีขนาดของชิ้นข้อมูลตั้งแต่ 2 ชิ้นข้อมูลจนถึง m ชิ้นข้อมูลมีลักษณะการทำงานเช่นเดียวกับสมการ (5.8) ทำให้สามารถสรุปได้ว่าเวลาที่ใช้ในการสร้างกลุ่มข้อมูลท้าชิงนานมีค่า Big-O เป็น $O(ad_{a-1}^{a})$

- ตรวจสอบกลุ่มข้อมูลจากตาราง BitTable เพื่อนับค่าความถี่ของกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเป็น d_{a-1}^{a}

จากการดำเนินงานทั้งหมดในส่วนนี้ เมื่อต้องตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นตั้งแต่กลุ่มข้อมูลท้าชิงที่มีขนาด 2 ชิ้นข้อมูลถึง a ชิ้นข้อมูล สามารถแสดงดังสมการ (5.9) ทำให้ในขั้นตอนของการตรวจสอบกลุ่มข้อมูลท้าชิงนี้มีค่า Big-O เป็น $O(ad_{a-1}^{a})$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.10) ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีเฉลี่ยของขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$ ■

<u>ทฤษฎีบท 5.9</u> ประสิทธิภาพการทำงาน*กรณีที่แย่ที่สุด*สำหรับขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(mc_{m-1}^{m})$

<u>พิสูจน์</u>

กรณีที่แย่ที่สุดสำหรับการทำงานของขั้นตอนวิธี Modified-Apriori จะเกิดขึ้นเมื่อทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย และทุกรายการข้อมูลในฐานข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด โดยการทำงานทั้งหมดสามารถแสดงได้ดังนี้

1) การทำงานในการสร้างตาราง ItemTable และตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

(1.1) การสร้างตาราง ItemTable

- บันทึกชิ้นข้อมูลทั้งหมดที่ปรากฏในแต่ละรายการข้อมูลในฐานข้อมูลลงในตาราง ItemTable เป็น m

- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง Temp-BitTable เป็น m

- เรียงลำดับชิ้นข้อมูลในตาราง ItemTable จากค่าความถี่มากไปหาค่าความถี่น้อยเป็น $m\log_2 m$

(1.2) การสร้างตาราง BitTable

- บันทึกทุกชิ้นข้อมูลจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง Temp-BitTable เป็น p

- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำออกจากตาราง Temp-BitTable เป็น m

- เรียงรายการข้อมูลในตาราง Temp-BitTable เพื่อใช้ในการตรวจสอบรายการข้อมูลที่ซ้ำกันเป็น $n\log_2 n$

- ตรวจสอบรายการข้อมูลในตาราง Temp-BitTable ที่ซ้ำกันเป็น n

- บันทึกรายการข้อมูลจากตาราง Temp-BitTable หลังตรวจสอบรายการข้อมูลที่ซ้ำกันลงในตาราง BitTabel เป็น p

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง ItemTable และตาราง BitTable สามารถแสดงได้ดังสมการ (5.14) มีค่า Big-O เป็น $O(p)$

$$\begin{aligned}\text{Big-O} &= O(m+m+m\log_2 m+p+m+n\log_2 n+n+p)\\ &= O(3m+m\log_2 m+p+n\log_2 n+n+p)\\ &= O(p) \qquad (5.14)\end{aligned}$$

2) ขั้นตอนการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย ประกอบด้วยการทำงานดังนี้

- สร้างกลุ่มข้อมูลท้าชิงที่มีขนาดของชิ้นข้อมูล 2 ชิ้นข้อมูลจากชิ้นข้อมูลที่ผ่านค่าสนับสนุนขั้นต่ำ ดังสมการ (5.7) ซึ่งการสร้างกลุ่มข้อมูลท้าชิงทั้งหมดที่มีขนาดของชิ้นข้อมูลตั้งแต่ 2 ชิ้นข้อมูลจนถึง m ชิ้นข้อมูลมีลักษณะการทำงานเช่นเดียวกับสมการ (5.8) ทำให้สามารถสรุปได้ว่าเวลาที่ใช้ในการสร้างกลุ่มข้อมูลท้าชิงมีค่า Big-O เป็น $O(md^m_{m-1})$

- ตรวจสอบกลุ่มข้อมูลจากตาราง BitTable เพื่อนับค่าความถี่ของกลุ่มข้อมูลท้าชิงที่สร้างขึ้นเป็น d^m_{m-1}

จากการดำเนินงานทั้งหมดในส่วนนี้ เมื่อต้องตรวจสอบกลุ่มข้อมูลท้าชิงที่สร้างขึ้นตั้งแต่กลุ่มข้อมูลท้าชิงที่มีขนาด 2 ชิ้นข้อมูลถึง m ชิ้นข้อมูล สามารถแสดงดังสมการ (5.9) ทำให้ในขั้นตอนของการตรวจสอบกลุ่มข้อมูลท้าชิงนี้มีค่า Big-O เป็น $O(md^m_{m-1})$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.10) ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีเฉลี่ยของขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(md^m_{m-1})$ ■

5.2.4 ค่า Big-O ของขั้นตอนวิธี Index-BitTableFI

หลักการทำงานของขั้นตอนวิธี Index-BitTableFI โดยหลักๆ แล้วจะประกอบด้วยขั้นตอนการทำงานทั้งหมด 2 ขั้นตอนด้วยกัน ขั้นตอนแรกคือการสร้างตาราง BitTable พร้อมทั้งค้นหา Index Array ขั้นตอนที่สองคือการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากตาราง BitTable และ Index Array มีค่า Big-O ดังทฤษฎีบท 5.10 ถึง 5.12

ทฤษฎีบท 5.10 ประสิทธิภาพการทำงาน*กรณีที่ดีที่สุด*สำหรับขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(p)$

พิสูจน์

กรณีที่ดีที่สุดสำหรับการทำงานของขั้นตอนวิธี Index-BitTableFI จะเกิดขึ้นเมื่อไม่มีชิ้นข้อมูลใดมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ ทำให้การทำงานจะเกิดขึ้นเพียงส่วนหนึ่งของขั้นตอนการสร้างตาราง BitTable โดยประกอบด้วยการทำงาน ดังนี้

- บันทึกทุกชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง BitTable เป็น p
- ตรวจสอบและลบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง BitTable เป็น m

และเนื่องจากทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันไม่บ่อยจึงไม่สามารถดำเนินการต่อได้ในขั้นตอนของการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยได้ ซึ่งค่า Big-O ในส่วนนี้สามารถแสดงดังสมการที่ (5.15)

$$\begin{aligned} \text{Big-O} &= O(p+m) \\ &= O(p) \end{aligned} \tag{5.15}$$

ดังนั้นสามารถสรุปได้ว่า การค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Index-BitTableFI ในกรณีที่ดีที่สุดมีค่า Big-O เป็น $O(p)$ ■

ทฤษฎีบท 5.11 ประสิทธิภาพการทำงาน*กรณีเฉลี่ย*สำหรับขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(a2^{a-i+e_i})$

พิสูจน์

กรณีเฉลี่ยสำหรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Index-BitTableFI จะประกอบด้วยการทำงาน 2 ขั้นตอนดังนี้

1) การสร้างตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

- บันทึกทุกชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง BitTable เป็น p

- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง BitTable เป็น m

- เรียงลำดับชิ้นข้อมูลใหม่ตามค่าความถี่ของแต่ละชิ้นข้อมูลจากน้อยไปหามากพร้อมทั้งให้ค่าดัชนี เป็น $a\log_2 a + a$

- หาค่า Index Array ของแต่ละดัชนี เป็น an

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง BitTable สามารถแสดงได้ดังสมการ (5.16) มีค่า Big-O เป็น $O(an)$

$$\begin{aligned} \text{Big-O} &= O(p + m + a\log_2 a + a + an) \\ &= O(an) \end{aligned} \tag{5.16}$$

2) การสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก Index Array โดยสามารถแยกการทำงานออกเป็นกรณีได้ดังนี้

(2.1) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกเป็นเซตว่างและมีค่าสนับสนุนมากกว่าค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่อยู่หลังดัชนี i ในตาราง BitTable ซึ่งมีค่า Big-O เป็น $O(2^{a-i})$

(2.2) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกจำนวน e_i ตัวแต่มีค่าสนับสนุนเท่ากับค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก Subsume ของดัชนี i ซึ่งมีค่า Big-O เป็น $O(2^{e_i})$

(2.3) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกจำนวน e_i ตัวและมีค่าสนับสนุนมากกว่าค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก ดัชนีที่อยู่หลังดัชนี i ในตาราง BitTable และจาก Subsume ของดัชนี i ซึ่งมีค่า Big-O เป็น $O(2^{a-i+e_i})$

ซึ่งจากการทำงานในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของทั้ง 3 กรณี การทำงานที่ใช้เวลานานที่สุดคือกรณีที่ 3 ซึ่งหากต้องทำทุกดัชนีในตาราง BitTable ซึ่งมีทั้งหมด a ดัชนีจะทำให้มีค่า Big-O เป็น $O(a2^{a-i+e_i})$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.17)

$$\begin{aligned} \text{Big-O} &= O(an + a2^{a-i+e_i}) \\ &= O(a2^{a-i+e_i}) \end{aligned} \tag{5.17}$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีเฉลี่ยของขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(a2^{a-i+e_i})$ ■

<u>ทฤษฎีบท 5.12</u> ประสิทธิภาพการทำงาน*กรณีที่แย่ที่สุด*สำหรับขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(m2^{m-i+e_i})$

<u>พิสูจน์</u>

กรณีที่แย่ที่สุดสำหรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี Index-BitTableFI จะเกิดขึ้นเมื่อทุกชิ้นข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย และทุกรายการข้อมูลในฐานข้อมูลเป็นชิ้นข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด ประกอบด้วยการทำงาน 2 ขั้นตอนดังนี้

1) การสร้างตาราง BitTable ประกอบด้วยการทำงาน ดังนี้

- บันทึกทุกชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูลลงในตาราง BitTable เป็น p
- ตรวจสอบชิ้นข้อมูลที่ไม่ผ่านค่าสนับสนุนขั้นต่ำเพื่อตัดออกจากตาราง BitTable เป็น m
- เรียงลำดับชิ้นข้อมูลใหม่ตามค่าความถี่ของแต่ละชิ้นข้อมูลจากน้อยไปหามากพร้อมทั้งให้ค่าดัชนี เป็น $m\log_2 m+m$
- หาค่า Index Array ของแต่ละดัชนี เป็น mn

การดำเนินงานทั้งหมดในส่วนขั้นตอนของการสร้างตาราง BitTable สามารถแสดงได้ดังสมการ (5.18) มีค่า Big-O เป็น $O(mn)$

$$\begin{aligned}\text{Big-O} &= O(p+m+a\log_2 a+a+an)\\ &= O(an) \qquad (5.18)\end{aligned}$$

2) การสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก Index Array โดยสามารถแยกการทำงานออกเป็นกรณีได้ดังนี้

(2.4) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกเป็นเซตว่างและมีค่าสนับสนุนมากกว่าค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากดัชนีที่อยู่หลังดัชนี i ในตาราง BitTable ซึ่งมีค่า Big-O เป็น $O(2^{m-i})$

(2.5) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกจำนวน e_i ตัวแต่มีค่าสนับสนุนเท่ากับค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก Subsume ของดัชนี i ซึ่งมีค่า Big-O เป็น $O(2^{e_i})$

(2.6) กรณี Index Array ของดัชนี i มี Subsume ที่ประกอบด้วยสมาชิกจำนวน e_i ตัวและมีค่าสนับสนุนมากกว่าค่าสนับสนุนขั้นต่ำ จะสามารถสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจาก ดัชนีที่อยู่หลังดัชนี i ในตาราง BitTable และจาก Subsume ของดัชนี i ซึ่งมีค่า Big-O เป็น $O(2^{m-i+e_i})$

ซึ่งจากการทำงานในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของทั้ง 3 กรณี การทำงานที่ใช้เวลานานที่สุดคือกรณีที่ 3 ซึ่งหากต้องทำทุกดัชนีในตาราง BitTable ซึ่งมีทั้งหมด m ดัชนี จะทำให้มีค่า Big-O เป็น $O(m2^{m-i+e_i})$

และจากการทำงานข้างต้นของทั้งสองขั้นตอนสามารถแสดงค่า Big-O ได้ดังสมการที่ (5.19)

$$\begin{aligned} \text{Big-O} &= O(mn + m2^{m-i+e_i}) \\ &= O(m2^{m-i+e_i}) \end{aligned} \tag{5.19}$$

ดังนั้นสามารถสรุปได้ว่าการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับกรณีที่แย่ที่สุดของขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(m2^{m-i+e_i})$ ■

5.2.5 สรุปการวิเคราะห์ค่า Big-O ของแต่ละขั้นตอนวิธี

จากการวิเคราะห์ค่า Big-O ของแต่ละขั้นตอนวิธีที่ได้นำเสนอข้างต้น สามารถอธิบายถึงประสิทธิภาพการทำงานของการค้นหากลุ่มข้อมูลที่ปรากฏบ่อยของแต่ละขั้นตอนวิธี (โดยในที่นี้ขอกล่าวถึงการทำงานในกรณีเฉลี่ย) ได้ดังนี้

1) ขั้นตอนวิธี AMFIST มีค่า Big-O เป็น $O(a2^{r_i})$ เมื่อ r_i คือ จำนวนรายการข้อมูลที่ได้หลังการรวมรายการข้อมูลที่ซ้ำกันในตาราง BitTableIndex ของดัชนี i ซึ่งหากจำนวนของรายการข้อมูลของดัชนี i นั้นมีจำนวนที่ซ้ำกันมากๆ ก็จะทำให้ขั้นตอนวิธี AMFIST มีประสิทธิภาพมากขึ้น

2) ขั้นตอนวิธี Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$ เมื่อ a คือจำนวนชิ้นข้อมูล และ d_{a-1}^{a} คือ กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีขนาดของชิ้นข้อมูล $a+1$ ชิ้นข้อมูล ซึ่งจะเห็นได้ว่าหากจำนวนของ a มีมากแล้วจะทำให้ในรอบถัดไปของการสร้างกลุ่มข้อมูลท้าชิงที่ประกอบด้วย $a+1$ ชิ้นข้อมูล จะต้องสร้างกลุ่มข้อมูลท้าชิงและทดสอบกลุ่มข้อมูลท้าชิงเหล่านั้นเป็นจำนวนมาก

3) ขั้นตอนวิธี Modified-Apriori มีค่า Big-O เป็น $O(ad_{a-1}^{a})$ จะเห็นได้ว่ามีค่า Big-O เช่นเดียวกับขั้นตอนวิธี Apriori นั่นคือการทำงานที่ใช้เวลานานที่สุดคือขั้นตอนของการสร้างกลุ่มข้อมูลท้าชิง

4) ขั้นตอนวิธี Index-BitTableFI มีค่า Big-O เป็น $O(a2^{a-i+e_i})$ โดยการทำงานของขั้นตอนวิธีนี้จะใช้เวลามากในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย เนื่องจากต้องสร้างกลุ่มข้อมูลจากสมาชิกที่เป็น Subsume และจากตาราง BitTable ซึ่งต้องใช้เวลาในการตรวจสอบเป็นจำนวนมาก

จากการวิเคราะห์ค่า Big-O ของขั้นตอนวิธีทั้ง 4 พบว่า ขั้นตอนวิธี AMFIST มีค่า Big-O ที่น้อยที่สุดเมื่อเปรียบเทียบกับขั้นตอนวิธี Apriori ขั้นตอนวิธี Modified-Apriori และขั้นตอนวิธี Index-BitTableFI ซึ่งแสดงให้เห็นว่าขั้นตอนวิธี AMFIST มีประสิทธิภาพมากที่สุด

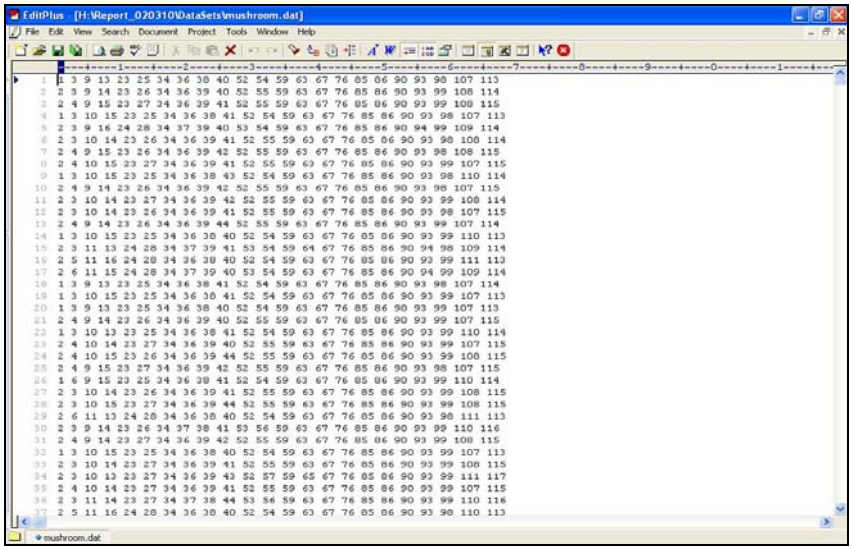
5.3 การวิเคราะห์ประสิทธิภาพการทำงานของขั้นตอนวิธีด้วยระยะเวลาการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

การวัดประสิทธิภาพการทำงานของขั้นตอนวิธีในหัวข้อนี้จะเป็นการทดสอบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยเริ่มจับเวลาตั้งแต่ขั้นตอนของการอ่านข้อมูลจากฐานข้อมูลจนถึงกระทั้งได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมดจากฐานข้อมูลดังกล่าว โดยในการทดสอบครั้งนี้แต่ละขั้นตอนวิธีคือ ขั้นตอนวิธี AMFIST ขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori ได้ทดสอบบนเครื่องคอมพิวเตอร์รุ่น Intel® Core™ 2 ที่มีหน่วยประมวลผลกลางความเร็ว 1.86 GHz หน่วยความจำ 1 GB ฮาร์ดดิสก์ความจุ 150 GB ระบบปฏิบัติการ Microsoft Windows XP และใช้ตัวแปลภาษาซี (C Compiler) ในการเขียนโปรแกรม ซึ่งขั้นตอนการทำงานแสดงดังภาคผนวก ก

5.3.1 ชุดข้อมูล (Datasets)

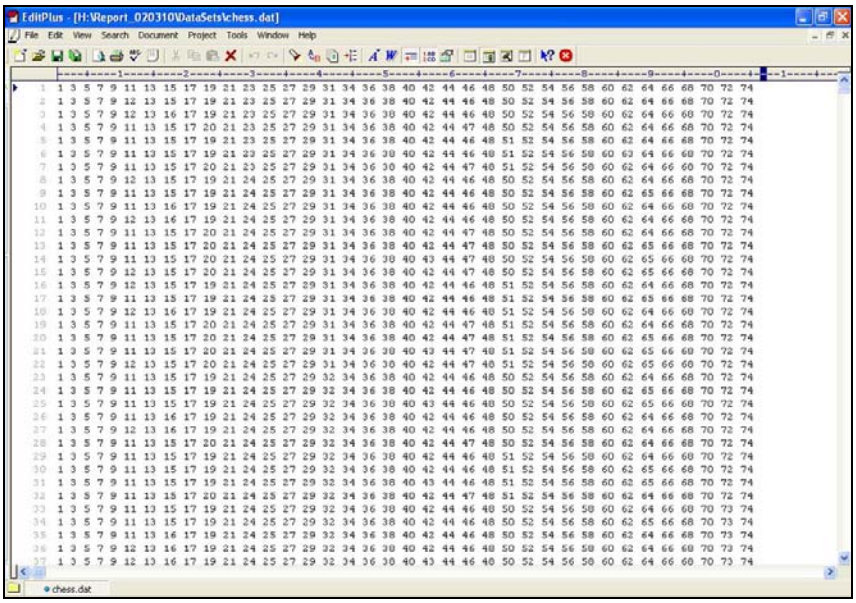
ชุดข้อมูลที่ใช้สำหรับการทดสอบระยะเวลาการทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori นั้นเป็นข้อมูลที่ได้จาก Frequent Itemset Mining Implementations Repository (Goethals and Zaki, 2003) ซึ่งลักษณะของข้อมูลในแต่ละรายการข้อมูลมีความคล้ายคลึงกัน ซึ่งเหมาะสมสำหรับการทดสอบในครั้งนี้ ประกอบด้วยชุดข้อมูล Mushroom และชุดข้อมูล Chess และผู้วิจัยได้เพิ่มการทดสอบกับชุดข้อมูลการวิเคราะห์น้ำ (Water Analysis) เป็นชุดข้อมูลของการใช้บริการในการวิเคราะห์สารเคมีที่อยู่ในน้ำของหน่วยเครื่องมือกลาง คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ ซึ่งมีลักษณะของรายการข้อมูลที่คล้ายคลึงกันเช่นกัน โดยแต่ละชุดข้อมูลมีรายละเอียดดังนี้

1) ชุดข้อมูล Mushroom (Mushroom Dataset) เป็นชุดข้อมูลที่ประกอบด้วยจำนวนชิ้นข้อมูลทั้งหมด 119 ชิ้นข้อมูลและมีรายการข้อมูลทั้งหมด 8,124 รายการข้อมูล ซึ่งแต่ละรายการข้อมูลจะมีชิ้นข้อมูลปรากฏอยู่จำนวน 23 ชิ้นข้อมูล จะเห็นได้ว่าอัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏในรายการข้อมูลต่อจำนวนชิ้นข้อมูลทั้งหมดในชุดข้อมูลนั้นมีค่าน้อย โดยจะเรียกชุดข้อมูลลักษณะนี้ว่า ชุดข้อมูลที่มีความหนาแน่นน้อย (Sparse Dataset) โดยลักษณะของชุดข้อมูล แสดงดังภาพประกอบ 5.1



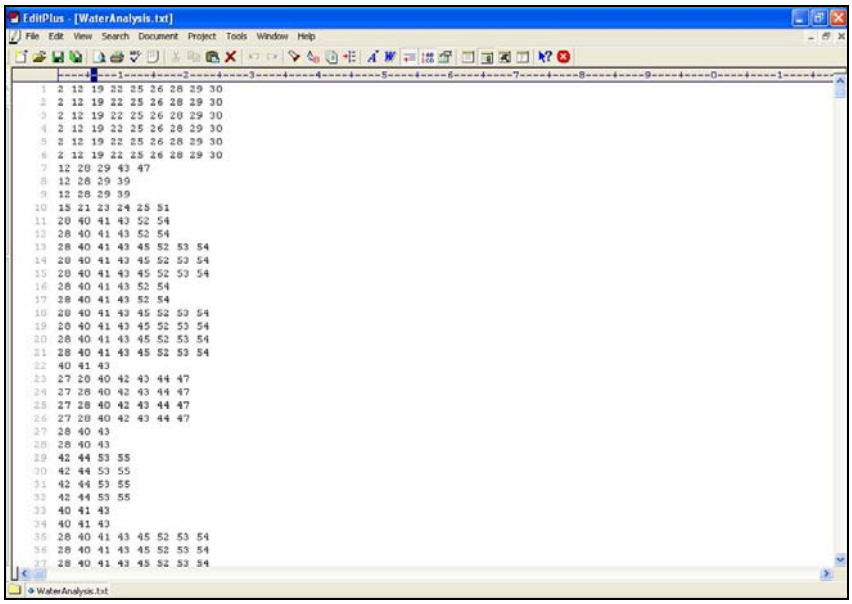
ภาพประกอบ 5.1 ชุดข้อมูล Mushroom

2) ชุดข้อมูล Chess (Chess Dataset) เป็นชุดข้อมูลที่ประกอบด้วยจำนวนชิ้นข้อมูลทั้งหมด 75 ชิ้นข้อมูลและมีรายการข้อมูลทั้งหมด 3,196 รายการข้อมูล ซึ่งแต่ละรายการข้อมูลจะมีชิ้นข้อมูลปรากฏอยู่จำนวน 37 ชิ้นข้อมูล จะเห็นได้ว่าอัตราส่วนของจำนวนชิ้นข้อมูลที่ปรากฏในรายการข้อมูลต่อจำนวนชิ้นข้อมูลทั้งหมดในชุดข้อมูลนั้นมีค่ามาก เราจะเรียกชุดข้อมูลลักษณะนี้ว่า ชุดข้อมูลที่มีความหนาแน่นมาก (Dense Dataset) โดยลักษณะของชุดข้อมูล แสดงดังภาพประกอบ 5.2



ภาพประกอบ 5.2 ชุดข้อมูล Chess

3) ชุดข้อมูล Water Analysis (Water Analysis Dataset) เป็นชุดข้อมูลจริงที่ได้จากการเก็บข้อมูลการใช้บริการในการวิเคราะห์สารเคมีที่อยู่ในน้ำของหน่วยเครื่องมือกลาง คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ ในช่วงระหว่างเดือนมกราคม พ.ศ.2548 ถึงเดือนกันยายน พ.ศ.2550 ประกอบด้วยจำนวนชิ้นข้อมูลทั้งหมด 60 ชิ้นข้อมูลและมีรายการข้อมูลทั้งหมด 1,649 รายการข้อมูล ซึ่งชิ้นข้อมูลในที่นี้จะหมายถึงสารเคมีที่อยู่ในน้ำที่ผู้ใช้บริการนำน้ำมาวิเคราะห์ซึ่งแทนอยู่ในรูปของตัวเลข โดยลักษณะของชุดข้อมูล Water Analysis แสดงดังภาพประกอบ 5.3



ภาพประกอบ 5.3 ชุดข้อมูล Water Analysis

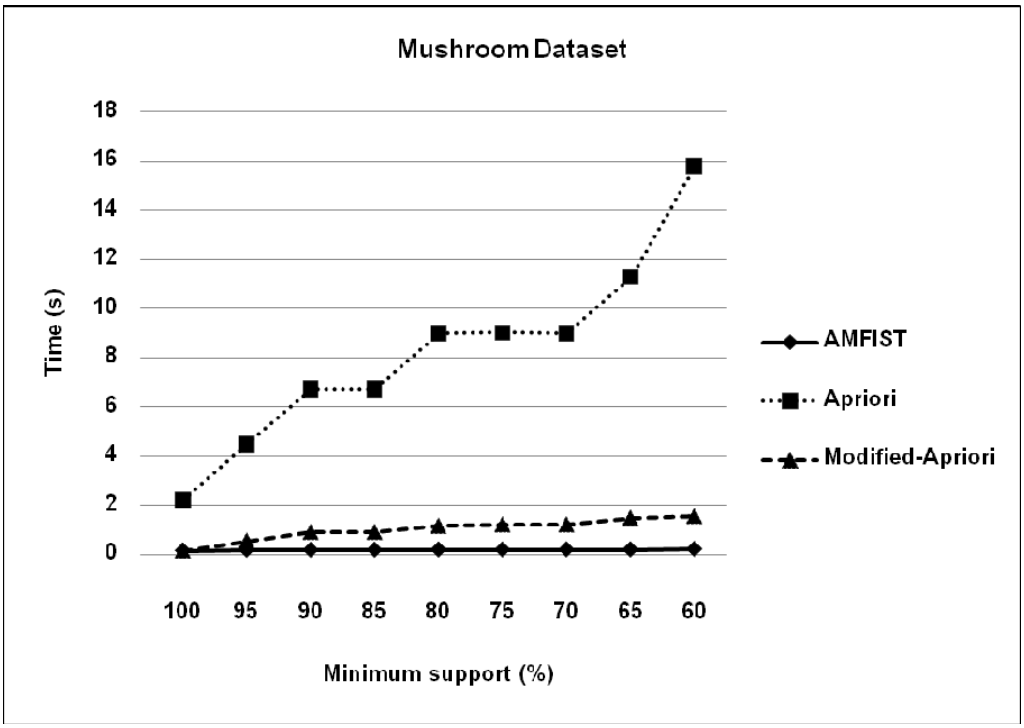
5.3.2 การเปรียบเทียบระยะเวลาการทำงานเฉลี่ยของแต่ละขั้นตอนวิธี

การทดสอบประสิทธิภาพการทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของแต่ละขั้นตอนวิธีในครั้งนี้จะแบ่งออกเป็น 3 กรณี โดยกรณีแรกเป็นการทดสอบประสิทธิภาพการทำงานกับชุดข้อมูลที่มีความหนาแน่นน้อย คือชุดข้อมูล Mushroom กรณีที่สองทดสอบประสิทธิภาพการทำงานกับชุดข้อมูลที่มีความหนาแน่นมาก คือชุดข้อมูล Chess และสุดท้ายทดสอบประสิทธิภาพการทำงานกับชุดข้อมูลจริงคือ ชุดข้อมูล Water Analysis โดยแต่ละกรณีสามารถแสดงได้ดังนี้

1) ผลการทดสอบระยะเวลาการทำงานเฉลี่ยสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเมื่อใช้ชุดข้อมูล Mushroom ของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori โดยระยะเวลาที่ใช้ในการทำงานแสดงดังตารางที่ 5.1 และสามารถแสดงการเปรียบเทียบระยะเวลาการทำงานของแต่ละขั้นตอนวิธีได้ดังภาพประกอบ 5.4

ตารางที่ 5.1 ระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Mushroom ของแต่ละขั้นตอนวิธี

ค่าสนับสนุนขั้นต่ำ (%)	ระยะเวลาที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย(s)		
	ขั้นตอนวิธี AMFIST	ขั้นตอนวิธี Apriori	ขั้นตอนวิธี Modified-Apriori
100	0.177	2.219	0.173
95	0.204	4.505	0.578
90	0.204	6.734	0.938
85	0.208	6.745	0.941
80	0.219	9.021	1.192
75	0.219	9.036	1.250
70	0.224	9.005	1.247
65	0.223	11.302	1.518
60	0.240	15.802	1.597

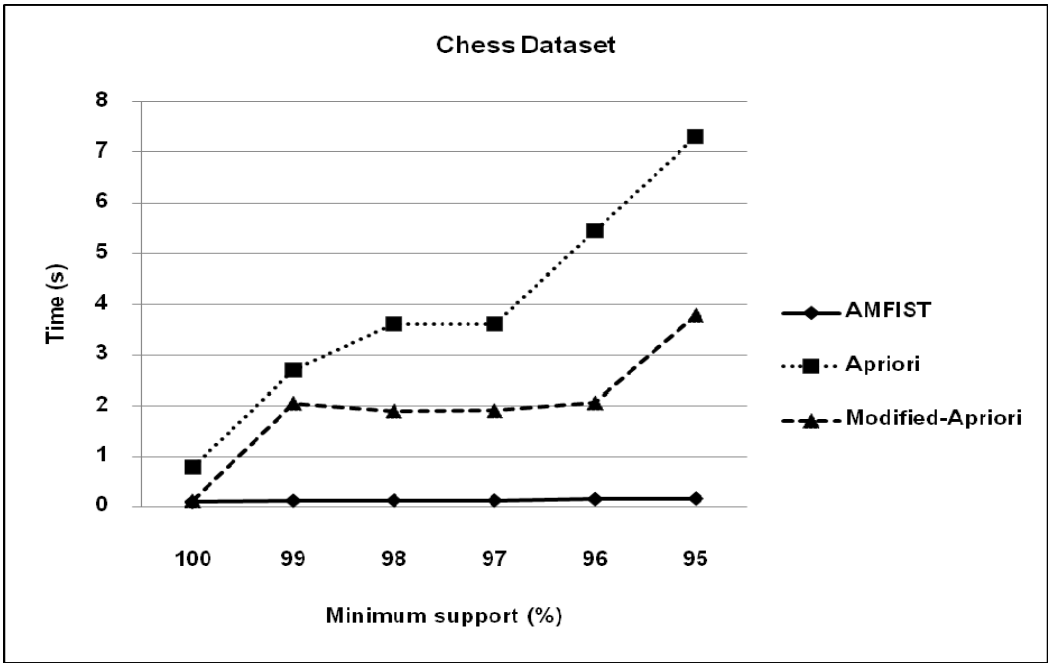


ภาพประกอบ 5.4 กราฟเปรียบเทียบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Mushroom ของแต่ละขั้นตอนวิธี

2) ผลการทดสอบระยะเวลาการทำงานเฉลี่ยในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเมื่อใช้ชุดข้อมูล Chess ของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori โดยระยะเวลาที่ใช้ในการทำงานแสดงดังตารางที่ 5.2 และสามารถแสดงการเปรียบเทียบระยะเวลาการทำงานของแต่ละขั้นตอนวิธีได้ดังภาพประกอบ 5.5

ตารางที่ 5.2 ระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Chess ของแต่ละขั้นตอนวิธี

ค่าสนับสนุนขั้นต่ำ (%)	ระยะเวลาที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย(s)		
	ขั้นตอนวิธี AMFIST	ขั้นตอนวิธี Apriori	ขั้นตอนวิธี Modified-Apriori
100	0.104	0.797	0.130
99	0.130	2.709	2.068
98	0.135	3.625	1.918
97	0.135	3.625	1.922
96	0.162	5.458	2.075
95	0.173	7.318	3.802

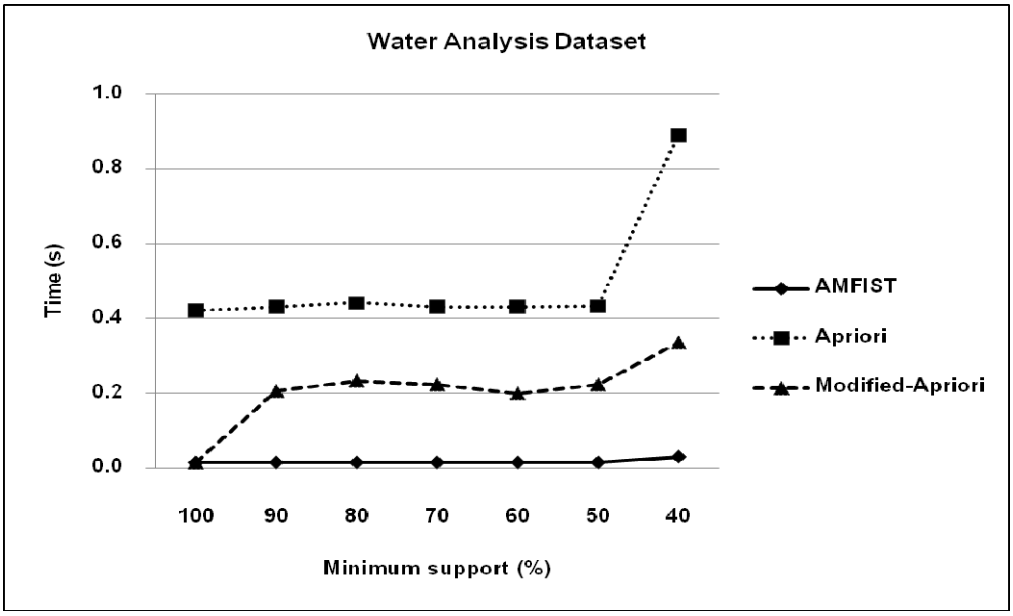


ภาพประกอบ 5.5 กราฟเปรียบเทียบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Chess ของแต่ละขั้นตอนวิธี

3) ผลการทดสอบระยะเวลาการทำงานเฉลี่ยในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเมื่อใช้ชุดข้อมูล Water Analysis ของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori โดยระยะเวลาที่ใช้ในการทำงานแสดงดังตารางที่ 5.3 และสามารถแสดงการเปรียบเทียบระยะเวลาการทำงานของแต่ละขั้นตอนวิธีได้ดังภาพประกอบที่ 5.6

ตารางที่ 5.3 ระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Water Analysis ของแต่ละขั้นตอนวิธี

ค่าสนับสนุนขั้นต่ำ (%)	**ระยะเวลาที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย(s)**		
	ขั้นตอนวิธี AMFIST	**ขั้นตอนวิธี Apriori**	**ขั้นตอนวิธี Modified-Apriori**
100	0.015	0.422	0.015
90	0.016	0.432	0.207
80	0.016	0.443	0.234
70	0.016	0.432	0.224
60	0.016	0.432	0.200
50	0.016	0.433	0.224
40	0.031	0.891	0.336



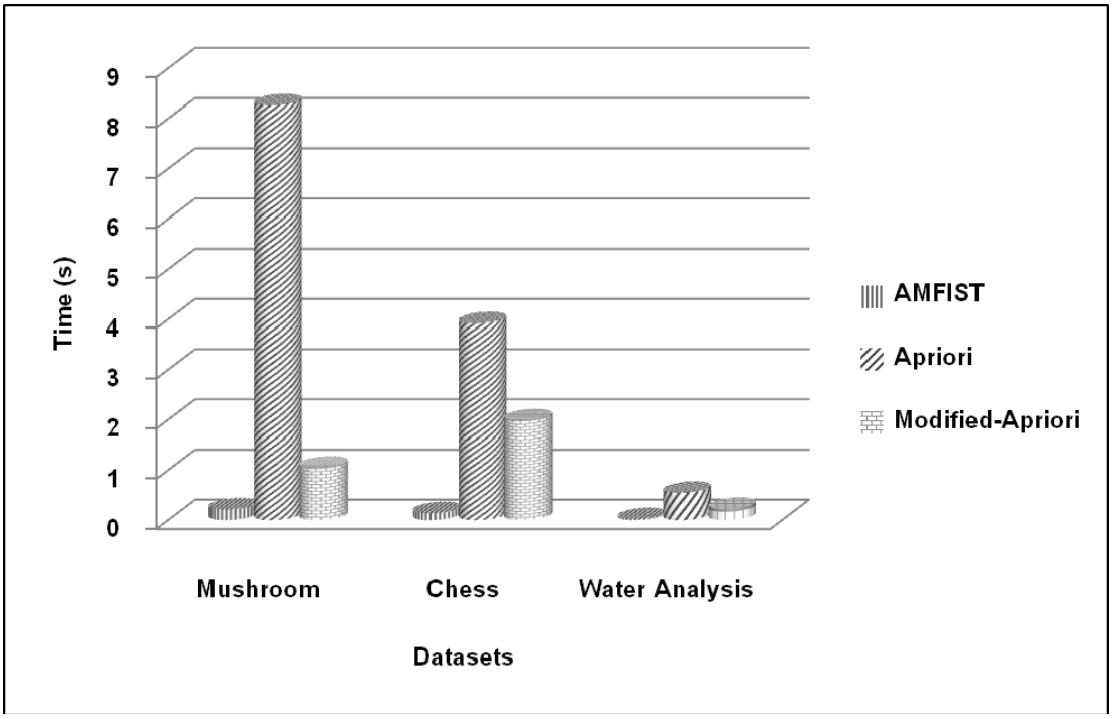
ภาพประกอบ 5.6 กราฟเปรียบเทียบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับชุดข้อมูล Water Analysis ของแต่ละขั้นตอนวิธี

5.3.3 การวิเคราะห์ระยะเวลาการทำงานเฉลี่ยของแต่ละขั้นตอนวิธี

จากผลการทดสอบประสิทธิภาพการทำงานในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของแต่ละขั้นตอนวิธี สามารถสรุประยะเวลาเฉลี่ยที่ใช้ในการทำงานสำหรับแต่ละชุดข้อมูลของแต่ละขั้นตอนวิธีได้ดังตารางที่ 5.4 ซึ่งจะเห็นได้ว่าขั้นตอนวิธี AMFIST ใช้เวลาในการทำงานน้อยที่สุด โดยใช้เวลาในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเฉลี่ยอยู่ที่ 0.213 วินาทีสำหรับชุดข้อมูล Mushroom 0.140 วินาทีสำหรับชุดข้อมูล Chess และ 0.020 วินาทีสำหรับชุดข้อมูล Water Analysis ขั้นตอนวิธี Modified-Apriori ใช้เวลาในการทำงานน้อยเป็นอันดับสองรองจากขั้นตอนวิธี AMFIST โดยใช้เวลาในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเฉลี่ยอยู่ที่ 1.048 วินาทีสำหรับชุดข้อมูล Mushroom 1.986 วินาทีสำหรับชุดข้อมูล Chess และ 0.225 วินาทีสำหรับชุดข้อมูล Water Analysis ซึ่งจะเห็นได้ว่าขั้นตอนวิธี Modified-Apriori ใช้เวลามากกว่าขั้นตอนวิธี AMFIST เนื่องจากขั้นตอนวิธี Modified-Apriori ต้องใช้เวลาในการสร้างกลุ่มข้อมูลท้าชิงแต่ขั้นตอนวิธี AMFIST จะไม่มีการสร้างกลุ่มข้อมูลท้าชิงจึงลดเวลาในส่วนนี้ และสุดท้ายขั้นตอนวิธี Apriori ใช้เวลาในการทำงานมากที่สุด โดยใช้เวลาในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเฉลี่ยอยู่ที่ 8.263 วินาทีสำหรับชุดข้อมูล Mushroom 3.922 วินาทีสำหรับชุดข้อมูล Chess และ 0.546 วินาทีสำหรับชุดข้อมูล Water Analysis เนื่องจากขั้นตอนวิธี Apriori ต้องสร้างกลุ่มข้อมูลท้าชิงและทดสอบกลุ่มข้อมูลท้าชิงเหล่านั้นด้วยการอ่านข้อมูลจากชุดข้อมูลหลายรอบทำให้ใช้เวลาในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนานที่สุดเมื่อเทียบกับขั้นตอนวิธี AMFIST และขั้นตอนวิธี Modified-Apriori และเมื่อนำระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับแต่ละชุดข้อมูลของแต่ละขั้นตอนวิธีมาเขียนกราฟแท่งเพื่อเปรียบเทียบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสามารถแสดงดังภาพประกอบ 5.7

ตารางที่ 5.4 ระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับแต่ละชุดข้อมูลของแต่ละขั้นตอนวิธี

ชุดข้อมูล (Datasets)	**ระยะเวลาที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย(s)**		
	ขั้นตอนวิธี AMFIST	**ขั้นตอนวิธี Apriori**	**ขั้นตอนวิธี Modified-Apriori**
Mushroom	0.213	8.263	1.048
Chess	0.140	3.922	1.986
Water Analysis	0.020	0.546	0.225



ภาพประกอบ 5.7 กราฟเปรียบเทียบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับแต่ละชุดข้อมูลของแต่ละขั้นตอนวิธี

ซึ่งจากการเปรียบเทียบระยะเวลาการทำงานของขั้นตอนวิธี AMFIST กับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori จะเห็นได้ว่าการทำงานของขั้นตอนวิธี Apriori นั้นใช้เวลามากที่สุดอันเนื่องจาก ขั้นตอนวิธี Apriori ต้องอ่านข้อมูลจากฐานข้อมูลหลายครั้งทำให้ต้องมีการติดต่อกับอุปกรณ์ I/O (Input/Output Device) หลายรอบซึ่งทำให้ประสิทธิภาพการทำงานน้อยลงเมื่อเทียบกับขั้นตอนวิธี AMFIST และขั้นตอนวิธี Modified-Apriori

บทที่ 6

บทสรุปและข้อเสนอแนะ

วิทยานิพนธ์นี้ได้สร้างขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยโดยรองรับรายการข้อมูลที่คล้ายคลึงกัน เรียกว่า ขั้นตอนวิธี AMFIST (AMFIST Algorithm) โดยลักษณะการทำงานจะไม่มีการสร้างกลุ่มข้อมูลท้าชิง อ่านข้อมูลจากฐานข้อมูลเพื่อใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยเพียงครั้งเดียว ใช้การจัดเก็บข้อมูลในรูปแบบบิต และดำเนินการระดับบิตในการค้นหาค่าความถี่และสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย นอกจากนี้ใช้วิธีการรวมรายการข้อมูลที่ซ้ำกันหลังจากตัดชิ้นข้อมูลที่ไม่ปรากฏร่วมกันบ่อยเพื่อให้สามารถทำงานได้อย่างรวดเร็วและลดเนื้อที่ในการจัดเก็บข้อมูลในหน่วยความจำระหว่างการประมวลผล

การทำงานของขั้นตอนวิธี AMFIST โดยหลักๆ ประกอบด้วยขั้นตอนการทำงานทั้งหมด 2 ขั้นตอนด้วยกัน ขั้นตอนแรกคือการสร้างตาราง ItemTable และตาราง BitTable เพื่อใช้ในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย โดยในขั้นตอนนี้จะอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียวเพื่อลดระยะเวลาในการอ่านข้อมูลจากฐานข้อมูล และมีการรวมรายการข้อมูลที่ซ้ำกันในตาราง BitTable เพื่อลดจำนวนของรายการข้อมูลทำให้สามารถตรวจสอบค่าความถี่ของกลุ่มข้อมูลได้เร็วขึ้นอีกทั้งยังลดเนื้อที่ในการจัดเก็บข้อมูลในระหว่างการประมวลผล และขั้นตอนที่สองคือการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากตาราง BitTable และตาราง ItemTable ที่ได้ในขั้นตอนแรก โดยในขั้นตอนนี้จะใช้การดำเนินการระดับบิต AND และการดำเนินการระดับบิต OR ในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมด

6.1 สรุปผลการวิจัย

ขั้นตอนวิธี AMFIST เป็นขั้นตอนวิธีที่มีประสิทธิภาพในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยสำหรับรายการข้อมูลที่มีความคล้ายคลึงกันในฐานข้อมูล ซึ่งจากการวิเคราะห์และผลการทดลองจะเห็นได้ว่าขั้นตอนวิธี AMFIST มีประสิทธิภาพการทำงานที่ดีกว่าขั้นตอนวิธี Apriori ขั้นตอนวิธี Modified-Apriori และขั้นตอน Index-BitTable โดยสามารถอธิบายรายละเอียดได้ดังนี้

6.1.1 ผลการเปรียบเทียบค่า Big-O ของขั้นตอนวิธี AMFIST

จากการวิเคราะห์ค่า Big-O ของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธีอื่น คือ ขั้นตอนวิธี Apriori ขั้นตอนวิธี Modified-Apriori และขั้นตอนวิธี Index-BitTableFI ปรากฏว่าในกรณีที่ดีที่สุดแต่ละขั้นตอนวิธีใช้เวลาในการทำงานที่ไม่แตกต่างกัน คือมีค่า Big-O เป็น $O(p)$ เมื่อ p คือจำนวนของชิ้นข้อมูลทั้งหมดจากทุกรายการข้อมูลในฐานข้อมูล แต่สำหรับในกรณีเฉลี่ยและกรณีที่แย่ที่สุดนั้นจะเห็นได้ว่าขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori ใช้เวลานานที่สุดอีกทั้งยังใช้เวลาเท่ากันนั้นคือ $O(ad_{a-1}^{a})$ และ $O(md_{a-1}^{a})$ ตามลำดับ ซึ่งการทำงานจะใช้เวลานานในการสร้างกลุ่มข้อมูลท้าชิงเพราะเนื่องจากต้องสร้างกลุ่มข้อมูลเป็นจำนวนมากทำให้ใช้เวลานานกว่าขั้นตอนวิธีอื่นที่เหลือ และขั้นตอนวิธีที่ทำงานนานรองลงมาคือขั้นตอนวิธี Index-BitTableFI และสุดท้ายขั้นตอนวิธีที่ใช้เวลาการทำงานน้อยที่สุดในการเปรียบเทียบค่า Big-O นี้คือขั้นตอนวิธี AMFIST ซึ่งค่าที่ได้จากการวิเคราะห์ค่า Big-O ทั้งหมดสามารถสรุปได้ดังแสดงในตารางที่ 6.1

ตารางที่ 6.1 ประสิทธิภาพการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของแต่ละขั้นตอนวิธี

Big-O	**ขั้นตอนวิธี AMFIST**	**ขั้นตอนวิธี Apriori**	**ขั้นตอนวิธี Modified-Apriori**	**ขั้นตอนวิธี Index-BitTable**
กรณีดีที่สุด	$O(p)$	$O(p)$	$O(p)$	$O(p)$
กรณีเฉลี่ย	$O(a2^{r_i})$	$O(ad_{a-1}^{a})$	$O(ad_{a-1}^{a})$	$O(a2^{a-i+e_i})$
กรณีแย่ที่สุด	$O(m2^{r_i})$	$O(md_{a-1}^{a})$	$O(md_{a-1}^{a})$	$O(m2^{m-i+e_i})$

6.1.2 ผลการทดสอบเวลาการทำงานของขั้นตอนวิธี AMFIST

หลังจากการทดสอบระยะเวลาเฉลี่ยที่ใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยของขั้นตอนวิธี AMFIST เปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori โดยเริ่มจับเวลาการทำงานตั้งแต่ขั้นตอนของการอ่านข้อมูลจากฐานข้อมูลจนถึงกระทั่งได้กลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยทั้งหมดจากฐานข้อมูลดังกล่าว ผลการทดสอบปรากฏว่าขั้นตอนวิธี AMFIST สามารถทำงานได้ดีทั้งกับฐานข้อมูลที่มีความหนาแน่นมากและฐานข้อมูลที่มีความหนาแน่นน้อย โดยใช้เวลาในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยน้อยกว่าขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori

6.2 ปัญหาและอุปสรรค

เนื่องจากการสร้างขั้นตอนวิธีเพื่อค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยในวิทยานิพนธ์นี้เป็นการสร้างขั้นตอนวิธีสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่รองรับรายการข้อมูลที่คล้ายคลึงกัน ทำให้การทดสอบประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST ต้องใช้ชุดข้อมูลที่มีลักษณะของรายการข้อมูลที่คล้ายคลึงกันปรากฏเป็นจำนวนมากในฐานข้อมูล ซึ่งชุดข้อมูลโดยทั่วไปที่ใช้ในการทดสอบขั้นตอนวิธีสำหรับค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นไม่สามารถใช้ได้ทั้งหมด เนื่องจากขั้นตอนวิธี AMFIST นั้นเป็นขั้นตอนวิธีใหม่ที่มีการสร้างขึ้นเพื่อรองรับฐานข้อมูลที่รายการข้อมูลที่คล้ายคลึงกัน ทำให้มีชุดข้อมูลที่รองรับการทดสอบที่สอดคล้องกับลักษณะของขั้นตอนวิธี AMFIST น้อย

6.3 ข้อเสนอแนะและงานในอนาคต

ขั้นตอนวิธี AMFIST ถูกออกแบบมาเพื่อรองรับการทำงานสำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่มีรายการข้อมูลคล้ายคลึงกันจำนวนมากในฐานข้อมูล ดังนั้นหากฐานข้อมูลที่นำมาใช้สำหรับการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยนั้นมีจำนวนของรายการข้อมูลที่คล้ายคลึงกันมีจำนวนน้อยก็จะไม่เหมาะสมที่จะใช้ขั้นตอนวิธี AMFIST และสืบเนื่องจากขั้นตอนวิธี AMFIST ในวิทยานิพนธ์นี้ถูกคิดค้นขึ้นมาจากแนวความคิดที่จะใช้ในการค้นหากลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยกับฐานข้อมูลการใช้บริการวิเคราะห์สารเคมีของผู้ใช้บริการหน่วยเครื่องมือกลาง คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ เพื่อนำกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยที่ได้ไปสร้างโปรโมชั่น (Promotions) แก่ผู้ใช้บริการ ซึ่งในอนาคตอาจนำขั้นตอนวิธี AMFIST ไปประยุกต์ใช้งานจริง

บรรณานุกรม

C. Berberidis, G. Tzanis, and L. Vlahavas. “Mining for Contiguous Frequent Itemsets in Transaction databases”. In Proceedings of IEEE Workshop on Intelligent Data Acquisition and Advanced Computing systems: Technology and Applications, Sofia, Bulgaria, pp. 679-685, 2005.

B. Goethals. 2004. Frequent Itemset Mining Dataset Repository. http://fimi.cs.helsinki.fi (accessed 09/03/2010).

C. Borgelt. 2010. Christian Borgelt's Webpages. http://www.borgelt.net//index.html (accessed 09/03/2010).

C. Lin, C. Hong, “Using customer knowledge in designing electronic catalog”, Expert Systems with Applications, vol. 34, pp. 119-127, 2008.

C. Meinel, and T. Theobald. “Algorithms and Data Structures in VLSI Design”. Published by Springer-Verlag, Berlin Heidelberg, New York, 1998.

C. Rygielski, J. Wang, and D. Yen. “Data mining techniques for customer relationship management”. Published in Technology in Society 24, pp. 483-502, 2002.

D. Olson and D. Delen. “Advance Data Mining Thechniques”. Published by Springer-Verlag, Berlin Heidelberg, New York, pp. 53-68, 2008.

D. Simovic, C. Djeraba. “Mathematical Tools for Data Mining: Set Theory, Partial Order, Combinatorics”. Published by Springer, London, England, 2008.

D. Taniar. “Research and Trends in Data Mining Technologies and Applications”. Published by Idea Group Publishing,United States of America, 2007, pp. 86-114.

H. Schildt. “C++ by Herbert Schildt”. Published by Osborne McGraw-Hill, United States of America, 1990.

I. Willen and E. Frank. “Data Mining: Practical Machine Tool and Techniques”. Published by Morgan Kaufmann series in data management systems, United States of America, pp. 61-82, 2005.

J. Dong, M. Han. “BitTableFI: an efficient mining frequent itemsets algorithm”, Knowledge Based Systems vol.20, no.4, pp. 329-335, 2007.

J. Han, J. Pei, and Y. Yin. “Mining Frequent Patterns without Candidate Generation”. In Proceedings of ACM SIGMOD Int’l Conference on Management of Data, Dallas, pp. 1-12, 2000.

J. Han, M. Kamber, and J. Pei. “Data Mining: Concepts and Techniques”. Published by Morgan Kaufmann, California, United States, 2006.

J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. “H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Database”. In Proceedings of the 2001 IEEE ICDM, San Jose, California, 2001.

K. Rosen. “Discrete Mathematics and Its Applications”. Published by McGraw-Hill Companies, United States of America, 1999.

M. Kantardzic. “Data Mining: Concepts, Models, Methods, and Algorithms”. Published by John Wiley & Sons,Inc., New York, 2002.

N. Pasquier, Y. Basetide, R. Taouil, and L. Lakhal. “Efficient Mining of Association Rules Using Closed Itemset Lattices”. In Information Systems vol.24, no.1, pp. 25-46, 1999.

R. Agrawal and R. Srikant. “Fast Algorithms for Mining Association Rules”. In Proceedings of the 20th Int’l Conference of Very Large Databases (VLDB’94) pp. 478-499, 1994.

R. Agrawal, T. Imielinski, and A. Sawami. “Mining Association Rules between Sets of Items in Large Databases”. In Procceding of ACM SIGMOD, pp. 207-216, 1993.

R. Gopalan and Y. Sucahyo. “ITL-MINE: Mining Frequent Itemsets more Efficiently”. In Proceedings of the 2002 Int’l Conference on Fuzzy Systems and Knowladge Discovery, Singapore, 2002.

R. Roiger, M. Geats. “Data Mining: A TUTORIRL-BASED PRIMER”. Published by Pearson Education,Inc., United States of America, 2003.

S. Liao, C. Hsieh, and S. Huang. “Mining product maps for new product development”, Expert Systems with Applications, vol. 34, pp. 50-62, 2008.

S. Liao, Y. Chen. “Mining customer knowledge for electronic catalog marketing”, Expert Systems with Applications, vol. 27, pp. 521-532, 2004.

S. Mitra and T. Acharya. “Data mining: Multimedia, Soft Computing, and Bioinformatics”. Published by John Wiley & Sons, Inc., Hoboken, New Jersey, United States of America, pp. 5-10, 2003.

T. Koshy. “Discrete Mathematics with Applications”. Published by Elsevier, New Delhi, India, 2005.

U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “From Data Mining to Knowlagde Discovery in Databases”. AI Magazine, pp. 37-54, 1996.

W. Song, B. Yang, Z. Xu. “Index-BitTableFI: An improved algorithm for mining frequent itemsets”, Knowledge Based Systems, pp. 507-513, 2008.

Y.G. Sucahyo, R. Gopalan. “CT-ITL: Efficient Frequent Item Set Mining Using a Compressed Prefix Tree with Pattern Growth”, In Proceedings of 14th Australasian Database Conference, Adelaide, Australia, 2003.

Y.G. Sucahyo, R. Gopalan. “CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure”. In Proceedings of IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), Brighton UK, November 2004.

ภาคผนวก

ภาคผนวก ก

การเขียนโปรแกรมเพื่อวัดประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST

ก.1 ฟังก์ชันการทำงานของขั้นตอนวิธี AMFIST

ในการเขียนโปรแกรมวัดประสิทธิภาพการทำงานของขั้นตอนวิธี AMFIST เพื่อเปรียบเทียบกับขั้นตอนวิธี Apriori และขั้นตอนวิธี Modified-Apriori นั้น ใช้การเขียนโปรแกรมด้วยตัวแปลภาษาซี (C Compiler) โดยมีฟังก์ชัน (Function) การทำงานที่สำคัญ ดังต่อไปนี้

1) InsertInBitTable(*t*,*i*,Item) คือ ฟังก์ชันการบันทึกชิ้นข้อมูลที่อ่านจากฐานข้อมูลลงในตาราง Temp-BitTable ที่รายการข้อมูลที่ *t* และตำแหน่งบิตที่ *i*

2) CreateItemTable(*t*,*i*,Temp-BitTable) คือ ฟังก์ชันการสร้างตาราง ItemTable โดยการอ่านข้อมูลจากตาราง Temp-BitTable โดยเริ่มบันทึกค่าความถี่ของชิ้นข้อมูลตั้งแต่ชิ้นข้อมูลแรกของรายการข้อมูลแรก จนถึงชิ้นข้อมูลที่ *i* ของรายการข้อมูลที่ *t*

3) DeleteInFrequent(ItemTable,Temp-BitTable,min_sup) คือ ฟังก์ชันการลบชิ้นข้อมูลที่มีค่าสนับสนุนต่ำกว่าค่าสนับสนุนขั้นต่ำออกจากตาราง ItemTable และตาราง Temp-BitTable

4) DefiendIndex(ItemTable) คือ ฟังก์ชันการกำหนดค่าดัชนีให้แก่แต่ละชิ้นข้อมูลในตาราง ItemTable

5) RearrangeBitTable(ItemTable,Temp-BitTable, BitTable) คือ ฟังก์ชันการจัดลำดับของตำแหน่งบิตชิ้นข้อมูลใหม่ในตาราง Temp-BitTable โดยเรียงตามลำดับของค่าดัชนีของชิ้นข้อมูลในตาราง ItemTable แล้วบันทึกลงตาราง BitTable

6) PrintFrequentItems(ItemTable) คือ ฟังก์ชันการแสดงชิ้นข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent 1-Itemsets) ลงไฟล์ข้อมูล

7) CreateBitTableIndex(j,BitTable) คือ ฟังก์ชันการสร้างตาราง BitTableIndex ของดัชนีที่ *j* จากตาราง BitTable เพื่อใช้ในการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย

8) AgregateSimilarTransactions(*j*,BitTableIndex) คือ ฟังก์ชันการรวมรายการข้อมูลในตาราง BitTableIndex ที่มีบิตแรกจนถึงบิตที่ *j* เหมือนกันทุกบิต โดยใช้การดำเนินการระดับบิต XOR

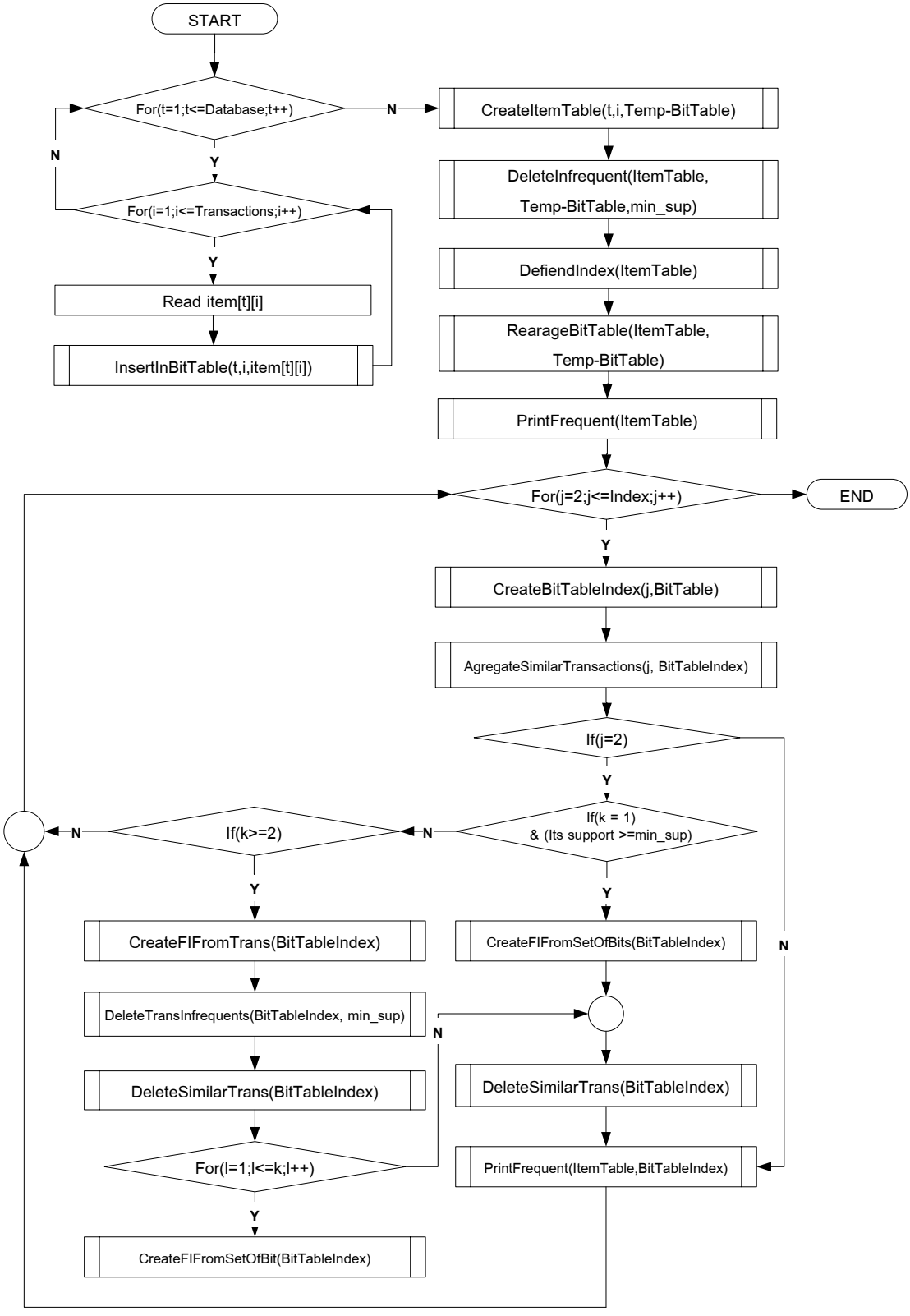
9) CreateFIFromTrans(BitTableIndex) คือ ฟังก์ชันการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากรายการข้อมูลในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต AND

10) CreateFIFromSetOfBits(BitTableIndex) คือ ฟังก์ชันการสร้างกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อยจากเซตของบิตของแต่ละรายการข้อมูลในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต OR

11) DeleteSimilarTrans(BitTableIndex) คือ ฟังก์ชันการลบรายการข้อมูลที่ซ้ำกันในตาราง BitTableIndex โดยใช้การดำเนินการระดับบิต XOR

12) PrintFrequentItemsets(ItemTable, BitTableIndex) คือ ฟังก์ชันการแสดงกลุ่มข้อมูลที่ปรากฏร่วมกันบ่อย (Frequent Itemsets) ลงไฟล์ข้อมูล

ก.2 ลำดับการทำงานของขั้นตอนวิธี AMFIST



ภาคผนวก ข

ผลงานตีพิมพ์

เรื่อง	FIAST: A Novel Algorithm for Mining Frequent Itemsets
งานประชุมวิชาการ	2009 International Conference on Future Computer and Communication (ICFCC 2009)
สถานที่	กัวลาลัมเปอร์ ประเทศมาเลเซีย
วันที่	3 – 5 เมษายน 2552

FIAST: A Novel Algorithm for Mining Frequent Itemsets

Fudailah Duemong, Ladda Preechaveerakul and Sirirut Vanichayobon
Computer Science Department
Faculty of Science, Prince of Songkla University
Hat Yai, Songkhla, Thailand
e-mail: {s5010220092, ladda.p, sirirut.v}@psu.ac.th

***Abstract*—An efficient algorithm to mine frequent itemsets is crucial for mining association rules. Most of the previously used algorithms have generally been developed for using the computational time effectively, reducing the number of candidate itemsets and decreasing the number of scan in the database. However, the time can be reduced by aggregate transactions having similar itemsets. This paper, then proposes an efficient algorithm for mining frequent itemsets without generating candidate itemsets called FIAST (Frequent Itemsets Algorithm for Similar Transactions). The algorithm uses a divide-and-conquer method to reduce the task into the bitwise AND operation for finding itemsets and uses a depth-first search strategy to generate all frequent itemsets. The time complexity is analyzed by mathematical proof.**

Keywords-association rules; frequent itemsets; mining

I. INTRODUCTION

Data mining [1] is a tool of specific algorithms for extracting patterns from data, and is actually a part of a larger process called Knowledge Discovery in Databases (KDD) [2] which describes the step taken to ensure meaningful result.

Association rules mining [3] is one of the most important and well researched techniques. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of the items in the transaction database or other data repositories. It is widely used in various areas such as cross marketing [4], new product development [5], personalized service and commercial credit evaluation in e-business [6], etc. The process of discovering all the association rules [7] consists of two steps: 1) discovery of all frequent itemsets that have minimum support, and 2) creating of all rules from the discovered frequent itemsets that meet the confidence threshold. Most researches have focused on efficient methods for finding frequent itemsets because it is computationally the most expensive step and solve the candidate itemsets generation by avoiding candidate generation, and reducing the time to scan database. However, they do not concentrate mining frequent itemsets based on more similar transactions in the database as found in the real world, e.g. wholesale transactions and medical prescription transactions. Therefore, in this paper, we introduce a new algorithm to mining frequent itemsets named FIAST (Frequent Itemsets Algorithm for Similar Transactions) algorithm. We use an efficient data structure, the BitTable, to aggregate transactions that have similar itemsets, and avoid candidate generation. The FIAST algorithm consists of two steps as follows: 1) constructing the ItemTable and the BitTable by a single scan of the database, and 2) mining all the frequent itemsets of two or more items.

This paper gives the preliminaries and background in Section 2. The FIAST algorithm for mining frequent itemsets is described in Section 3. We discuss the time complexity of the algorithm in Section 4. A discussion of the results is given in Section 5. Finally, we conclude our work and indicate our future work in Section 6.

II. PRELIMINARIES AND BACKGROUND

A. Frequent Itemsets Mining Definitions

In this section, we define the terms used for describing frequent itemsets mining. The conceptual basic terms are presented followed by Definitions 1-4 [8].

Definition 1. Suppose that $I = \{i_1, i_2, \ldots, i_m\}$ is a finite set of *items*. A transaction data set on I is a function $T : \{1, 2, \ldots, n\} \rightarrow P(I)$. The set $T(k)$ is the k^{th} transaction of T. The numbers 1, 2, …, n are the *transaction identifiers* (*tids*).

Definition 2. A set $K \subseteq I$ is called an *itemsets*. An itemsets with k items are called *k-itemsets*.

Definition 3. Let $T : \{1, 2, \ldots, n\} \rightarrow P(I)$ be a transaction data set on a set of items I. The support count of a subset K of the set of item I in T is the number of $sup\_count_T(K)$ given by

$$\sup\_count_T(K) = \left|\{k \mid 1 \le k \le n \text{ and } K \subseteq T(K)\}\right| \qquad (1)$$

The support of an itemsets K is the number given by

$$\sup_T(K) = \frac{\sup\_count_T(K)}{n} \qquad (2)$$

978-0-7695-3591-3/09 $25.00 © 2009 IEEE
DOI 10.1109/ICFCC.2009.54

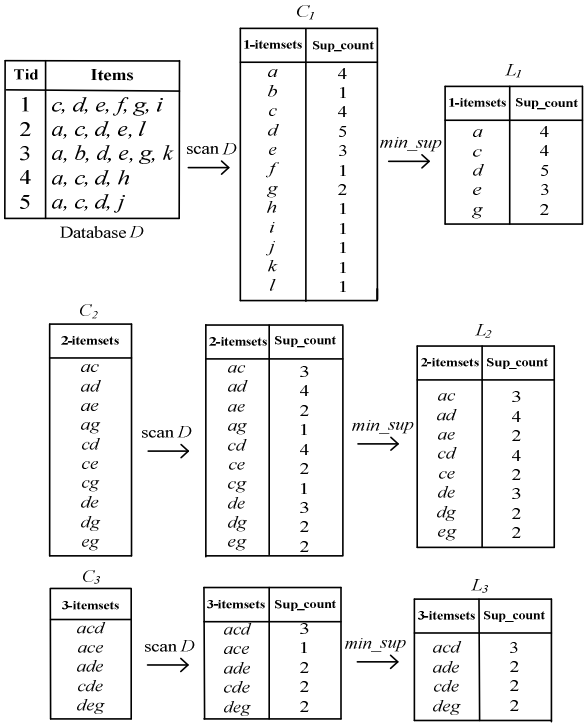


Figure 1. Mining frequent itemsets with Apriori algorithm.

Definition 4. Let *min_sup* be the threshold minimum support value specified by the user. If $sup_T(K) \geq min\_sup$, itemsets K is called a *frequent itemsets*.

B. Fundamental Algorithm: Apriori

Apriori [9] is a fundamental algorithm for mining frequent itemsets. The algorithm starts scanning the database to construct the candidate 1-itemsets (C_1) and checks the *sup_count* of the C_1 for pruning some of the C_1, if $sup_T(C_1)$ is lower than the *min_sup*. After the algorithm finds all the frequent 1-itemsets (L_1), the C_k is constructed by joining the L_{k-1}, where $k > 1$. Next, the process to prune some infrequent itemsets from the C_k is created which is called L_k. This process is repeated until no more candidate itemsets can be created. Fig. 1 illustrates mining frequent itemsets using the Apriori algorithm with the assumption that the *min_sup* is 40%.

C. Related Work

Generally, the method for finding frequent itemsets can be divided into two approaches: candidate generation-and-test and pattern growth. A basic algorithm for candidate generation-and-test is the Apriori which makes use of items for the candidate and combines the pre-given threshold value to count frequent itemsets in the database. This algorithm shows that it requires multiple database scans, as many as the longest frequent itemsets. In addition, Dong et al. [10] proposed a BitTableFI. The BitTableFI compresses the database into a BitTable which means a set of bits that represents the items [11]. Although, the BitTableFI was focused on solving the candidate itemsets generation and helping to count transactions quickly, it did not reduce the size of the candidate itemsets and the time of scanning with a large number of frequent itemsets, long itemsets, or quite low *min_sup* thresholds. Afterwards, Song et al. [12] proposed an Index-BitTableFI, which is similar to BitTableFI. The BitTable structure is used horizontally and vertically to find itemsets and *sup_count*, respectively. This works especially well for dense datasets.

For a pattern growth approach, Han et al. [13] proposed a frequent pattern tree (FP-tree) structure, which is an extended prefix-tree structure and scans the database only twice called FP-Growth algorithm. This method is a depth-first search algorithm. However, for sparse datasets the tree will be bigger and bushier. Then, Sucahyo et al. [14] proposed a Compressed FP-Tree (CFP-Tree) structure with a new frequent itemsets mining algorithm named CT-PRO. The CT-PRO makes only one pass through the database. The number of nodes in CFP-Tree could be up to as much as half less than in the corresponding FP-Tree.

As mentioned above, we found that each research is aimed at different work, e.g. solving the candidate itemsets generation, avoiding candidate generation, and reducing the time to scan the database. However, most researches do not propose mining frequent itemsets based on more similar transactions in the database. Therefore, we propose the FIAST algorithm (Frequent Itemsets Algorithm for Similar Transactions) to reduce the cost and time of mining frequent itemsets.

III. FIAST ALGORITHM

There are two steps in the FIAST algorithm as follows:

1. Creating the ItemTable and the BitTable: The ItemTable and the BitTable are constructed by a single scan of the database and aggregated the transactions that have similar itemsets.
2. Mining Frequent Itemsets: All frequent itemsets are mined by the FIAST algorithm.

The algorithm for constructing the ItemTable and the BitTable is shown in Fig. 2 and we describe how this algorithm works in Example 1.

Example 1. Let us consider the transaction database D in Fig. 1 and suppose that the *min_sup* is 40%.

Step 1: *Creating the ItemTable and the BitTable*

This step starts scanning all transactions in the database to construct both the ItemTable and the BitTable as shown in Fig. 3 (a) and (b), respectively. Then, the infrequent 1-itemsets in those tables are deleted and the index for each item is assigned as illustrated in Fig. 3 (c). Next, giving the index number in ascending order depends on the descendent *sup_count* as shown in Fig. 3 (d). Finally, the transactions of the BitTable with similar transactions are aggregated as shown in Fig. 3 (e).

```
1  Input: Database D, min_sup
2  Output: ItemTable, BitTable
3  Procedure IdentifyItems
4  For each transaction t ∈ D
5      For each item i ∈ t
6        If i ∈ ItemTable
7          Increment count of item i in ItemTable
8        Else
9          Insert i in ItemTable with count =1
10       End If
11       BitTable[t][k] = 1   //k is the k^th item of item i in
12                              ItemTable
13     End For
14 End For
15 Delete infrequent items in ItemTable and BitTable
16 Sort frequent items in ItemTable in descending order and
17     assign an index for each item
18 Sort in ascending order to rearrange the items in
19     BitTable by index
20 Aggregate transactions which have similar transactions
```

Figure 2. Algorithm for constructing the ItemTable and the BitTable.

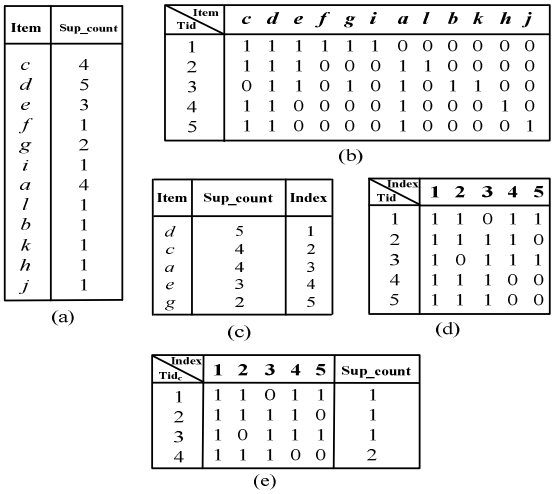


Figure 3. Creating the ItemTable and the BitTable.

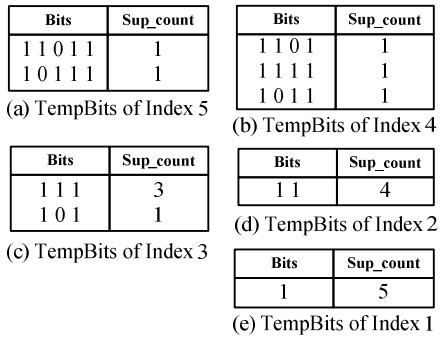


Figure 4. The TempBits of each index.

```
1  Input: ItemTable, BitTable, min_sup
2  Output: Frequent Itemsets
3  Procedure MiningFrequentItemsets
4  For each index i ∈ ItemTable from the least to the most
5       frequent 1-itemsets
6        For each transaction t ∈ BitTable
7            If bit i^th of t =1
8                Insert bit 1st to i^th of t in TempBits and
9                     their sup_count
10           End If
11       End For
12       Mine(TempBits, min_sup, FI-Tree)
13 End For
14 Traverse the FI-Tree to print the frequent itemsets
15
16 Procedure Mine(TempBits, min_sup, FI-Tree)
17 For each transaction t ∈ TempBits
18     If t.sup ≥ min_sup
19       InsertToFI-Tree(t)
20     End If
21     AndBit = ∩_{t ⊆ P(TempBits)} //not include the set itself and ∅
22     If AndBit.sup ≥ min_sup
23       InsertToFI-Tree(AndBit)
24     End If
25 End For
```

Figure 5. Algorithm for mining frequent itemsets.

Step 2: *Mining Frequent Itemsets*

All the frequent itemsets of two or more items are mined by using the FIAST algorithm that is described in Fig. 5. The algorithm aims to find the frequent itemsets, that start from the least frequent 1-itemsets in the ItemTable, obtains bits called a set of bits which is a transaction in the BitTable, uses bitwise AND operation to find itemsets and eventually, creates a FI-Tree (Frequent Index Tree) to generate frequent itemsets. For example, the index 5 in Fig. 3 (e) is a least frequent 1-itemsets that has two transactions in the BitTable. Each set of bits stores in the TempBits as illustrated in Fig. 4 (a). We use bitwise AND operation with transactions in the TempBits, their result is inserted into FI-Tree for generating frequent itemsets described by 11011 AND operation with 10111 (transaction 1st and 2nd in the TempBits) and the sum of their *sup_count* is 2. Then, the result of bitwise AND operation is 10011 (that represents index 1, 4, 5) and their *sup_count* is inserted into the FI-Tree as shown in Fig. 6 (in the left-hand path) by the number after ":" in the node that indicates the *sup_count*. We traversed by using depth-first search in the FI-Tree for generating frequent itemsets. Thus, the following frequent itemsets generated are *g*:2, *ge*:2, *gd*:2, *ged*:2. This process is repeated until the most frequent 1-itemsets in the ItemTable is constructed in FI-Tree.

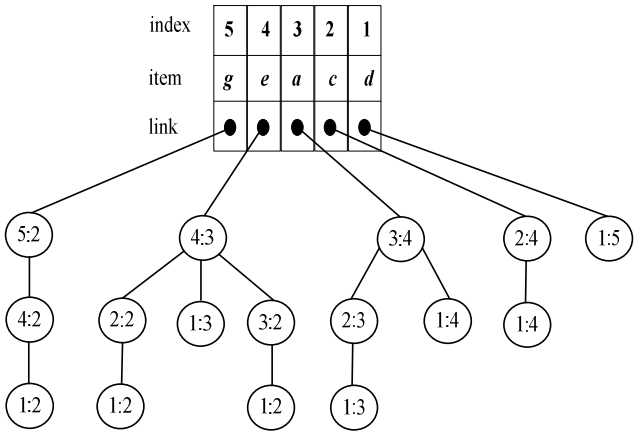


Figure 6. The complete FI-Tree.

IV. TIME COMPLEXITY

Before we discuss the time complexity of the FIAST algorithm, we summarize the notation used in the best-case and worst-case time complexity for mining frequent itemsets by the FIAST algorithm in Table 1.

TABLE I. NOTATION

Symbols	Definitions
m	All items in the database
n	All transactions in the database
p	The total number of items in all transactions
q_i	The total number of transactions in the TempBits of each index i

Lemma 1. In the best-case, the cost of mining frequent itemsets is O(p).

Proof. The best-case for the FIAST algorithm happens when their sup_T(*1-itemsets*) is lower than the *min_sup* (no frequent 1-itemsets). The BitTable stores the total number of items in all transactions (p) and the ItemTable stores *sup_count* of their items (m). The *sup_count* of items in the ItemTable is checked to determine whether there are any frequent 1-itemsets or not (m). If there is infrequent 1-itemsets, the process to generate frequent itemsets is stopped.

Equation (3) shows the time complexity of the FIAST algorithm.

$$(p + 2m) = O(p) \tag{3}$$

Therefore, the total best-case of the FIAST algorithm is O(p). ■

Lemma 2. In the worst-case, the cost of mining frequent itemsets is O(2^m).

Proof. The worst-case occurs when all m items are frequent 1-itemsets and all association of them is displayed in n transactions (all transactions are frequent itemsets). The cost of this step is:

1) Creating the ItemTable and BitTable: The ItemTable consists of all items in the database (m) and sorting items in the ItemTable with their *sup_count* ($p \log_2 p$). Thus, the cost of creating the ItemTable is ($m + p \log_2 p$). The BitTable consists of all items in all transactions (n) and sorting transactions in the BitTable for aggregating similar transactions ($n \log_2 n$). Then, the cost of creating the BitTable is ($n + n \log_2 n$). Therefore, the cost of the first step is ($m + p \log_2 p + n + n \log_2 n$).

2) Generating all frequent itemsets: The process for doing bitwise AND operations for finding itemsets of each index is shown in (4).

$$\sum_{i=1}^{m}(2^{q_i} - q_i - 1) \tag{4}$$

where $1 \le i \le m$ and constructing the FI-Tree for use to generate frequent itemsets (2^{m-1}). Traversing the FI-Tree for generating frequent itemsets is 2^{m-1}. Thus, the cost of this step is shown in (5).

$$\sum_{i=1}^{m}(2^{q_i} - q_i - 1) + 2^{m-1} + 2^{m-1} \tag{5}$$

Equation (6) shows the time complexity of the FIAST algorithm.

$$(m + p\log_2 p + n + n\log_2 n + \sum_{i=1}^{m}(2^{q_i} - q_i - 1) + 2^m) = O(2^m) \tag{6}$$

Therefore, the total worst-case of the FIAST algorithm is O(2^m). ■

V. DISCUSSION

The FIAST algorithm is a pattern growth approach without candidate generation. The algorithm shows the following profits:

1. *Minimize I/O*: The FIAST algorithm minimizes I/O resources by scanning database only once for mining frequent itemsets.
2. *Save space*: The FIAST algorithm saves space by storing items in the set of bits regardless of the size of the datasets.
3. *Reduce time*: The FIAST algorithm reduces time by using bitwise AND operation for fast finding itemsets, and using a divide-and-conquer method to reduce finding tasks into smaller ones.
4. *Appropriate for similar transactions*: The FIAST algorithm appropriates for mining frequent itemsets that have a number of similar transactions after deleting infrequent 1-itemsets, and works especially well for dense datasets.

VI. Conclusions and Future Work

In this paper, we have proposed a new algorithm called the FIAST algorithm for mining frequent itemsets based on similar transactions after deleting infrequent 1-itemsets. The algorithm applies the BitTable to aggregate transactions that have similar itemsets. The aggregation significantly reduces the number of transactions in the BitTable and the time using the divide-and-conquer method to reduce the bitwise AND operation for finding itemsets. Also, a depth-first search strategy is used to generate all frequent itemsets. We scan the database only once and avoid candidate generation to minimize I/O. We are implementing the FIAST algorithm and comparing our algorithm with other pattern growth approaches.

References

[1] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowlagde Discovery in Databases," AI Magazine, 1996, pp. 37-54.

[2] C. Rygielski, J. Wang, and D. Yen, "Data mining techniques for customer relationship management," in Technology in Society, vol. 24, 2002, pp. 483-502.

[3] S. Kotsiantis, and D. Kanellopoulos, "Association Rules Mining: A Recent Overview," Proc. GESTS International Transactions on Computer Science and Engineering, vol. 32 (1), 2006, pp. 71-82.

[4] P. Gong, C. Yang, H. Li, and W. Kou, "The Application of Improved Association Rules Data Mining Algorithm Apriori in CRM," IEEE Transactions on Knowledge and Data Engineering, 2007.

[5] S. Liao, C. Hsieh, and S. Huang, "Mining product maps for new product development," in Expert Systems with Applications, vol. 34, pp. 50-62, 2008.

[6] A. Omari, S. Conrad, and S. Alcic, "Designing a Well-Structured E-Shop Using Association Rule Mining," IEEE Transactions on Knowledge and Data Engineering, 2008.

[7] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large database," Proc. ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993, pp. 207-216.

[8] D. Simovici and C. Djeraba. "Mathematical Tools for Data Mining: Set theory, Partial Orders, Combinatorics," Springer Publishing, Inc., London, 2008. pp.273-293.

[9] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. the 20th Int'l Conference of Very Large Databases (VLDB'94), June 1994, pp. 478-499.

[10] J. Dong and M. Han, "BitTableFI: an efficient mining frequent itemsets algorithm," in Knowledge Based Systems vol.20, no.4, 2007, pp. 329–335.

[11] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A Maximal Frequent Itemset Algorithm," IEEE Transaction on Knowledge and Data Engineering, Vol. 17, No 11, November 2005, pp.1490-1504.

[12] W. Song, B. Yang, and Z. Xu, "Index-BitTableFI: An improved algorithm for mining frequent itemsets," in Knowledge Based Systems, 2008.

[13] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM SIGMOD Int'l Conference on Management of Data, Dallas, May 2000, pp. 1-12.

[14] Y.G. Sucahyo and R. Gopalan, "CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure," Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), Brighton UK, November 2004.

ประวัติผู้เขียน

ชื่อ สกุล นางสาวฟูไดละห์ ดือมอง

รหัสประจำตัวนักศึกษา 5010220092

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วท.บ. (คณิตศาสตร์ประยุกต์)	มหาวิทยาลัยสงขลานครินทร์	2550

การตีพิมพ์เผยแพร่ผลงาน

F. Duemong, L. Preechaveerakul and S. Vanichayobon. "FIAST: A Novel Algorithm for Mining Frequent Itemsets". In Proceedings of 2009 International Conference on Future Computer and Communication, Kuala Lumpur, Malaysia, pp.140-144, 2009.