อัลกอริทึมแฮชเบสฟาสท์อัพเดทเพื่อค้นหากฎความสัมพันธ์ในฐานข้อมูลแบบเพิ่มขยาย

HASH-BASED FAST UPDATE ALGORITHM

แจ่มจันทร์ คงปาน

JAMJUN KONGPAN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2552

KMITL-2010-IT-M-001-007

HASH-BASED FAST UPDATE ALGORITHM

JAMJUN KONGPAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2010

KMITL-2010-IT-M-001-007

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ อัลกอริทึมแฮชเบสฟาสท์อัพเดทเพื่อค้นหากฎความสัมพันธ์ในฐานข้อมูลแบบเพิ่มขยาย
Hash-Based Fast Update Algorithm

นักศึกษา นางสาวแจ่มจันทร์ คงปาน

รหัสประจำตัว 48066401

ปริญญา วิทยาศาสตรมหาบัณฑิต

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาวิทยานิพนธ์ รองศาสตราจารย์ ดร.วรพจน์ กรีสุระเดช

คณะกรรมการสอบวิทยานิพนธ์	ลายมือชื่อ
ผู้ช่วยศาสตราจารย์ ดร.สมหญิง ไทยนิมิต	
รองศาสตราจารย์ ดร.อาริต ธรรมโน	
ผู้ช่วยศาสตราจารย์ ดร.พรฤดี เนติโสภากุล	
รองศาสตราจารย์ ดร.วรพจน์ กรีสุระเดช	
ดร. ปานวิทย์ ธุวะนุติ	

วัน / เดือน/ ปี ที่สอบ วันศุกร์ที่ 4 มิถุนายน 2553 เวลา 10.30 น.

สถานที่สอบ ณ ห้อง M 23 ชั้น M คณะเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศรับรองแล้ว

(รองศาสตราจารย์ ดร.จันทรัมพร สถิตวิริยวงศ์)

คณบดีคณะเทคโนโลยีสารสนเทศ

วันที่ 7 **เดือน** มิถุนายน **พ.ศ.** 2553

หัวข้อวิทยานิพนธ์	อัลกอริทึมแฮชเบสฟาสท์อัพเดทเพื่อค้นหากฎความสัมพันธ์ในฐานข้อมูลแบบเพิ่มขยาย
นักศึกษา	นางสาวแจ่มจันทร์ คงปาน
รหัสประจำตัว	48066401
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2553
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.วรพจน์ กรีสุระเดช

บทคัดย่อ

อัลกอริทึม FUP เป็นอัลกอริทึมที่รู้จักกันดีสำหรับการค้นหากฎความสัมพันธ์ (Association Rule Discovery) ในฐานข้อมูลที่มีการเปลี่ยนแปลง เมื่อมีทรานแซคชั่นใหม่ถูกเพิ่มเข้ามาในฐานข้อมูล ในงานวิจัยชิ้นนี้ได้นำเสนอแนวทางในการค้นหากฎความสัมพันธ์ของข้อมูล โดยใช้ Hash-Based Fast Update Algorithm (HFUP) เพื่อปรับปรุงการทำงานของอัลกอริทึม FUP บนพื้นฐานของเทคนิค direct hashing and pruning ซึ่งอัลกอริทึม HFUP จะช่วยลดจำนวนของ Candiadte Itemsets และทำให้เวลาในการประมวลผลข้อมูลน้อยกว่าอัลกอริทึม FUP

Thesis Title	Hash-Based Fast Update Algorithm (HFUP)
Student	Miss.Jamjun Kongpan
Student ID.	48066401
Degree	Master of Science
Programme	Information Technology
Year	2010
Thesis Advisor	Asso. Prof. Dr. Worapoj Kreesuradej

ABSTRACT

FUP algorithm is a well-known algorithm for discoverying association rules in a dynamic database when new transaction data are added to the database. This thesis proposes Hash-Based Fast Update Algorithm (HFUP) to improve the performance of FUP algorithm. Based on direct hashing and pruning technique, the new algorithm has less candidate itemsets and execution time than that of FUP.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้อย่างดีด้วยคำแนะนำ คำปรึกษา และการแก้ปัญหาต่าง ๆ จาก รศ.ดร. วรพจน์ กรีสุระเดช ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ข้าพเจ้าขอขอบพระคุณท่านอาจารย์เป็นอย่างสูง

ขอขอบพระคุณคณาจารย์คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณเพื่อน ๆ พี่ ๆ น้อง ๆ ในคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

ขอขอบคุณบัณฑิตศึกษา และบัณฑิตวิทยาลัย คณะเทคโนโลยีสารสนเทศที่ให้ความช่วยเหลือในเรื่องต่าง ๆ

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกเรื่อง ๆ ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

แจ่มจันทร์ คงปาน

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบัน การดำเนินธุรกิจต่างๆ ไม่ว่าจะเป็นธุรกิจขนาดเล็ก ขนาดกลาง หรือขนาดใหญ่ ต่างก็มีอัตราการแข่งขันกันสูง ดังนั้นบริษัทที่ประกอบธุรกิจจำนวนมากมักจะทำการวิเคราะห์ข้อมูลที่ผ่านมาในอดีตเพื่อหาข้อมูลที่เกี่ยวข้องกับลักษณะพฤติกรรมการบริโภคของลูกค้า เพื่อนำข้อมูลเหล่านี้ไปใช้ในการสร้างแผนกลยุทธ์ทางการตลาด หรือรายการส่งเสริมการขาย (เช่น การออกแบบการจัดวางสินค้าที่สามารถขายร่วมกันได้ไว้ใกล้ๆ กัน, การออกแบบคูปองเพื่อใช้ลดราคาสินค้าในครั้งต่อไป หรือการออกแบบลักษณะการจัดวางสินค้าบนชั้นสินค้า เป็นต้น) เพื่อเพิ่มประสิทธิภาพและทำให้ธุรกิจได้รับกำไรสูงสุด และเนื่องจากความก้าวหน้าของเทคโนโลยี Bar-Code ทำให้การจัดเก็บข้อมูลการขายสินค้าได้รับความสะดวกรวดเร็ว และข้อมูลที่ได้มีความน่าเชื่อถือสูง

การค้นหาความรู้จากฐานข้อมูล (Knowledge Discovery in Databases :KDD) เป็นการประยุกต์ใช้ข้อมูลที่เก็บอยู่ในฐานข้อมูลให้เกิดประโยชน์สูงสุดแก่ธุรกิจ หรือหน่วยงานต่างๆ ที่เป็นเจ้าของข้อมูลเหล่านั้น โดยที่การประยุกต์ใช้ข้อมูลที่กล่าวถึงนี้มักจะเป็นการสรุปภาพรวมของข้อมูลในฐานข้อมูล, การวิเคราะห์แนวโน้มการเปลี่ยนแปลงของข้อมูล หรือการค้นหาความสัมพันธ์ที่ซ่อนอยู่ภายในกลุ่มของข้อมูล ซึ่งสิ่งที่ค้นพบจากข้อมูลเหล่านี้จะเรียกว่า ความรู้ (knowledge) และความรู้ที่ได้นั้นสามารถนำไปใช้วางแผนการดำเนินธุรกิจ หรือนำไปใช้เพื่อก่อให้เกิดประโยชน์แก่หน่วยงานนั้น ๆ

ปัจจุบันข้อมูลที่ต้องนำมาวิเคราะห์มีแนวโน้มที่จะเพิ่มขึ้นมาก ซึ่งจะทำให้การวิเคราะห์ข้อมูลมีความซับซ้อนมากขึ้นและต้องใช้เวลามากขึ้น ทำให้ธุรกิจต่างๆ เริ่มสนใจวิธีที่จะนำข้อมูลมาวิเคราะห์เพื่อนำข้อมูลที่ถูกต้องมาประยุกต์ใช้กับธุรกิจอย่างเหมาะสม ซึ่งวิธีดังกล่าวเรียกว่าเหมืองข้อมูล (Data Mining) คือการนำข้อมูลมาวิเคราะห์โดยใช้เทคนิคต่างๆ เข้ามาช่วย เพื่อนำข้อมูลที่ผ่านการวิเคราะห์ไปใช้ในธุรกิจให้เกิดประโยชน์สูงสุด และเทคนิคกฎความสัมพันธ์ (Association Rule) เป็นเทคนิคหนึ่งของการทำเหมืองข้อมูล ที่ช่วยในการค้นหาความสัมพันธ์ของข้อมูลในกลุ่มข้อมูลขนาดใหญ่ (Data Set) เพื่อนำไปใช้ในการวิเคราะห์และทำนายปรากฏการณ์ต่างๆ ซึ่งจะเห็นว่า Apriori Algorithm เป็นอัลกอริทึมพื้นฐานในการหากฎความสัมพันธ์ โดยใช้หลักการในลักษณะ level-wise search คือการทำซ้ำการทำงานเป็นรอบๆ คือการนำค่าใน k-itemsets มาใช้ในการค้นหาไอเทมใน (k+1)-itemsets โดยที่ไอเทมจะต้องมีค่าสนับสนุนขั้นต่ำ (Minimum Support) มากกว่าหรือเท่ากับค่าที่กำหนดไว้ โดยจะทำการค้นหาไอเทมไปเรื่อยๆ จนกว่าจะไม่สามารถค้นหาไอเทม

ใน (k+1)-itemsets ได้ ดังนั้นการใช้ Apriori Algorithm จะช่วยในการสร้างกฎความสัมพันธ์ที่มีประสิทธิภาพ และลดเวลาในการค้นหาข้อมูล ซึ่งเหมาะกับฐานข้อมูลที่ไม่มีการเปลี่ยนแปลงข้อมูล แต่ในการทำงานส่วนใหญ่ฐานข้อมูลมีการเปลี่ยนแปลงอยู่ตลอดเวลา ทำให้กฎความสัมพันธ์ที่มีความน่าเชื่อถือ (Strong Association Rule) บางกฎอาจจะไม่ใช่กฎความสัมพันธ์ที่มีความน่าเชื่อถือเมื่อฐานข้อมูลมีการเปลี่ยนแปลง ดังนั้นจึงเกิดเทคนิค Increment Association Rule Discovery เพื่อใช้ในการค้นหากฎความสัมพันธ์ในฐานข้อมูลที่มีการเปลี่ยนแปลง และช่วยรักษาประสิทธิภาพของกฎความสัมพันธ์เดิม โดยใช้ Fast Update Algorithm (FUP) มาช่วยในการค้นหากฎความสัมพันธ์ โดยหลักการทำงานคือจะสร้าง Candidate Set ขนาดเล็กเพื่อค้นหา New Large Itemsets และทำการปรับปรุงฐานข้อมูลโดยการนำข้อมูลในฐานข้อมูลเดิมมารวมเข้ากับข้อมูลใหม่ที่เพิ่มขึ้นเพื่อหา New Large Itemsets ทำให้การค้นหากฎความสัมพันธ์มีประสิทธิภาพมากขึ้น และลดเวลาในการค้นหากฎความสัมพันธ์ได้ดีกว่า Apriori Algorithm ซึ่งเรายังได้นำวิธีการค้นหาความสัมพันธ์แบบ Direct Hashing and Pruning (DHP) มาปรับใช้ร่วมกับ FUP เพื่อเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ และลดเวลาในการค้นหาอีกด้วย

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. ศึกษาการค้นหากฎความสัมพันธ์กรณีที่ฐานข้อมูลมีการเพิ่มข้อมูล
2. ศึกษาการค้นหากฎความสัมพันธ์แบบ Direct Hashing and Pruning (DHP)
3. ปรับปรุงอัลกอริทึมสำหรับการเพิ่มข้อมูลเข้ามาในฐานข้อมูล ให้มีความสามารถในการค้นหากฎความสัมพันธ์ของข้อมูล ด้วยวิธีการค้นหากฎความสัมพันธ์แบบ Direct Hashing and Pruning (DHP) โดยเราจะเรียกวิธีการค้นหากฎความสัมพันธ์แบบใหม่นี้ว่า A Hash-Based Fast Update Algorithm (HFUP) และเมื่อมีการกล่าวถึงต่อๆไปจะขอใช้ชื่อย่อ HFUP แทน

1.3 สมมติฐานของการศึกษา

ในงานวิจัยนี้ได้มุ่งเน้นไปที่การนำแนวคิดเรื่องการค้นหากฎความสัมพันธ์ข้อมูลแบบ Direct Hashing and Pruning (DHP) มาใช้ร่วมกับแนวคิดในการค้นหากฎความสัมพันธ์กรณีที่ฐานข้อมูลมีการเพิ่มข้อมูลใหม่เข้ามา (Fast Update Algorithm : FUP) โดยมีแนวคิดและทฤษฎีที่น่าสนใจดังนี้

1. Knowledge Discovery in Database
2. Association Rule

1. Knowledge Discovery in Database
2. Association Rule
3. A Hash-Based Fast Update Algorithm

1.4 ขอบเขตการวิจัย

วิทยานิพนธ์ฉบับนี้ได้ศึกษาเกี่ยวกับวิธีการที่นำกฎความสัมพันธ์แบบ Direct Hashing and Pruning (DHP) มาใช้ร่วมกับการเพิ่มขึ้นของข้อมูลในฐานข้อมูล โดยอัลกอริทึม FUP เป็นอัลกอริทึมสำหรับใช้ค้นหากฎความสัมพันธ์ของข้อมูลในกรณีเพิ่มข้อมูลใหม่เข้าสู่ฐานข้อมูล จึงได้ปรับปรุงอัลกอริทึม FUP ร่วมกับอัลกอริทึม DHP เพื่อให้สามารถค้นหากฎความสัมพันธ์ได้อย่างรวดเร็วมากขึ้น

1.5 ขั้นตอนของการศึกษา

ขั้นตอนในการศึกษาวิธีการวิจัย จากเริ่มจนถึงสิ้นสุดการทำงานวิจัยดังนี้

1.5.1 ศึกษาทฤษฎีและงานวิจัยจากเอกสาร บทความต่างๆ ในส่วนที่เกี่ยวข้องกับการทำงานวิจัยในฉบับนี้

1.5.2 กำหนดหัวข้อ วัตถุประสงค์ ขอบเขตการทำงานวิจัย

1.5.3 วิเคราะห์และปรับปรุงอัลกอริทึม

1.5.4 เตรียมข้อมูลเพื่อใช้ทดลอง

1.5.5 พัฒนาโปรแกรม ทดสอบ และแก้ไขข้อผิดพลาด

1.5.6 รวบรวมผลการทดลองจากการทำงานของโปรแกรม

1.5.7 วิเคราะห์และสรุปผลการทดลอง

1.5.8 ดำเนินการจัดทำเอกสารงานวิจัย

วิทยานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาทั้งหมดออกเป็น 5 บท คือ

บทที่ 1 ความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ สมมติฐาน ทฤษฎีที่ใช้ ขอบเขตของการวิจัย และขั้นตอนการศึกษา

บทที่ 2 ทฤษฎีพื้นฐานที่ใช้ในการวิจัย

บทที่ 3 การค้นหากฎความสัมพันธ์โดยใช้อัลกอริทึม HFUP

บทที่ 4 วิเคราะห์ผลการทดลอง

บทที่ 5 บทสรุปของงานวิจัย

บทที่ 2
ทฤษฎีพื้นฐานที่ใช้ในการวิจัย

2.1 การค้นหากฎความสัมพันธ์ของข้อมูล (Association Rule Discovery)

การค้นหากฎความสัมพันธ์ของข้อมูลเป็นเทคนิคหนึ่งของการทำ Data Mining ซึ่งเป็นวิธีที่ได้รับความสนใจอย่างแพร่หลายในการค้นหาความสัมพันธ์ของข้อมูลในกลุ่มข้อมูลขนาดใหญ่ (Data Set) เพื่อนำไปใช้ในการวิเคราะห์การซื้อสินค้าของลูกค้าที่เรียกว่า "Market Basket Analysis" ซึ่งเป็นเทคนิคที่นำไปใช้ในด้านการตลาด เพื่อวิเคราะห์พฤติกรรมการซื้อของลูกค้า โดยหาความสัมพันธ์ระหว่างสินค้าต่างๆ ที่ลูกค้าซื้อ การค้นหาความสัมพันธ์สามารถช่วยผู้ขายพัฒนากลยุทธ์ด้านการตลาด โดยพิจารณาจากสินค้าที่มักจะถูกซื้อไปพร้อมๆ กัน เช่น ถ้าลูกค้าซื้อนม ลูกค้ามักจะซื้อขนมปังพร้อมกัน ข้อมูลเหล่านี้สามารถนำไปสู่การเพิ่มยอดขาย โดยการช่วยผู้ขายในการวางแผนทางการตลาด และการวางแผนการจัดชั้นวางสินค้า เช่น การวางขนมปังกับนมไว้ใกล้กัน อาจจะเพิ่มยอดขายสินค้าทั้งสองชนิด หรืออาจจะวางไว้คนละมุมของร้าน เพื่อที่ว่าลูกค้าซื้อนมแล้วต้องการที่จะซื้อขนมปังก็จะต้องเดินผ่านสินค้าชนิดอื่นๆ ทำให้มีโอกาสที่ลูกค้าจะซื้อสินค้าตัวอื่นๆ เพิ่มขึ้นด้วย

รูปแบบของกฎความสัมพันธ์ที่ได้อยู่ในรูปแบบ "IF X Then Y" หรือ "IF Codition 1 Then Condition 2" โดยที่ X และ Y เกิดขึ้นพร้อมกันในทรานแซคชั่นเดียวกัน หรือเรียกได้ว่าถ้าเหตุการณ์ X หรือ Condition 1 เกิดขึ้นแล้ว จะเกิดเหตุการณ์ Y หรือ Condition 2 ขึ้นด้วย โดยใช้สัญลักษณ์ $X \longrightarrow Y$ ซึ่งมีตัวแปรที่ใช้ในการวัดค่าข้อมูลอยู่ด้วยกัน 2 ตัวแปร คือ

1. ค่าสนับสนุน (Support factor) คือเปอร์เซ็นต์ที่แสดงความสัมพันธ์ระหว่างจำนวนรายการของข้อมูล $X \longrightarrow Y$ ที่เกิดขึ้นเปรียบเทียบกับจำนวนรายการของข้อมูลทั้งหมด เช่น จากกฎความสัมพันธ์ คอมพิวเตอร์ $\longrightarrow$ ซอฟต์แวร์ หมายความว่า เมื่อลูกค้าซื้อเครื่องคอมพิวเตอร์ก็จะซื้อซอฟต์แวร์ด้วย โดยคำนวณค่าเปอร์เซ็นต์โดยใช้สูตร

$$\frac{\text{จำนวนลูกค้าที่ซื้อคอมพิวเตอร์และซอฟต์แวร์ร่วมกัน}}{\text{จำนวนสินค้าที่ขายทั้งหมดในร้าน}} \qquad (2.1)$$

ดังนั้นค่าสนับสนุน (Support factor) สามารถเขียนให้อยู่ในรูปแบบสมการได้ดังนี้

$$Support(X \longrightarrow Y) = P(X \cup Y) \qquad (2.2)$$

2. ค่าความเชื่อมั่น (Confidence factor) คือเปอร์เซ็นต์ที่แสดงค่าความสัมพันธ์ที่ซ่อนอยู่ในตัวเองกับสินค้ารายการอื่น เช่น จากกฎในการหาความสัมพันธ์ คอมพิวเตอร์ $\longrightarrow$

ซอฟต์แวร์ หมายความว่า เมื่อลูกค้าซื้อเครื่องคอมพิวเตอร์อาจจะซื้อซอฟต์แวร์ด้วย โดยคำนวณค่าเปอร์เซ็นต์โดยใช้สูตร

$$\frac{\text{จำนวนลูกค้าที่ซื้อคอมพิวเตอร์และซอฟต์แวร์ร่วมกัน}}{\text{จำนวนลูกค้าที่ซื้อคอมพิวเตอร์ทั้งหมด}} \quad (2.3)$$

ดังนั้นค่าความเชื่อมั่น (Confidence factor) สามารถเขียนเป็นสมการได้ดังนี้

$$Confidence(X \rightarrow Y) = P(X \mid Y) \quad (2.4)$$

ถ้ากฎความสัมพันธ์เป็นไปตาม Minimum Support Threshold (Min_Sup) และ Minimum Confidence Threshold (Min_Conf) เรียกว่า Strong Association

2.2 นิยามเบื้องต้นของกฎความสัมพันธ์

กฎความสัมพันธ์เป็นเทคนิคที่ใช้สำหรับค้นหากฎความสัมพันธ์ โดยกำหนดให้

$I = \{ i_1, i_2, \ldots, i_n \}$ คือกลุ่มของข้อมูลแต่ละตัวที่ใช้ในการหากฎความสัมพันธ์ เรียกว่าไอเทม (Items)

$D = \{ T_1, T_2, \ldots, T_m \}$ คือฐานข้อมูลที่ประกอบไปด้วยทรานแซคชั่น (T)

T คือเซตของไอเทมก็ต่อเมื่อ $T \subseteq I$

TID คือแต่ละทรานแซคชั่น T สัมพันธ์กันด้วยตัวระบุ (identifier) ที่เป็นหนึ่งเดียว (unique) เพื่อใช้แยกแยะความแตกต่างของแต่ละทรานแซคชั่น

X คือเซตของไอเทมในแต่ละทรานแซคชั่นก็ต่อเมื่อ $X \subseteq T$

โดยปกติแล้วกฎความสัมพันธ์สามารถจัดให้อยู่ในรูปของ $X \rightarrow Y$ คือ IF X THEN Y โดยที่ $X \subset I$ กับ $Y \subset I$ และ $X \cap Y = \varnothing$

Transaction ID (TID)	Item
1	Bread, Butter, Eggs
2	Butter, Eggs, Milk
3	Bread
4	Bread, Butter

รูปที่ 2.1 แสดงข้อมูลการซื้อสินค้าของลูกค้า

จากตัวอย่างในรูปที่ 2.1 แสดงข้อมูลที่ใช้ในการหากฎความสัมพันธ์ โดยมีรายละเอียดของข้อมูลดังนี้

กลุ่มของไอเทม I = {Bread, Butter, Eggs}

$D = \{ T_1, T_2, ..., T_m \} = \{ T1, T2, T3, T4 \}$

ถ้า $T \subseteq I$ ก็ต่อเมื่อ I เป็นกลุ่มของ Items ทั้งหมดที่เป็นไปได้ I = {Bread, Butter, Eggs, Milk}

Itemsets คือกลุ่มของ Item = {Butter, Eggs, Milk}

k-Itemset คือ Itemset ที่มีข้อมูล k-items = { Bread, Butter, Eggs} คือ 3-itemset

Frequency Itemset หรือ Large Itemset คือชุดของ Itemsets ที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุด (Minimum Support) แล้วกลุ่มของ Large k-itemsets เขียนแทนด้วย L_k

กฎของ Association Rule มีตัววัดหลักๆ 2 ตัว คือ Support factor และ Confidence factor เราสามารถกล่าวได้ว่ากฎ $X \longrightarrow Y$ ที่สร้างจากเซตของทรานแซคชั่น D มีค่า Support factor เท่ากับ s ก็ต่อเมื่อ (จำนวนทรานแซคชั่น T_i ที่มีทั้งรายการสินค้า X และ Y อยู่ด้วยกัน) / (จำนวน ทรานแซคชั่น T_i ทั้งหมดที่อยู่ในเซตของทรานแซคชั่น D) เท่ากับ s และเราสามารถกล่าวได้ว่ากฎ $X \longrightarrow Y$ ที่สร้างจากเซตของทรานแซคชั่น D มีค่า Confidence factor เท่ากับ c ก็ต่อเมื่อ (จำนวน ทรานแซคชั่น T_i ที่มีทั้งรายการสินค้า X และ Y อยู่ด้วยกัน) / (จำนวนทรานแซคชั่น T_i ที่มีรายการสินค้า X) เท่ากับ c

ตัวอย่างในรูปที่ 2.1 แสดงให้เห็นว่ามีกลุ่มของไอเทม {Bread, Butter} อยู่ในทรานแซคชั่นที่ 1 และทรานแซคชั่นที่ 4 ดังนั้นค่าสนับสนุนสำหรับกลุ่มไอเทม {Bread, Butter} เมื่อคิดเป็นเปอร์เซ็นต์คือ 2/4*100 = 50% ซึ่งจะเห็นว่าค่าสนับสนุน 50% ที่ได้มาจากกลุ่มไอเทม {Bread, Butter} จำนวน 2 ทรานแซคชั่นจากจำนวนทรานแซคชั่นทั้งหมดในฐานข้อมูลคือ 4 โดยกฎ {Bread} $\longrightarrow$ {Butter} มีความหมายคือ ลูกค้าที่ซื้อขนมปังและเนยไปด้วยกันคิดเป็น 50% เมื่อเทียบกับรายการขายทั้งหมด และเราสามารถหาค่าเปอร์เซ็นต์ของความเชื่อมั่นจากกฎ {Bread} $\longrightarrow$ {Butter} ได้คือ 2/3*100 = 66.7% ซึ่งจะเห็นว่าค่าความเชื่อมั่น 66.7% ที่ได้มาจากกลุ่มไอเทม {Bread, Butter} จำนวน 2 ทรานแซคชั่น และกลุ่มไอเทม {Bread} จำนวน 3 ทรานแซคชั่นนั้น หมายความว่ามีลูกค้าที่ซื้อขนมปังทั้งหมด 66.7% จะซื้อเนยไปด้วย

ปัญหาของการทำ Mining Association Rule ก็คือการสร้าง Association Rule ทั้งหมดที่มีค่าของ Support factor ที่มากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุด (Minimum Support) และมีค่าของ Confidence factor ที่มากกว่าหรือเท่ากับค่าความเชื่อมั่นน้อยที่สุด (Minimum Confidence) ดังนั้น

เราสามารถแบ่งขั้นตอนของการค้นหา Association Rule ออกเป็นขั้นตอนย่อยๆ ได้ 2 ขั้นตอน ดังต่อไปนี้

1. ค้นหา Itemsets (เซตของรายการสินค้าที่ถูกซื้อไปพร้อมๆ กัน) ทั้งหมดที่มีค่า Support factor มากกว่าหรือเท่ากับ Min_Sup ซึ่งเราสามารถเรียก Itemset เหล่านี้ว่า Large Itemsets หรือ Frequent Itemsets โดยที่ขนาดของ Itemset จะถูกนำเสนอด้วยจำนวนรายการสินค้าที่อยู่ใน Itemset เช่น ถ้าขนาดของ Itemset เท่ากับ k ดังนั้นจะสามารถเรียก Itemset นี้ว่า k-Itemset
2. การสร้าง Association Rule จาก Large Itemsets โดยกฎที่ได้จะถูกต้องก็ต่อเมื่อมีค่า Confidence Factor มากกว่าหรือเท่ากับ Min_Conf

2.3 การค้นหากฎความสัมพันธ์ด้วยอัลกอริทึมอะพิโอริ (Apriori Algorithm)

อัลกอริทึมอะพิโอริเป็นอัลกอริทึมหนึ่งที่นิยมใช้ในการหากฎความสัมพันธ์ของข้อมูล และนำไปใช้ในการวิเคราะห์หรือทำนายปรากฏการณ์ เป็นอัลกอริทึมดั้งเดิมที่นำมาค้นหากฎความสัมพันธ์ที่น่าสนใจ ถึงแม้ว่าจะมีอัลกอริทึมอื่นที่มีประสิทธิภาพดีกว่า แต่ก็มีพื้นฐานมาจากอัลกอริทึมนี้เป็นส่วนใหญ่ ดังนั้นจึงใช้อัลกอริทึมนี้ในการอธิบายการหากฎความสัมพันธ์ ซึ่งประกอบด้วย 2 ขั้นตอนดังนี้

2.3.1 การหา Large Itemset หรือ Frequent Itemset

การหา Large Itemset มีขั้นตอนการทำงานที่ทำซ้ำไปเรื่อยๆ จนกว่าจะไม่สามารถหา Large Itemset ได้อีก กล่าวคือ Large k-Itemsets จะถูกใช้ในการหา Large (k+1)-Itemsets (ในที่นี้ใช้ L_1 เป็นสัญลักษณ์แทน Large 1-Itemset และ L_k เป็นสัญลักษณ์แทน Large k-Itemsets) กล่าวคือ L_1 จะถูกใช้ในการหา Large 2-Itemsets หรือ L_2 และ L_2 ก็จะถูกใช้เพื่อหา Large 3-Itemsets หรือ L_3 เช่นนี้ไปเรื่อยๆ จนกว่าจะไม่สามารถหา Large Itemset ได้อีก เพื่อเป็นการเพิ่มประสิทธิภาพของอัลกอริทึมโดยการช่วยลดพื้นที่ที่จะต้องค้นหา Large Itemset ในฐานข้อมูล Itemset ใดๆ ที่มีค่าสนับสนุนน้อยกว่าค่าสนับสนุนน้อยที่สุดถือว่า Itemset นั้นๆ ไม่เป็น Large Itemset โดยมีตัวแปรที่นำมาใช้ดังนี้

- D คือฐานข้อมูลที่เก็บแต่ละทรานแซคชั่น
- L_k คือเซตของ Large k-itemset ซึ่งทุกเซตมีความถี่หรือค่าสนับสนุนในการเกิดมากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุด
- k-Itemset คือเซตของข้อมูลที่มีจำนวนสมาชิก k ตัว
- C_k คือเซตของ Candidate k-itemset เป็นเซตที่สร้างมาจาก Large k-1 itemset

ขั้นตอนการหา Large Itemset ประกอบด้วย 2 ขั้นตอน ดังนี้

1. ขั้นตอนการ join

ในการหา L_k นั้น เป็นการนำ L_{k-1} มาทำการ join กับตัวมันเอง เพื่อหา Candidate k-Itemset (C_k) ซึ่งจะเขียนเป็นสัญลักษณ์คือ $L_{k-1} \bowtie L_{k-1}$ วิธีการเชื่อมความสัมพันธ์ทำโดยการพิจารณาว่าจะเชื่อมความสัมพันธ์จากจำนวน Itemset เท่าไหร่ ไปเป็นเท่าไหร่ โดยกำหนดให้หา C_k ซึ่งจะเกิดจากการเชื่อความสัมพันธ์ระหว่าง L_{k-1} และ L_{k-1} พิจารณาว่า Itemset ใดสามารถเชื่อมความสัมพันธ์กันได้ ให้พิจารณาที่ Item ที่ 1 ถึง Item ตัวรองสุดท้ายของทั้งสอง Itemset หากเหมือนกันก็สามารถเชื่อมความสัมพันธ์กันได้ เช่น {I1, I2}, {I1, I3}, {I2, I3} จะเห็นว่า Itemset ที่ 1 และ 2 สามารถเชื่อมความสัมพันธ์กันได้ เพราะลำดับแรกของ Itemset เหมือนกัน เมื่อเชื่อมความสัมพันธ์จะได้ {I1, I2, I3} และในการเชื่อมความสัมพันธ์ต่อๆ ไป Itemset ที่ 1 และ 3 ไม่สามารถเชื่อมความสัมพันธ์กันได้ เพราะลำดับแรกของ Item ทั้งสองไม่เหมือนกัน

Algorithm: Apriori. Find frequent itemsets using and iterative level-wise approach based on candidate generation.

Input: Database, *D*, of transaction; minimum support threshold, *min_sup*.

Output: *L*, frequent itemsets in *D*.

Method:

L_1= find_frequent_1-itemsets(D);
for ($k = 2$; $L_{k-1} \neq \phi$; k++) {
  C_k = **apriori_gen**(L_{k-1}, *min_sup*);
    for each transaction $t \in D$ { // สแกน D สำหรับนับจำนวนแต่ละทรานแซคชั่น
      C_t = subset(C_k, t);
      for each candidate $c \in C_t$
        c.count++;
    }
    $L_k = \{c \in C_k \mid c.count \geq min\_sup\}$
}
Return $L = \cup_k L_k$;

รูปที่ 2.2 แสดง Apriori Algorithm

2. ขั้นตอนการ prun

Candidate k-Itemset (C_k) คือ Superset ของ L_k ซึ่งอาจจะมีจำนวนมาก เนื่องมาจากการ join กันของ L_{k-1} จึงต้องมีการลดจำนวนของ C_k โดย Item ที่ได้จากการ join ของ L_{k-1} กับ L_{k-1} จะเป็น C_k ได้ก็ต่อเมื่อ ทุกๆ Subset ของตัวมันเองเป็นสมาชิกของ L_{k-1} หากมี Subset ใดไม่เป็นสมาชิกของ L_{k-1} แล้วนั้น Itemsets ก็จะถูกลบทิ้งไป เช่น C_3 คือ {I1, I2, I3} ดังนั้น Subset ของ {I1, I2, I3} จะต้องมี Itemsets อยู่ใน L_2 คือ {I1, I2}, {I1, I3}, {I2, I3} ครบทุกตัว

และสมาชิกใน C_k จะต้องมีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุดด้วย ไม่เช่นนั้น Itemsets ตัวนั้นก็จะต้องถูกตัดทิ้งออกไปจาก C_k

procedure apriori_gen (L_{k-1} : frequent (k-1)-itemsets; *min_sup*: minimum support threshold)

for each itemset $l_1 \in L_{k-1}$

for each itemset $l_2 \in L_{k-1}$

if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2] \wedge \ldots \wedge (l_1[k\text{-}2] = l_2[k\text{-}2] \wedge$ $(l_1[k\text{-}1] < l_2[k\text{-}1])$ then {

$c = l_1 \bowtie l_2$; // join step: เป็นการสร้าง Candidate Itemsets

if **has _infrequent_subset**(c, L_{k-1}) then

delete c; //ขั้นตอนการ prune ออกจาก Candidate Itemsets

else add c to C_k;

}

return C_k;

รูปที่ 2.3 แสดง procedure apriori_gen

procedure has_infrequnt_subset (c: candidate k-itemset; L_{k-1}: frequent (k-1) – itemset);

for each (k-1)-subset s of c

if $s \notin L_{k-1}$ then

return TRUE;

return FALSE;

รูปที่ 2.4 แสดง procedure has_infrequnt_subset

2.3.2 การสร้างกฎความสัมพันธ์จาก Large Itemsets

เมื่อได้ค่า Large Itemsets ทั้งหมดแล้วก็จะนำมาสร้างเป็นกฎความสัมพันธ์ โดยกฎความสัมพันธ์ได้มาจากแต่ละ Large Itemsets (L_k เมื่อ k = 2, …, n) แล้วหา Subset ของ Large Itemsets ของ L_k และทุกๆ Subset ของ L_k ยกเว้นเซตว่างจะได้ผลลัพธ์คือ $s \longrightarrow (L-s)$

สูตรในการหาค่า Confidence คือ

$$\text{Confidence}(X \longrightarrow Y) = \frac{\text{support\_count}(X \cup Y)}{\text{support\_count}(X)} \qquad (2.5)$$

โดยที่

- support_count($X \cup Y$) คือจำนวนของทรานแซคชั่นที่มี Itemset $X \cup Y$
- support_count(X) คือจำนวนทรานแซคชั่นที่มี Itemset A

ซึ่งกฎความสัมพันธ์ที่ได้จะมีความน่าเชื่อถือก็ต่อเมื่อ ผลลัพธ์ของแต่ละกฎความสัมพันธ์จะต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นน้อยที่สุด Minimum Confidence

2.3.3 อธิบายการทำงานของอัลกอริทึมอะพิโอริ

1. ทำการหาค่า L_1 โดยได้จากการสร้าง C_1 (ได้จากแต่ละไอเทมเซตที่มีในทรานแซคชั่นที่อยู่ในฐานข้อมูล) นำแต่ละไอเทมเซตใน C สแกนหาในฐานข้อมูลว่ามีค่าสนับสนุนเท่าไร แล้วนำค่าสนับสนุนแต่ละไอเทมเซตมาเปรียบเทียบกับค่าสนับสนุนน้อยที่สุดว่ามีค่ามากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุดหรือไม่ ไอเทมเซตที่ผ่านเกณฑ์จะปรากฏอยู่ใน L_1

2. ทำการหา C_2 โดยได้จากการ join L_1 กับ L_1 เข้าด้วยกันแล้วนำไปพิจารณาว่าแต่ละไอเทมเซตใน C_2 มีซับเซตอยู่ใน L_1 ทุกตัวหรือไม่ ถ้ามีไม่ครบทุกตัวก็จะทำการตัดออกจาก C_2 แล้วนำไอเทมเซตแต่ละตัวใน C_2 ไปสแกนหาค่าสนับสนุนเพื่อนำมาเปรียบเทียบว่ามีค่ามากกว่าหรือเท่ากับค่าสนับสนุนน้อยที่สุดหรือไม่ ไอเทมเซตที่ผ่านเกณฑ์จะปรากฏอยู่ใน L_2 และจะทำการหาค่า Large Itemsets โดยวนกลับไปทำงานเช่นเดียวกับขั้นตอนนี้ไปเรื่อยๆ จนไม่สามารถหาค่า Large Itemsets ได้อีก

การทำงานของอัลกอริทึมอะพิโอริเมื่อกฎความสัมพันธ์ที่มีค่าสนับสนุนและมีค่าความเชื่อมั่นที่ได้ผ่านค่าสนับสนุนน้อยที่สุด และค่าความเชื่อมั่นน้อยที่สุดตามกำหนด แต่ละกฎความสัมพันธ์ที่ผ่านเกณฑ์พิจารณาดังกล่าว สามารถได้เป็นกฎความสัมพันธ์ที่แข็งแกร่ง (Strong Association Rule)

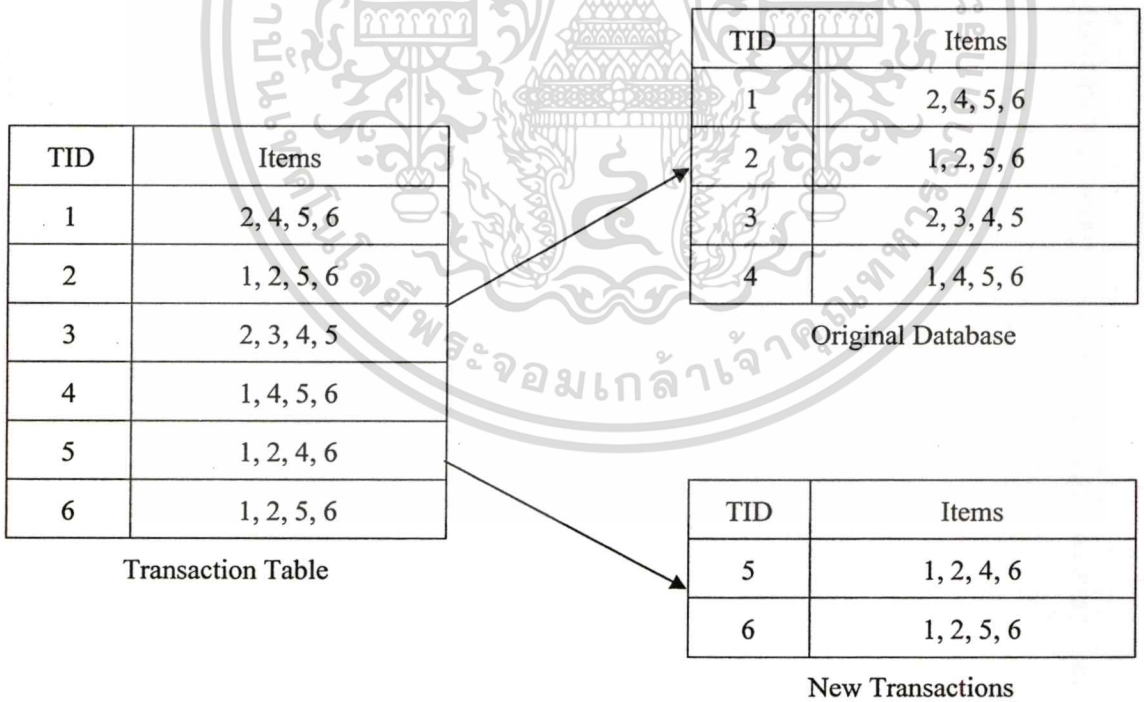
ข้อด้อยของอัลกอริทึมอะพิโอริ

1. เมื่อฐานข้อมูลมีการเปลี่ยนแปลงต้องทำการค้นหากฎความสัมพันธ์ใหม่ทั้งหมด และในการค้นหากฎความสัมพันธ์ใหม่แต่ละครั้ง จะต้องทำการสแกนข้อมูลทั้งฐานข้อมูล เพราะไม่ได้นำ Large Itemsets ที่เป็นความรู้เดิมซึ่งเคยได้ค้นหาไว้มาใช้ให้เกิดประโยชน์

2. จากปัญหาข้างต้นทำให้ต้องสูญเสียเวลาในการค้นหากฎความสัมพันธ์ใหม่ทั้งหมด ซึ่งอาจจะทำให้ต้องใช้เวลาในการค้นหากฎความสัมพันธ์เพิ่มมากขึ้น

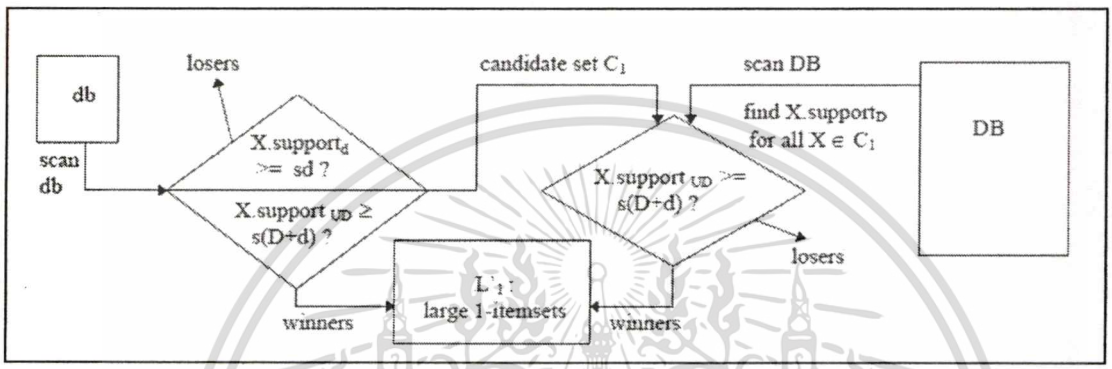
2.4 การค้นหากฎความสัมพันธ์ของการเพิ่มข้อมูล (Incremental Association Rule Discovery)

การค้นหากฎความสัมพันธ์ของการเพิ่มข้อมูล เป็นการค้นหากฎความสัมพันธ์เนื่องจากมีการเพิ่มขึ้นข้อมูลใหม่เข้าสู่ฐานข้อมูล ทำให้มีผลต่อการปรับปรุงค่าสนับสนุนไอเทมเซตของข้อมูล ซึ่งอาจมีผลทำให้กฎความสัมพันธ์เดิมที่เคยเป็นกฎความสัมพันธ์ที่ได้ก่อนมีการเพิ่มข้อมูลใหม่เข้าสู่ฐานข้อมูลเป็นกฎที่ไม่สามารถนำไปใช้งานได้ หรือเกิดการสร้างกฎความสัมพันธ์ขึ้นมาใหม่จากข้อมูลที่เพิ่มขึ้น ทำให้ต้องมีการค้นหากฎความสัมพันธ์ใหม่เพื่อให้กฎความสัมพันธ์ถูกต้องอยู่เสมอ รวมถึงการค้นหากฎความสัมพันธ์แต่ละครั้งจะต้องเริ่มต้นทำการค้นหา Large Itemsets ใหม่ทุกครั้ง เมื่อฐานข้อมูลที่ข้อมูลใหม่เพิ่มเข้ามา ทำให้เสียเวลาในการค้นหามากเพราะโดยทั่วไปแล้วฐานข้อมูลเก่ามีขนาดใหญ่ และฐานข้อมูลที่เพิ่มใหม่จะมีขนาดเล็กกว่า



รูปที่ 2.5 กฎความสัมพันธ์ของการเพิ่มข้อมูล

เทคนิคการค้นหากฎความสัมพันธ์ของการเพิ่มข้อมูล (Incremental Association Rule Discovery) ดังแสดงในรูปที่ 2.5 เป็นเทคนิคที่ช่วยรักษากฎความสัมพันธ์ที่ได้จากฐานข้อมูลเดิม และจัดการกับข้อมูลที่เพิ่มเข้ามาใหม่ เพื่อลดจำนวน Candidate Itemset ที่ได้จากการค้นหากฎความสัมพันธ์ ทำให้การค้นหากฎความสัมพันธ์มีประสิทธิภาพและลดเวลาในการค้นหาข้อมูล สามารถนำไปประยุกต์ใช้ในงานด้านต่างๆ เช่น แผนกลยุทธ์ทางการตลาด (Market strategies) การทำนายด้านการเงิน (Financial forecasts) เป็นต้น



รูปที่ 2.6 กระบวนการของอัลกอริทึม FUP สำหรับการค้นหา Large 1-itemset

2.4.1 การค้นหากฎความสัมพันธ์ด้วยอัลกอริทึมเอฟยูพี (FUP Algorithm)

อัลกอลิทึมเอฟยูพี (Fast Update Algorithm: FUP) เป็นอัลกอลิทึมสำหรับการค้นหากฎความสัมพันธ์เมื่อมีการเพิ่มข้อมูลใหม่เข้าสู่ฐานข้อมูล ทำให้ช่วยรักษากฎความสัมพันธ์ที่ได้จากฐานข้อมูลเดิม และเมื่อมีทรานแซคชั่นเพิ่มขึ้นในฐานข้อมูลจะเกิดการปรับปรุงกฎความสัมพันธ์ในฐานข้อมูล

FUP เป็นอัลกอริทึมที่พัฒนามาจาก Apriori อัลกอริทึมโดยมีหลักการทำงานคือ จัดการเก็บจำนวน Large Itemsets ที่ค้นพบในฐานข้อมูลเดิม จากนั้นทำการบันทึกและตรวจสอบทรานแซคชั่นใหม่ที่เพิ่มเข้ามาในฐานข้อมูล อัลกอริทึมจะสร้าง Candidate Set ขนาดเล็กเพื่อค้นหา New Large Itemsets และทำการปรับปรุงฐานข้อมูลโดยการนำข้อมูลในฐานข้อมูลเดิมมารวมเข้ากับข้อมูลใหม่ที่เพิ่มขึ้นเพื่อหา New Large Itemsets ทั้งหมดที่อยู่ในฐานข้อมูล

ลักษณะสำคัญของปัญหาในการปรับปรุงกฎความสัมพันธ์มีดังนี้

1. การค้นหา New Large Itemsets ใช้เวลาในการค้นหานาน
2. Large Itemsets ในฐานข้อมูลเดิม ที่เป็นกฎความสัมพันธ์แบบน่าเชื่อถือ (Strong Association) อาจจะกลายเป็นกฎความสัมพันธ์แบบไม่น่าเชื่อถือ (Weak Association) เมื่อฐานข้อมูลมีการปรับปรุง

3. Large Itemsets ในฐานข้อมูลเดิม ที่เป็นกฎความสัมพันธ์แบบไม่น่าเชื่อถือ (Weak Association) อาจจะกลายเป็นกฎความสัมพันธ์แบบน่าเชื่อถือ (Strong Association) เมื่อฐานข้อมูลมีการปรับปรุง
4. เมื่อมีการเพิ่มข้อมูลใหม่เข้ามาในฐานข้อมูลเดิม จะทำการค้นหา Candidate Itemset ทุกครั้งในการอ่านข้อมูล

ขั้นตอนการทำงานของ Fast Update Algorithm (FUP) แบ่งออกเป็น 2 ขั้นตอนดังนี้

2.4.1.1 ในการอ่านข้อมูลรอบแรก (Large 1-Itemsets)

ทำการอ่านข้อมูลครั้งแรกเพื่อลดปริมาณ Frequent Itemsets ที่ไม่เป็นไปตามเงื่อนไข (Loser) โดยทำการอ่านข้อมูลรอบแรกเพื่อหา Candidate Itemset จำนวนหนึ่ง รวมถึงค้นหาปริมาณของ Frequent Itemsets ที่เป็นไปตามเงื่อนไข (Winner) โดยมีการทำงานดังนี้

1. Itemset(X) ใน Large 1-Itemsets (L_1) ในฐานข้อมูลเดิม (Original Database) จะมี Itemset เป็นไปตามเงื่อนไข (winner) เมื่อฐานข้อมูลมีการเปลี่ยนแปลง (DB $\cup$ db) ก็ต่อเมื่อ $X.support_{UD} \geq s(D + d)$
2. Itemset(X) ใน Large 1-Itemsets ในทรานแซคชั่นที่เพิ่มขึ้น (Update Database) จะมี Itemset เป็นไปตามเงื่อนไข (Winner) เมื่อฐานข้อมูลมีการเปลี่ยนแปลง (DB $\cup$ db) ก็ต่อเมื่อ $X.support_d \geq s \times d$

ในทฤษฎีดังกล่าวจะทำการค้นหา Large 1-Itemset (L'_1) เมื่อทำการปรับปรุงฐานข้อมูล (DB $\cup$ db) โดยทำตามขั้นตอนดังนี้

1. ทำการรวมข้อมูลทรานแซคชั่นที่เพิ่มขึ้น (db) ร่วมกับ Large Itemset (L_1) เพื่อหาค่า Support Count ($X.support_{UD}$) จากนั้นทำการตรวจสอบเงื่อนไข $X.support_{UD} > s(D+d)$ โดยที่ $X \in L_1$ (ตามทฤษฎีที่ 1) ถ้า Itemsets ดังกล่าวเป็นไปตามเงื่อนไขจะถูกแสดงใน Large 1-Itemsets (L'_1) แต่ถ้าไม่เป็นไปตามเงื่อนไข จะถูกตัดออกจาก Large 1-Itemsets (L_1)
2. ทำการหา Candidate Itemset (C_1) ใน Large Itemsets (L_1) จากนั้นทำการข้อมูลในฐานข้อมูล (db) เพื่อหาค่า Support Count ($X.support_d$) ต่อมาทำการตรวจสอบเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_1$ (ตามทฤษฎีที่ 2) ถ้า Itemsets ดังกล่าวเป็นไปตามเงื่อนไขจะถูกแสดงอยู่ใน Candidate Itemset (C_1) แต่ถ้าไม่เป็นไปตามเงื่อนไข จะถูกตัดออกจาก Candidate Itemset (C_1)
3. ทำการอ่านข้อมูลในฐานข้อมูล (DB) เพื่อนำค่า Support Count มารวมเข้ากับค่า Support Count ของ Candidate Itemsets (C_1) เพื่อทำการปรับปรุงฐานข้อมูล จากนั้นทำการตรวจสอบเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ (ตามทฤษฎีที่ 1) ถ้า

Itemsets ดังกล่าวเป็นไปตามเงื่อนไข Itemsets ที่อยู่ใน Candidate Itemset (C_1) และ Large Itemsets (L_1) จะถูกรวมเข้าด้วยกันเพื่อทำการปรับปรุงข้อมูลและแสดงค่าใน New Large Itemset (L'_1) แต่ถ้าไม่เป็นไปตามเงื่อนไขจะไม่ถูกแสดงค่าใน New Large Itemset (L'_1)

Original Database (DB)
D = 1,000
s = 3%
$L_1 = \{I_1, I_2\}$
I_1.support = 32
I_2.support = 31
I_3.support = 28
I_4 is not in L_1

New Transactions (db)
d = 100
s = 3%
I_1.support = 4
I_2.support = 1
I_3.support = 6
I_4.support = 2

รูปที่ 2.7 ตัวอย่างฐานข้อมูลในการอ่านข้อมูลรอบแรก

จากรูปที่ 2.7 เป็นตัวอย่างฐานข้อมูลที่นำมาใช้ในการปรับปรุงกฎความสัมพันธ์ในการอ่านข้อมูลรอบแรก ประกอบด้วย DB คือฐานข้อมูลเดิมที่ผ่านการประมวลผลหากฎความสัมพันธ์มาแล้ว โดยมีข้อมูลทั้งหมด 1,000 ทรานแซคชั่น ประกอบด้วย Itemsets = $\{I_1, I_2, I_3, I_4\}$ และมี Large Itemsets (L_1) = $\{I_1, I_2\}$ และ db คือข้อมูลที่เพิ่มขึ้น ซึ่งประกอบด้วย Itemsets = $\{I_1, I_2, I_3, I_4\}$ มีข้อมูลทั้งหมด 100 ทรานแซคชั่น จากนั้นนำข้อมูลทั้งสองส่วนมาปรับปรุงร่วมกันเพื่อหากฎความสัมพันธ์ในการอ่านข้อมูลรอบแรกในฐานข้อมูลทั้งหมด เพื่อหา Itemset ที่เป็นไปตามเงื่อนไขตามค่า Support ที่ถูกกำหนดไว้เท่ากับ 3% ของข้อมูลทั้งหมด โดยมีขั้นตอนการทำงานดังนี้

ขั้นตอนที่ 1 ทำการอ่านข้อมูลใน db เพื่อนับจำนวนข้อมูลที่เกิดขึ้นของแต่ละ Large Itemsets (L_1) = $\{I_1, I_2\}$ ใน DB เพื่อปรับปรุงค่า Support ของแต่ละ Large Itemsets (L_1) ในฐานข้อมูลทั้งหมด เพื่อดูว่ากฎความสัมพันธ์เดิมยังเป็นไปตามเงื่อนไขหรือไม่ เมื่อฐานข้อมูลมีการเปลี่ยนแปลง โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$

$I_1.support_{UD}$ คือค่า Support รวมทั้งหมดซึ่งจะเท่ากับ 32 + 4 = 36 และสามารถคำนวณหาค่าของ min_sup รวมได้ดังนี้ 3% × 1,100 = 33 เพราะฉะนั้นจะเห็นว่าค่าของ $I_1.support_{UD}$ เท่ากับ 36 นั้นมีค่ามากกว่าค่าของ min_sup รวมคือ 33 ดังนั้น $I_1 \in L'_1$

$I_2.support_{UD}$ = 32 มีค่ามากกว่าค่าของ min_sup รวมคือ 33 ดังนั้น $I_2 \notin L'_1$ จึงถูกตัดออกจาก Large 1-Itemsets (L_1)

ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น ที่ไม่ได้อยู่ใน Large Itemsets (L_1) = {I_3, I_4} เพื่อหา Candidate Itemset (C_1) ไว้ใช้ในการหากฎความสัมพันธ์ใหม่ โดยทำการหาค่า Support ของ Itemsets ใน Candidate Itemsets (C_1) และทำการตรวจสอบตามเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_1$

ซึ่งสามารถคำนวณหาค่า min_sup ของฐานข้อมูลใหม่โดยการนำค่า support factor = 3% มาคูณกับจำนวนทรานแซคชั่นทั้งหมดของฐานข้อมูลใหม่คือ 100 จะได้ค่า min_sup เท่ากับ 3 แล้วนำค่า min_sup ที่คำนวณได้นี้ไปเปรียบเทียบกับค่า support ของ Item แต่ละตัว

$I_3.support_d$ = 6 มีค่ามากกว่าค่า min_sup ดังนั้น $I_3 \in C_1$

$I_4.support_d$ = 2 มีค่าน้อยกว่าค่า min_sup ดังนั้น $I_4 \notin C_1$ จึงถูกตัดออกจาก Candidate Itemsets (C_1)

ขั้นตอนที่ 3 ทำการอ่านข้อมูลใน DB เพื่อนับจำนวนข้อมูลของ Itemsets ใน Candidate Itemsets (C_1) = {I_3} มาปรับปรุงร่วมกับค่า Support ใน db เพื่อหากฎความสัมพันธ์ใหม่เมื่อฐานข้อมูลมีการเปลี่ยนแปลง โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ เพื่อทำการค้นหา New Large Itemsets (L'_1)

$I_3.support_{UD}$ = 34 มีค่ามากกว่าค่าของ min_sup รวมซึ่งมีค่าเท่ากับ 33 ดังนั้น $I_3 \in L'_1$

$\therefore$ ดังนั้นผลลัพธ์ของกฎความสัมพันธ์ใหม่ที่ได้จากการอ่านข้อมูลครั้งแรกจะได้ L'_1 = {I_1, I_3}

2.4.1.2 ในการอ่านข้อมูลรอบที่สองและรอบถัดไป (Large k-Itemsets เมื่อ $k > 1$)

ทำการลบ Itemsets ที่ไม่เป็นไปตามเงื่อนไข (Loser) ออกจากฐานข้อมูลและทำการตัด Itemsets ที่ไม่เป็นไปตามเงื่อนไขใน Candidate Itemset ออก รวมถึงทำการค้นหา Itemsets ที่เป็นไปตามเงื่อนไข (winner) ในฐานข้อมูล โดยมีการทำงานดังนี้

1. ถ้า k-Itemset ไม่ได้เป็นไปตามเงื่อนไข (Loser) ในฐานข้อมูลที่มีการเปลี่ยนแปลงในการอ่านข้อมูลครั้งที่ (k-1)-th (Itemset อยู่ใน L_{k-1} แต่ไม่ได้อยู่ใน L'_{k-1}) Large k-Itemset ใน L_k ที่ประกอบด้วย Itemset ดังกล่าว จะไม่สามารถเป็นไปตามเงื่อนไข (Winner) ในฐานข้อมูลที่มีการเปลี่ยนแปลงในการอ่านข้อมูลครั้งที่ k-th (Large k-Itemset ไม่ได้อยู่ใน L_k และ L'_k)

2. k-Itemset ในฐานข้อมูลเดิม (L_k) ไม่ได้เป็นไปตามเงื่อนไข (Loser) ในฐานข้อมูลที่มีการเปลี่ยนแปลง (Itemset อยู่ใน L_k แต่ไม่ได้อยู่ใน L'_k) ก็ต่อเมื่อ k-Itemset. $support_{UD} < s(D + d)$

Input: (1) *DB*: the original database (with its size, i.e., the total number of transaction, equal to *D*); (2) L_k: the set of all large *k*-itemsets in *DB*, where $k = 1, \ldots, r$; (3) *db*: an increment database (with its size equal to *d*); and (4) *s*: the minimum support threshold.

Output: L': The set of all large itemsets in $DB \cup db$.

Method: The 1st iteration: /* find L'_1, the set of all large 1-itemsets in $DB \cup db$ */

$W = L_1$; $C = \phi$; $L'_1 = \phi$; $P = \phi$;

/* *W*: winners, *C*: candidate sets, L'_1: initialized, *P*: for optimization */

for_all $T \in db$ do /* scan *db* */

for_all 1-itemset $X \subseteq T$ do {

if $X \in W$ then $X.support_d$++;

else {

if $X \notin C$ then { $C = C \cup \{X\}$; $X.support_d = 0$; }

/*init the support cont and add *X* into *C* */

$X.supprt_d$++; } };

for_all $X \in W$ do /* put winners into L'_1*/

if $X.support_{UD} \geq s \times (D + d)$

then $L'_1 = L'_1 \cup \{X\}$;

for_all $X \in C$ do /*prune candidate sets in *C*/

if $X.support_d < s \times d$

then { $C = C - \{X\}$; $P = P \cup \{X\}$; } /* *P* will be used for optimization */

for_all $T \in DB$ do /* scan *DB* */

for_all 1-itemset $X \subseteq T$ do {

if $X \in C$ then $X.support_D$++;

if $X \in P$ then removes *X* from *T*; /* Transaction *T* is reduced */

};

for_all $X \in C$ do /* put winners into L'_1 */

if $X.support_{UD} \geq s \times (D + d)$ then $L'_1 = L'_1 \cup \{X\}$;

return L'_1. /* end of the 1st iteration */

รูปที่ 2.8 เอฟยูพีอัลกอริทึมสำหรับหา Large 1-itemset

The k-th iteration: /* for $k = 2$ or larger, repeat this program fragment to find L'_k, the set of all large k-itemsets in the updated database, until either L'_k returned is empty or $db = \phi$ */

$W = L_k$; $L'_k = \phi$; /* W: winners; L'_k initialized */

C = apriori_gen1(L'_{k-1}) - L_k;

/* the size-k candidate sets */

for_all k-itemset $X \in W$ do /* prune off losers in W */

for_all (k-1)-itemset $Y \in L_{k-1} - L'_{k-1}$ do

if $Y \subseteq X$ then { $W = W - \{X\}$; break; }

for_all $T \in db$ do { /* scan db */

for_all $X \in$ Subset(W, T) do $X.support_d$++;

/* Subset(W,T) returns all the sets in W contained in T */

for_all $X \in$ Subset(C, T) do $X.support_d$++; /* find support of all $X \in C$ */

Reduce_db (T);

/* Some items in transactions in db can be removed, discussed in next section */

}

for_all $X \in W$ do /* put the winners from W into L'_k */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

for_all $X \in C$ do /* prune candidate sets in C */

if $X.support_d < s \times d$ then $C = C - \{X\}$;

for_all $T \in DB$ do { /* scan DB */

for_all $X \in$ Subset(C, T) do $X.support_D$++;

Reduce_Db(T); }

/* Some items in transactions in DB can be removed, discussed in next section */

for_all $X \in C$ do /* put the winners from C into L'_k */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

return L'_k. /* The end of the k-th iteration */

รูปที่ 2.9 เอฟยูพีอัลกอริทึมสำหรับหาตั้งแต่ Large 2-itemsets

3. k-Itemset ที่ไม่ได้อยู่ในฐานข้อมูลเดิม (L_k) สามารถเป็น k-Itemset ที่เป็นไปตามเงื่อนไข (Winner) ในฐานข้อมูลที่มีการเปลี่ยนแปลง (Itemset ไม่ได้อยู่ใน L_k แต่อยู่ใน L'_k) ก็ต่อเมื่อ k-Itemset.$support_d \geq s \times d$

ในทฤษฎีดังกล่าวจะทำการค้นหา Large 2-Itemset (L'_2) เมื่อทำการปรับปรุงฐานข้อมูล (DB $\cup$ db) โดยทำตามขั้นตอนดังนี้

1. การหา k-Itemset ที่ไม่ได้เป็นไปตามเงื่อนไข (Loser) ในฐานข้อมูล (db) มีการทำงาน 2 ขั้นตอน
   1.1 ถ้า k-Itemset $\{X_1, \ldots, X_n\}$ ใน Large 2-Itemset (L_2) ประกอบด้วย Itemset $\{Y_1, \ldots, Y_{k-1}\}$ ที่ไม่ได้เป็นไปตามเงื่อนไข (Loser) ในการอ่านข้อมูลครั้งแรกดังนั้นถ้า $X \in L_2$, $Y \in L_1\text{-}L'_1$ (ตามทฤษฎีบทที่ 3) เป็นไปตามเงื่อนไข k-Itemset ดังกล่าวจะไม่ปรากฏใน Large 2-Itemset (L_2)
   1.2 ทำการอ่านข้อมูลในฐานข้อมูล (db) เพื่อหาค่า Support Count ($X.support_d$) จากนั้นทำการตรวจสอบเงื่อนไข ถ้า $X.support_d \geq s \times d$ โดยที่ $X \in L_2$ เป็นไปตามเงื่อนไข k-Itemset ดังกล่าวจะถูกแสดงใน Large 2-Itemset (L_2)
2. ทำการหา Candidate Itemset (C_2) จาก New Large Itemsets (L'_1) จากนั้นทำการอ่านข้อมูลในฐานข้อมูล (db) เพื่อหาค่า Support Count ($X.support_d$) และทำการตรวจสอบเงื่อนไข ถ้า $X.support_d \geq s \times d$ โดยที่ $X \in C_2$ จะถือว่าเป็นไปตามเงื่อนไข k-Itemset ดังกล่าวจะถูกแสดงใน Candidate Itemset (C_2) เพื่อใช้ค้นหา New Large Itemsets (L'_2) แต่ถ้าไม่เป็นไปตามเงื่อนไขจะถูกตัดออกจาก Candidate Itemset (C_2)

Original Database (DB)
D = 1,000
s = 3%
$L_1 = \{I_1, I_2, I_3\}$
$L_2 = \{I_1I_2, I_2I_3\}$
$L'_1 = \{I_1, I_2, I_4\}$
I_1I_2.support = 50
I_2I_3.support = 31
I_4 is not in L_1

New Transactions (db)
d = 100
s = 3%
I_1I_2.support = 3
I_1I_4.support = 5
I_2I_4.support = 2

รูปที่ 2.10 ตัวอย่างฐานข้อมูลในการอ่านข้อมูลรอบที่สองและรอบถัดไป

3. ทำการอ่านข้อมูลเพื่อทำการปรับปรุงฐานข้อมูล (DB) โดยนำค่า Support Count ในฐานข้อมูลเดิม ($X.support_D$) มารวมกับค่า Support Count ของ Candidate Itemset (C_2) จากนั้นทำการตรวจสอบเงื่อนไข ถ้า $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in C_2$ เป็นไปตามเงื่อนไข Itemsets ดังกล่าวที่อยู่ใน Candidate Itemset (C_2) และ Large Itemsets (L_2) จะถูกรวมเข้าด้วยกันเพื่อทำการปรับปรุงข้อมูลและแสดงค่าใน New Itemset (L'_2) แต่ถ้าไม่เป็นไปตามเงื่อนไขจะไม่ถูกแสดงค่าใน New Large Itemset (L'_2)

จากรูปที่ 2.10 เป็นตัวอย่างฐานข้อมูลที่นำมาใช้ในการปรับปรุงกฎความสัมพันธ์ในการอ่านข้อมูลรอบที่สอง ประกอบด้วย DB คือฐานข้อมูลเดิมที่ผ่านการประมวลผลเพื่อหากฎความสัมพันธ์มาแล้ว ซึ่งมีข้อมูลทั้งหมด 1,000 ทรานแซคชั่น ประกอบด้วย Itemsets = $\{I_1, I_2, I_3, I_4\}$ โดยมี Large Itemsets (L_1) = $\{I_1, I_2, I_3\}$ และมี Large Itemsets (L_2) = $\{I_1I_2, I_2I_3\}$ และ db คือข้อมูลที่เพิ่มขึ้น มีข้อมูลทั้งหมด 100 ทรานแซคชั่น จากนั้นนำข้อมูลที่เพิ่มขึ้น (db) มาปรับปรุงร่วมกับฐานข้อมูลเดิม เพื่อหากฎความสัมพันธ์ในการอ่านข้อมูลรอบสอง Large Itemsets (L'_2) ในฐานข้อมูลทั้งหมด โดยมี Large Itemsets (L'_1) ของฐานข้อมูลทั้งหมด = $\{I_1, I_2, I_4\}$ จากนั้นทำการหา Large Itemsets (L'_2) ที่เป็นไปตามเงื่อนไขตามค่า Support ที่ถูกกำหนด = 3% ของข้อมูลทั้งหมด โดยมีขั้นตอนการทำงานดังนี้

ขั้นตอนที่ 1 ทำการลบ Itemset ใน Large Itemsets (L_2) = $\{I_1I_2, I_2I_3\}$ ที่ไม่ได้เป็นไปตามเงื่อนไข $X \in L_2$, $Y \in L_1\text{-}L'_1$ จะเห็นว่า $I_3 \in L_1\text{-}L'_1$ ดังนั้น I_2I_3 จึงถูกลบออกจาก Large Itemsets (L_2) จากนั้นทำการอ่านข้อมูลใน db เพื่อทำการปรับปรุงค่า Support ($X.support_{UD}$) ของ Large Itemsets (L_2) ในฐานข้อมูลทั้งหมด เพื่อดูว่ากฎความสัมพันธ์เดิมยังเป็นไปตามเงื่อนไขหรือไม่เมื่อฐานข้อมูลมีการเปลี่ยนแปลง ซึ่งเราจะทำการตรวจสอบเงื่อนไข $X.support_d \geq s(D + d)$ โดยที่ $X \in L_2$

$I_1I_2.support_{UD} = 53$ มีค่ามากกว่าค่า min_sup รวมซึ่งมีค่าเท่ากับ 33 ดังนั้น $I_1I_2 \in L'_2$

ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น โดยนำ Large Itemsets (L'_1) ของฐานข้อมูลทั้งหมด = $\{I_1, I_2, I_4\}$ มาทำการ join ข้อมูลเพื่อทำการหา Candidate Itemset (C_2) โดยที่ Itemsets ที่ได้ $\notin L_2$ ดังนั้น $C_2 = \{I_1I_2, I_1I_4, I_2I_4\}$ แต่ $I_1I_2 \in L_2$ จึงไม่ถูกแสดงใน C_2 ดังนั้น $C_2 = \{I_1I_4, I_2I_4\}$ จากนั้นทำการอ่านข้อมูลในฐานข้อมูล (db) เพื่อหาค่า Support ($X.support_d$) ของ Itemsets ใน Candidate Itemset (C_2) ซึ่งเราจะทำการตรวจสอบเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_2$

$I_1I_4.support_d = 5$ มีค่ามากกว่า min_sup ของฐานข้อมูลใหม่ที่คำนวณแล้วข้างต้นมีค่า = 3 ดังนั้น $I_1I_4 \in C_2$

$I_2I_4.support_d = 2$ มีค่าน้อยกว่าค่า min_sup ของฐานข้อมูลใหม่ซึ่งมีค่า = 3 ดังนั้น $I_2I_4 \notin C_2$ จึงถูกลบออกจาก Candidate Itemset (C_2)

ขั้นตอนที่ 3 ทำการอ่านข้อมูลใน DB เพื่อนับจำนวนข้อมูลของ Itemsets ใน Candidate Itemset (C_2) = $\{I_1I_4\}$ มาปรับปรุงร่วมกับค่า Support ใน db เพื่อหาค่า Support Count ($X.support_{UD}$) ของ Itemsets ที่อยู่ใน Candidate Itemset (C_2) ของฐานข้อมูลทั้งหมด และนำข้อมูลที่ได้ไปหากฎความสัมพันธ์เมื่อฐานข้อมูลมีการเปลี่ยนแปลง จากนั้นทำการตรวจสอบเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in C_2$ (กำหนดให้ $I_1I_4.support_D = 30$)

$I_1I_4.support_{UD} = 35$ มีค่ามากกว่าค่า min_sup รวมซึ่งมีค่า = 33 ดังนั้น $I_1I_4 \in L'_2$

$\therefore$ ดังนั้นผลลัพธ์ที่ได้จากการอ่านข้อมูลรอบที่สองจะได้ $L'_2 = \{I_1I_2, I_1I_4\}$

ข้อเด่นของอัลกอริทึม FUP

1. มีการนำ Large Itemsets ที่เคยค้นหาจากฐานข้อมูลเดิมนำมาใช้ให้เกิดประโยชน์ เพื่อลดการค้นหาซ้ำในฐานข้อมูลเดิม

2. สามารถลดจำนวน Candidate Itemsets ที่ได้จากฐานข้อมูลเพิ่มใหม่ เพื่อนำไปใช้ค้นหาในฐานข้อมูลเดิมทำให้มีการค้นหาน้อยลง โดย Candidate Itemsets จะทำการค้นหาเฉพาะไอเทมเซตที่ไม่เคยมีมาก่อนใน Large Itemsets ในฐานข้อมูลเดิม

ข้อด้อยของอัลกอริทึม FUP

ใช้เวลาในการประมวลผลนาน

2.5 การค้นหากฎความสัมพันธ์ด้วยอัลกอริทึมดีเอชพี (DHP Algorithm)

จากอัลกอริทึมที่ได้กล่าวมาแล้วข้างต้น จะเห็นว่าการค้นหากฎความสัมพันธ์เป็นงานที่ใช้ทรัพยากรในการคำนวณที่สูงมาก ดังนั้นเราจะนำเสนอ Direct Hashing and Pruning Algorithm (DHP) ที่จะนำมาใช้เพื่อช่วยเพิ่มประสิทธิภาพในการหา Large Itemsets ของการค้นหากฎความสัมพันธ์

ทั้ง Apriori และ DHP Algorithm สามารถสร้าง Candidate k+1-Itemsets (C_{k+1}) ได้จากการนำ Large k-Itemsets (L_k) มา join กับ L_k ($L_k * L_k$ หรือ $L_k \times L_k$) จากนั้นก็จะทำการสแกนฐานข้อมูลเพื่อนับค่า Support factor ของแต่ละ Itemset ที่อยู่ใน C_{k+1} โดยจะนำ Itemset ที่มีค่า Support factor ที่มากกว่าหรือเท่ากับ Min_Sup มาสร้างเป็น L_{k+1}

```
/* Part 1 */
s = a minimum support;
set all the buckets of H2 to zero;  /* hash table */
forall transaction t ∈ D do begin
    insert and count 1-Items occurrences in a hash tree;
    forall 2-Subsets x of t do
        H2[h2(x)] ++;
end
L1 = {c|c.count ≥ s, c is in a leaf node of the hash tree};
```

รูปที่ 2.11 ส่วนที่ 1 ของ Direct Hashing and Pruning Algorithm (DHP)

```
/* Part 2 */
k = 2;
Dk = D;              /* database for large k-Itemsets */
While (|{x|Hk[x] ≥ s}| ≥ LARGE) {
    /* make a hash table */
    gen_candidate(Lk-1, Hk, Ck);
    set all the buckets of Hk+1 to zero;
    Dk+1 = ϕ;
    forall transactions t ∈ Dk do begin
        count_support(t, Ck, k, Î);          /* Î ⊆ t */
        if (|Î| > k) then do begin
            make_hasht(Î, Hk, k, Hk+1, Ï);
            if (|Ï| > k) then Dk+1 = Dk+1 ∪ {Ï};
        end
    end
    Lk = {c ∈ Ck|c.count ≥ s};
    k ++;
}
```

รูปที่ 2.12 ส่วนที่ 2 ของ Direct Hashing and Pruning Algorithm (DHP)

```
/* Part 3 */
gen_candidate(L_{k-1}, H_k, C_k);
While (|C_k| > 0) {
    D_{k+1} = ∅;
    forall transactions t ∈ D_k do begin
        count_support(t, C_k, k, Î);          /* Î ⊆ t */
        if (|Î| > k) then D_{k+1} = D_{k+1} ∪ {Î};
    end
    L_k = {c ∈ C_k | c.count ≥ s};
    if (|D_{k+1}| = 0) then break;
    C_{k+1} = apriori_gen(L_k);
    k ++;
}
```

รูปที่ 2.13 ส่วนที่ 3 ของ Direct Hashing and Pruning Algorithm (DHP)

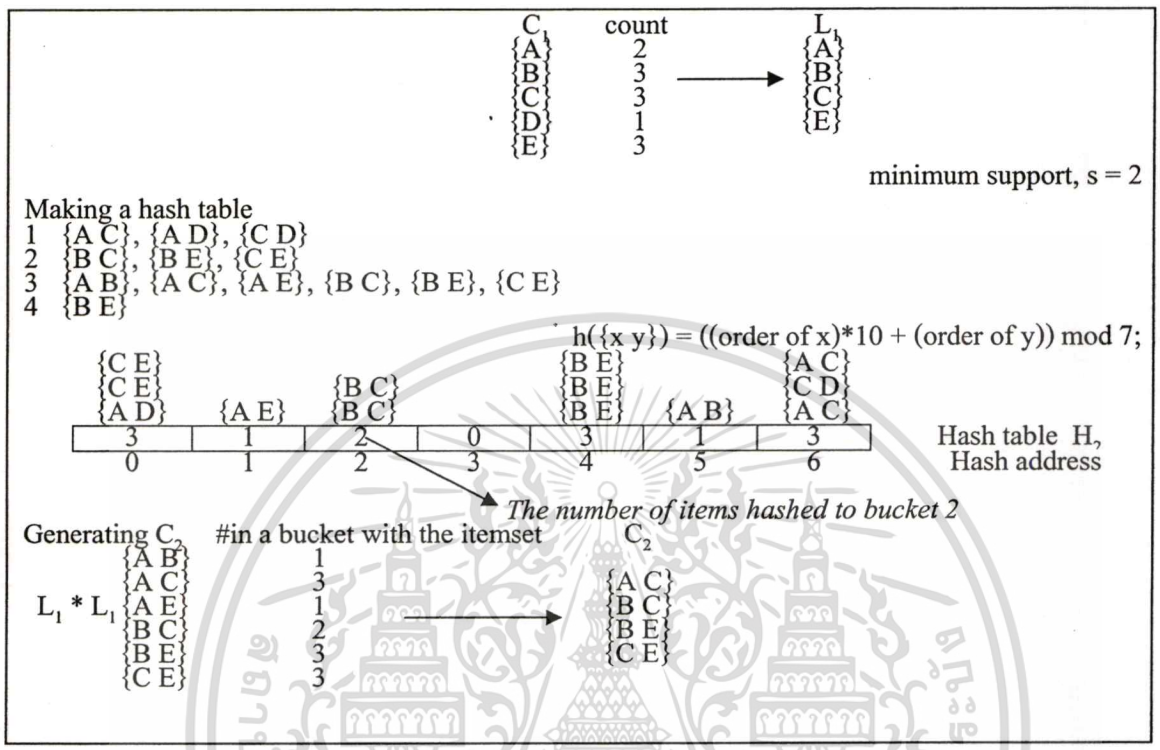
Database D

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

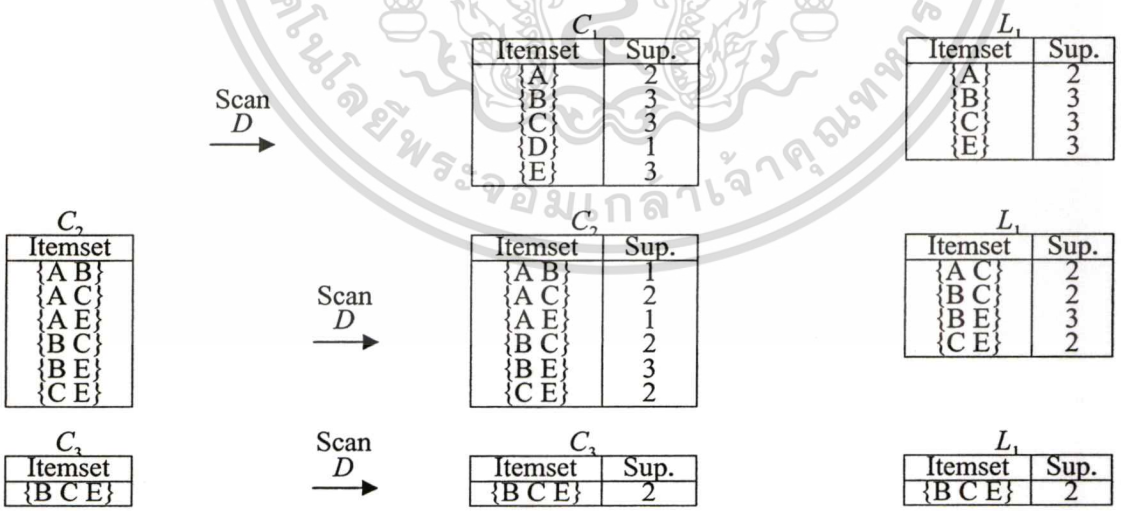
รูปที่ 2.14 ตัวอย่างฐานข้อมูลที่นำมาใช้สำหรับการทำ Mining Association Rule

ต่อไปจะเป็นการอธิบายลักษณะการทำงานของ DHP Algorithm โดยในขั้นตอนแรก DHP Algorithm จะทำการสแกนฐานข้อมูลรอบแรกเพื่อนับค่า Support Factor ของ 1-Itemsets และก็จะสร้างเซตย่อยของ 2-Itemsets ทั้งหมดที่เป็นไปได้ (ดูรูปที่ 2.15 ประกอบความเข้าใจ) จากนั้น DHP Algorithm ก็จะนำเซตย่อยของ 2-Itemses มาผ่าน Hash Function ทีละชุดเพื่อหา Hash Address แล้ว

ก็จะบวก 1 เพิ่มเข้าไปใน Bucket ของ Hash Table จากนั้นก็จะสร้าง L_1 จากค่า Support Factor ของ 1-Itemsets ที่มีค่ามากกว่าหรือเท่ากับ Min_Sup



รูปที่ 2.15 ตัวอย่างของ Hash Table และการสร้าง C_2



รูปที่ 2.16 สร้าง Candidate Itemsets และ Large Itemsets โดย Apriori Algorithm

ในขั้นตอนถัดมา DHP Algorithm ก็จะสร้าง C_k จากการนำ L_{k-1} มา join กันแล้วใช้ Hash Table ที่สร้างไว้ในรอบก่อนหน้านี้ช่วยในการกำหนดว่า Itemsets ชุดใดควรจะนำไปสร้างเป็น C_k

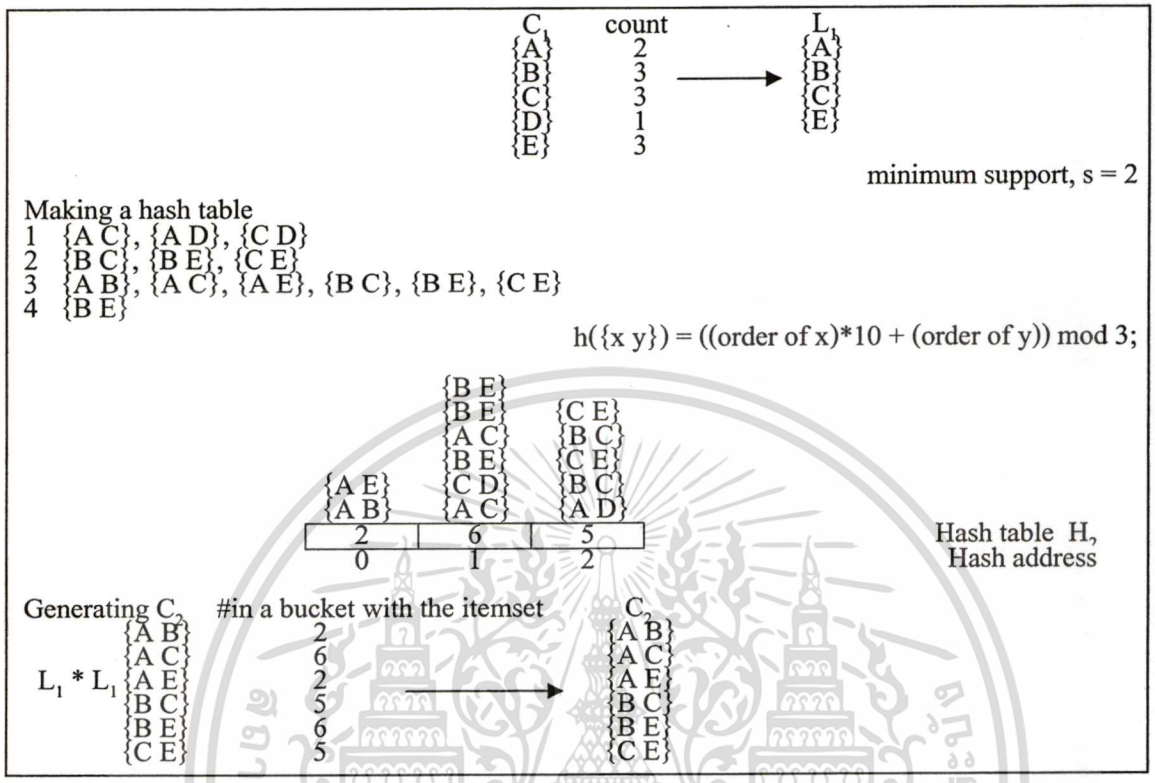
จากนั้นก็จะทำการสแกนฐานข้อมูลเพื่อหาว่ามี Itemsets ใดใน C_k ที่มีค่า Support Factor มากกว่าหรือเท่ากับ Min_Sup เพื่อนำ Itemsets เหล่านั้นมาสร้างเป็น L_k และสร้าง Hash Table ที่จะนำไปใช้ในการสร้าง C_k ในรอบถัดไป โดยขั้นตอนนี้จะทำงานวนลูปไปจนกระทั่งไม่สามารถสร้าง L_k ได้อีกแล้ว

จะเห็นว่า DHP Algorithm จะใช้เทคนิคของ Hashing มาช่วยถอน Itemsets ที่ไม่มีความจำเป็นสำหรับการสร้าง C_k ในรอบถัดไปออก ซึ่งจะทำให้ขนาดของ C_k ที่ได้จากการใช้ DHP Algorithm มีขนาดเล็กเมื่อเปรียบเทียบกับขนาดของ C_k ที่ได้จากการใช้ Apriori โดยสามารถสังเกตได้จากรูปที่ 2.15 และรูปที่ 2.16 จะพบว่า C_2 ที่ได้จากการใช้ DHP Algorithm จะมี Itemsets ทั้งหมด 4 ชุด คือ {A C}, {B C}, {B E} และ {C E} แต่ C_2 ที่ได้จากการใช้ Apriori จะมี Itemsets ทั้งหมด 6 ชุด คือ {A B}, {A C}, {A E}, {B C}, {B E} และ {C E} ดังนั้น DHP Algorithm จะมีประสิทธิภาพในการสร้าง L_k ที่ดีกว่า Apriori เพราะ Overhead ของการสแกนฐานข้อมูลเพื่อนับค่า Support Factor ของแต่ละ Itemset ที่อยู่ใน C_k จะลดลง

2.5.1 การกำหนดขนาดของตารางแฮชให้เหมาะสม

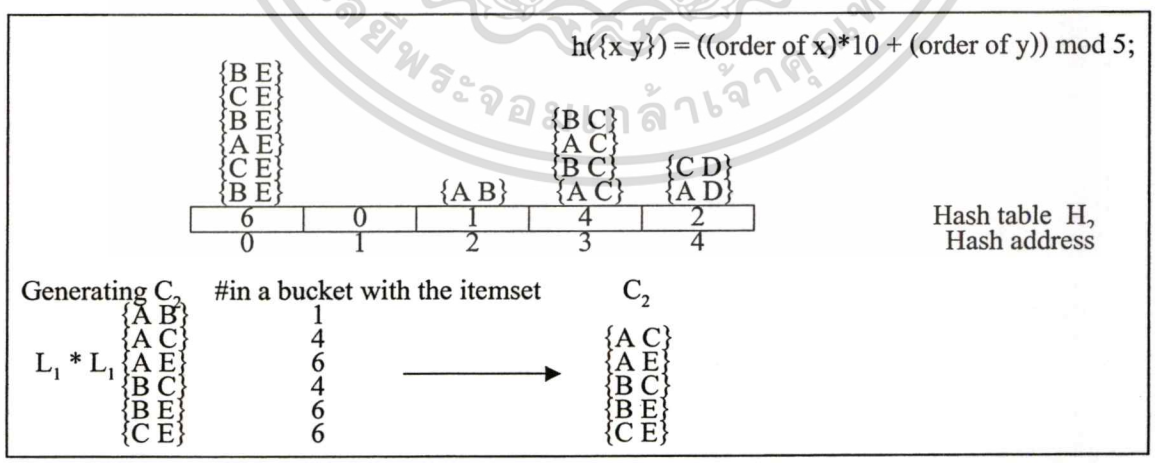
เนื่องจากขนาดของตารางแฮชอาจจะส่งผลต่อเวลา และจำนวนของ Candidate Itemsets ในการทดลอง ซึ่งจะนำไปสู่การหาค่า Large Itemsets ที่อาจผิดพลาดได้ เพราะตามคุณสมบัติของการสร้างตารางแฮชก็เพื่อช่วยลดขนาดของ Candidate 2-Itemsets ซึ่งจะมีขนาดใหญ่มาก เพื่อลดเวลาในการสแกนหาค่า Support Factor ของ Itemsets แต่ละตัว แต่ถ้าเรากำหนดขนาดของตารางแฮชไม่เหมาะสมอาจจะทำให้การสร้างตารางแฮชถือว่าเป็นการเพิ่มเวลาให้กับการประมวลไปโดยไม่เกิดประโยชน์ใดๆ เลยก็ได้ ซึ่งเราจะขอยกตัวอย่างในการอธิบายจากตัวอย่างการสร้างตารางแฮชในรูปที่ 2.15 ตามตัวอย่างเราใช้ขนาดของตารางแฮชเท่ากับ 7 แต่เราจะแสดงให้เห็นว่าถ้าเราเปลี่ยนขนาดของตารางแฮชเป็นค่าอื่นๆ จะส่งผลกระทบต่อการลดขนาดของ Candidate Itemsets อย่างไรบ้าง เพื่อนำไปสู่การวิเคราะห์เพื่อหาค่าของขนาดตารางแฮชที่เหมาะสมในการทดลอง โดยเราจะกำหนดให้มีขนาดของตารางมีค่าเท่ากับ 3, 5, 9 และ 10 ตามลำดับ

- ขนาดของตารางแฮชเท่ากับ 3 เราจะสร้างตารางแฮชได้ดังนี้



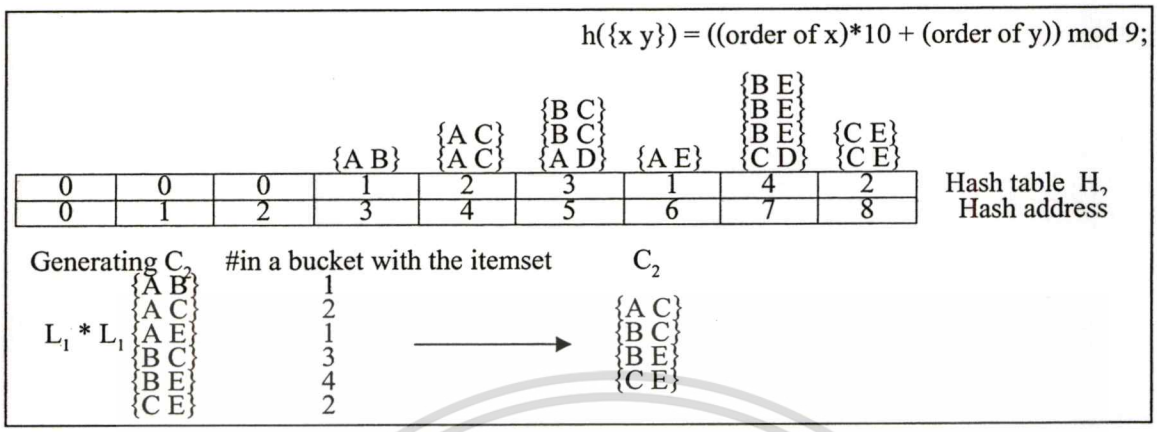
รูปที่ 2.17 ตัวอย่างการสร้างตารางแฮชที่มีขนาดเท่ากับ 3

- ขนาดของตารางแฮชเท่ากับ 5 เราจะสร้างตารางแฮชได้ดังนี้



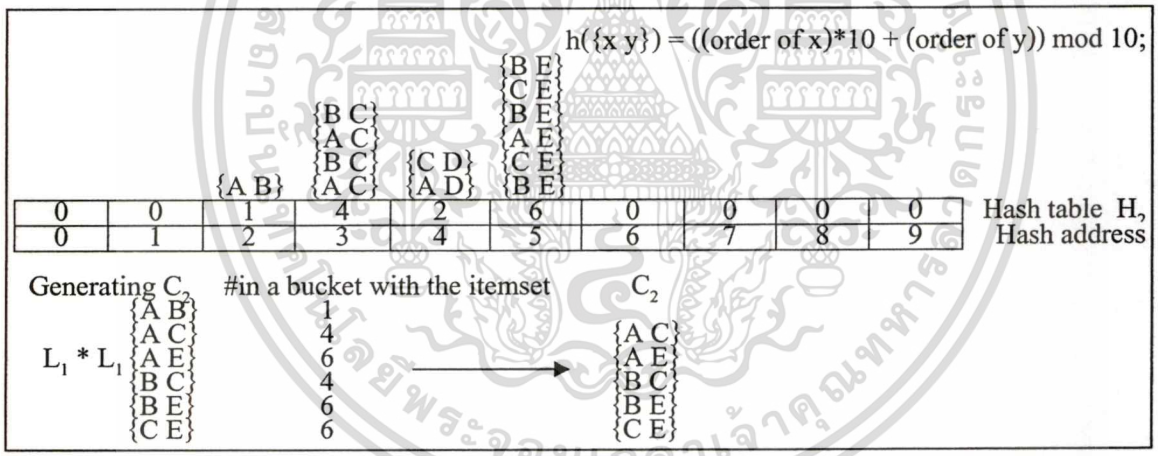
รูปที่ 2.18 ตัวอย่างการสร้างตารางแฮชที่มีขนาดเท่ากับ 5

- ขนาดของตารางแฮชเท่ากับ 9 เราจะสร้างตารางแฮชได้ดังนี้



รูปที่ 2.19 ตัวอย่างการสร้างตารางแฮชที่มีขนาดเท่ากับ 9

- ขนาดของตารางแฮชเท่ากับ 10 เราจะสร้างตารางแฮชได้ดังนี้



รูปที่ 2.20 ตัวอย่างการสร้างตารางแฮชที่มีขนาดเท่ากับ 10

จากรูปที่ 2.17, 2.18, 2.19 และ 2.20 นั้นจะเห็นว่าขนาดของตารางแฮชมีผลต่อการลดจำนวนของ Candidate Itemsets ถ้าเรากำหนดค่าขนาดของตารางแฮชให้มีขนาดเล็กเกินไปจะทำให้เกิดการชนกันของข้อมูลในแต่ละช่องของตารางแฮชมากจนเกินไป ซึ่งจะส่งผลให้จำนวนของ Candidate Itemsets ไม่ถูกลดขนาดลง แต่ในทางกลับกันถ้าเรากำหนดค่าขนาดของตารางแฮชให้มีขนาดใหญ่เกินไปตอนแรกเราอาจจะคิดว่าข้อมูลน่าจะมีการกระจายตัวอยู่ตามช่องตารางแฮชต่างๆ ซึ่งน่าจะส่งผลให้เราสามารถลดจำนวนของ Candiadte Itemsets ได้มากขึ้นไปด้วยนั้นเป็นความคิดที่ไม่ถูกต้อง จากตัวอย่างที่แสดงให้เห็นข้างต้นจะเห็นว่าขนาดของตารางแฮชที่เพิ่มมากขึ้นไม่ได้มี

ผลทำให้ข้อมูลเกิดการกระจายตัวเพิ่มมากขึ้น เพราะข้อมูลแต่ละเซตย่อยที่เรานำไปผ่านการคำนวณของ Hash Function นั้นจะได้ค่า Hash Address เพื่อเป็นตัวกำหนดตำแหน่งช่องของตารางแฮชสำหรับเซตย่อยแต่ละตัวอยู่แล้ว ซึ่งตำแหน่งของเซตย่อยแต่ละตัวอาจจะปรับเปลี่ยนตำแหน่งไปขึ้นอยู่กับค่าที่คำนวณได้ประกอบกับขนาดของตารางแฮช จึงทำให้สามารถสรุปได้ว่าการกำหนดขนาดของตารางแฮชให้เหมาะสมจะมีผลต่อประสิทธิภาพของอัลกอริทึม HFUP เป็นอย่างมาก โดยที่เราไม่สามารถหาหลักเกณฑ์ตายตัวอะไรมาใช้ในการพิจารณาถึงขนาดของตารางแฮชที่เหมาะสมได้ เนื่องจากมีปัจจัยอื่นๆ มาเกี่ยวข้องมากมาย เช่น จำนวนรายการสินค้าทั้งหมด ความน่าจะเป็นทั้งหมดของการสร้างเซตย่อยจากรายการสินค้าทั้งหมด และความหลากหลายของการสุ่มข้อมูลแต่ละชุดเพื่อนำมาใช้ในการทดลอง เป็นต้น

บทที่ 3
อัลกอริทึมแฮชเบสฟาสท์อัพเดทเพื่อค้นหากฎความสัมพันธ์ในฐานข้อมูลแบบเพิ่มขยาย

3.1 A Hash-Based Fast Update Algorithm (HFUP)

ปัจจุบันองค์กรส่วนใหญ่จะมีการจัดเก็บข้อมูลจำนวนมากลงฐานข้อมูล โดยที่ฐานข้อมูลนั้นจะมีการเปลี่ยนแปลงอยู่ตลอดเวลา เช่น ข้อมูลยอดการขาย (Grocery Sale Data), ข้อมูลคลังสินค้า (Stock Market Data) เป็นต้น การที่ฐานข้อมูลมีการเปลี่ยนแปลงทำให้กฎความสัมพันธ์ที่ได้ก่อนฐานข้อมูลจะมีการเปลี่ยนแปลงนั้นอาจเป็นกฎที่ไม่สามารถนำไปใช้งานได้อีก ดังนั้นเพื่อให้กฎความสัมพันธ์มีความน่าเชื่อถือ และเหมาะสมกับฐานข้อมูลทั้งหมดที่ทำการเพิ่มฐานข้อมูลใหม่เข้าไว้ด้วยกัน

TID	Items
1	I_2, I_4
2	I_1, I_2, I_5
3	I_2, I_3
4	I_1, I_3
5	I_1, I_2, I_4
6	I_2, I_5
7	I_2, I_6
8	I_5, I_6
9	I_1, I_3, I_6

Transaction Table

TID	Items
1	I_2, I_4
2	I_1, I_2, I_5
3	I_2, I_3
4	I_1, I_3
5	I_1, I_2, I_4

Original Database

TID	Items
6	I_2, I_5
7	I_2, I_6
8	I_5, I_6
9	I_1, I_3, I_6

New Transactions

รูปที่ 3.1 ตัวอย่างฐานข้อมูลเดิมและฐานข้อมูลที่เพิ่มเข้ามาใหม่

ในงานวิจัยนี้เราจะนำเสนอ A Hash-Based Fast Update Algorithm (HFUP) ซึ่งเป็นการนำเอาคุณสมบัติของ Direct Hashing and Pruning (DHP) มาใช้ร่วมกับอัลกอริทึม FUP เพื่อช่วยเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ให้เหมาะสมกับฐานข้อมูลที่มีการเปลี่ยนแปลง ทำให้เราสามารถนำกฎความสัมพันธ์ที่ได้นี้ไปใช้ในการพัฒนากลยุทธ์ด้านการตลาด และนำไปสู่การเพิ่มยอดขายให้กับธุรกิจต่อไป โดยรายละเอียดของ HFUP อัลกอริทึมจะกล่าวถึงในหัวข้อถัดไป

ในการหากฎความสัมพันธ์ของฐานข้อมูลเดิมนั้นเราจะนำเทคนิคของ DHP เข้ามาใช้แทนการหาด้วยอัลกอริทึมอะพิโอริ เพื่อลดเวลาในการทำงาน เราจะนำ Subset แต่ละตัวไปผ่านการคำนวณหาค่า Hash Function เพื่อจะได้ค่า Hash Address ของ Subset ตัวนั้นๆ และนำไปไว้ที่ Hash Table แล้วนำค่าจากตารางแฮชที่หาได้มาใช้ในการลดจำนวนของ Candidate 2-Itemsets ก่อน เพื่อเป็นการลดเวลาในการที่เราต้องมาสแกนหาค่า Support Factor จากจำนวนของ Candidate k-Itemsets ทั้งหมดตามวิธีการของอัลกอริทึมอะพิโอริเพื่อหา Large k-Itemsets และเพื่อเพิ่มความสะดวกในการประยุกต์ใช้งานร่วมกับ FUP อัลกอริทึมด้วย

```
/* Part 1 */
s = a minimum support;
set all the buckets of H2 to zero;  /* hash table */
forall transaction t ∈ D do begin
    count 1-Items occurrences in database;
    forall 2-Subsets x of t do
        H2[h2(x)] ++;
end
L1 = {c|c.count ≥ s, c is support factor};
```

รูปที่ 3.2 ส่วนที่ 1 ของ A Hash-Based Fast Update Algorithm (HFUP)

ในขั้นตอนแรกเราสามารถนำฐานข้อมูลเดิมตามรูปที่ 3.1 มาทำการหากฎความสัมพันธ์ด้วยเทคนิคของอัลกอริทึม HFUP โดยจะทำการสแกนฐานข้อมูลรอบแรกเพื่อนับค่า Support Factor ของ 1-Itemsets และก็จะสร้างเซตย่อยของ 2-Itemsets ทั้งหมดที่เป็นไปได้ (ตามรูปที่ 3.5) จากนั้นอัลกอริทึม HFUP ก็จะนำเซตย่อยของ 2-Itemses มาผ่าน Hash Function ทีละชุดเพื่อหา Hash Address แล้วก็จะบวก 1 เพิ่มเข้าไปใน Bucket ของ Hash Table จากนั้นก็จะสร้าง L_1 จากค่า Support Factor ของ 1-Itemsets ที่มีค่ามากกว่าหรือเท่ากับ Min_Sup ซึ่งจากตัวอย่างเราก็จะได้เซตของ L_1 ประกอบไปด้วย $\{I_1, I_2, I_3, I_4\}$

```
/* Part 2 */
k = 2;
D_k = D;            /* database for large k-Itemsets */
While (|{x|H_k[x] ≥ s}| ≥ LARGE) {
    /* make a hash table */
    gen_candidate(L_{k-1}, H_k, C_k);
    set all the buckets of H_{k+1} to zero;
    D_{k+1} = ϕ;
    forall transactions t ∈ D_k do begin
        count_support(t, C_k, k, Î);          /* Î ⊆ t */
        if (|Î| > k) then do begin
            make_hasht(Î, H_k, k, H_{k+1}, Ï);
            if (|Ï| > k) then D_{k+1} = D_{k+1} ∪ {Ï};
        end
    end
    L_k = {c ∈ C_k|c.count ≥ s};
    k ++;
}
```

รูปที่ 3.3 ส่วนที่ 2 ของ A Hash-Based Fast Update Algorithm (HFUP)

ในขั้นตอนถัดมาอัลกอริทึม HFUP (รูปที่ 3.3) ก็จะสร้าง C_2 จากการนำ L_1 มา join กันแล้วใช้ค่าจาก Hash Table ที่สร้างไว้ในรอบก่อนหน้านี้มาเปรียบเทียบกับค่า Min_sup โดยที่ Itemset ที่จะกลายเป็นสมาชิกของ C_2 จะต้องมีค่ามากกว่าหรือเท่ากับค่า Min_sup หลังจากได้ค่า C_2 แล้วเราจะทำการสแกนหาค่า Support Factor เพื่อนำมาตรวจเช็คดูว่ามี Itemsets ใดใน C_2 ที่มีค่า Support Factor มากกว่าหรือเท่ากับ Min_Sup เพื่อนำ Itemsets เหล่านั้นมาสร้างเป็น L_2 และจะนำ L_k มาจอยกันเพื่อสร้าง C_{k+1} ในรอบถัดไป และก็ทำการสแกนหาค่า Support Factor เพื่อนำมาเปรียบเทียบกับค่า Min_sup เพื่อนำมาใช้ในขั้นตอนการคัดเลือก C_k ไปเป็น L_k โดยขั้นตอนนี้จะทำงานวนลูปไปเรื่อยๆ จนกระทั่งไม่สามารถสร้าง L_k ได้อีกแล้ว (ตามรูปที่ 3.4) ซึ่งจากรูปที่ 3.5 เราจะเห็นว่า L_2 ประกอบไปด้วย $\{I_1I_2, I_2I_4\}$

เมื่อได้ Large k-Itemsets ทั้งหมดแล้ว เราก็จะนำ Large k-Itemsets โดยที่ k มีค่ามากกว่าหรือเท่ากับ 2 ไปสร้างเป็นกฎความสัมพันธ์ แล้วนำกฎความสัมพันธ์ที่ได้แต่ละชุดมาเปรียบเทียบกับค่าความเชื่อมั่นน้อยที่สุด เพื่อที่จะได้กฎความสัมพันธ์ที่มีความน่าเชื่อถือ และสามารถนำไปใช้ให้เกิดประโยชน์สูงสุด

```
/* Part 3 */
gen_candidate(L_{k-1}, H_k, C_k);
While (|C_k| > 0) {
    D_{k+1} = ϕ;
    forall transactions t ∈ D_k do begin
        count_support(t, C_k, k, Î);          /* Î ⊆ t */
        if (|Î| > k) then D_{k+1} = D_{k+1} ∪ {Î};
    end
    L_k = {c ∈ C_k | c.count ≥ s};
    if (|D_{k+1}| = 0) then break;
    C_{k+1} = apriori_gen(L_k);
    k ++;
}
```

รูปที่ 3.4 ส่วนที่ 3 ของ A Hash-Based Fast Update Algorithm (HFUP)

และเมื่อมีข้อมูลเพิ่มเข้ามาในฐานข้อมูล เราก็จะนำ FUP อัลกอริทึมมาร่วมประยุกต์ใช้ โดยในงานวิจัยนี้เราจะถือว่าเป็นส่วนที่ 4 ของ HFUP อัลกอริทึมไป ซึ่งเราจะแสดงส่วนที่ 4 ของ HFUP อัลกอริทึมไว้ดังรูปที่ 3.6 สำหรับหา New Large 1-Itemset (L'_1) โดยที่เราสามารถอธิบายการทำงานพร้อมทั้งยกตัวอย่างเป็นขั้นตอนต่างๆ ดังนี้

C_1	Support Factor (F_1)	L_1
$\{I_1\}$	3	$\{I_1\}$
$\{I_2\}$	4	$\{I_2\}$
$\{I_3\}$	2	$\{I_3\}$
$\{I_4\}$	2	$\{I_4\}$
$\{I_5\}$	1	

Making a hash table	
1	$\{I_2, I_4\}$
2	$\{I_1, I_2\}$, $\{I_1, I_5\}$, $\{I_2, I_5\}$
3	$\{I_2, I_3\}$
4	$\{I_1, I_3\}$
5	$\{I_1, I_2\}$, $\{I_1, I_4\}$, $\{I_2, I_4\}$

Minimum Support, s = 2

h({x y}) = ((order of x)*10 + (order of y)) mod 7;

	$\{I_2, I_4\}$			$\{I_1, I_2\}$		
$\{I_1, I_4\}$	$\{I_1, I_5\}$	$\{I_2, I_3\}$	$\{I_2, I_4\}$	$\{I_2, I_5\}$	$\{I_1, I_2\}$	$\{I_1, I_3\}$	
1	1	1	2	1	2	1	Hash table H_2
0	1	2	3	4	5	6	Hash address

Generating C_2		#in a bucket with the itemset
$L_1 * L_1$	$\{I_1, I_2\}$	2
	$\{I_1, I_3\}$	1
	$\{I_1, I_4\}$	1
	$\{I_2, I_3\}$	1
	$\{I_2, I_4\}$	2
	$\{I_3, I_4\}$	0

C_2
$\{I_1, I_2\}$
$\{I_2, I_4\}$

รูปที่ 3.5 แสดงการสร้าง Hash Table และ C_2 ด้วย HFUP อัลกอริทึม

ขั้นตอนที่ 1 ทำการอ่านข้อมูลใน New Transaction (จากรูปที่ 3.1) เพื่อนับจำนวนข้อมูลที่เกิดขึ้นของแต่ละ Large Itemsets (L_1) = {I_1, I_2, I_3, I_4} ใน New Transaction เพื่อนำมาปรับปรุงค่า Support Factor ของแต่ละ Large Itemsets (L_1) ในฐานข้อมูลทั้งหมด เพื่อดูว่ากฎความสัมพันธ์เดิมยังเป็นไปตามเงื่อนไขหรือไม่ เมื่อฐานข้อมูลมีการเปลี่ยนแปลง โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ และจากตัวอย่างข้างต้นที่เรากำหนดค่า Minimum Support ไว้เท่ากับ 2 นั้น เราสามารถนำมาคิดเป็นเปอร์เซ็นต์เมื่อเทียบกับจำนวนทรานแซคชั่นเดิมได้เป็น 40% เพื่อเราจะได้นำมาคิดหาค่า Minimum Support ใหม่เมื่อจำนวนทรานแซคชั่นเพิ่มมากขึ้น

คำนวณค่า Min_sup รวมได้ดังนี้ $40\% \times 9 = 4$

$I_1.support_{UD} = 4$ มีค่าเท่ากับค่าของ Min_sup รวม ดังนั้น $I_1 \in L'_1$

$I_2.support_{UD} = 6$ มีค่ามากกว่าค่าของ Min_sup รวม ดังนั้น $I_2 \in L'_1$

$I_3.support_{UD} = 3$ มีค่าน้อยกว่าค่าของ Min_sup รวม ดังนั้น $I_3 \notin L'_1$ จึงถูกตัดออกจาก Large 1-Itemsets (L_1)

$I_4.support_{UD} = 2$ มีค่าน้อยกว่าค่าของ Min_sup รวม ดังนั้น $I_4 \notin L'_1$ จึงถูกตัดออกจาก Large 1-Itemsets (L_1)

ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น ที่ไม่ได้อยู่ใน Large Itemsets (L_1) = {I_5, I_6} เพื่อหา Candidate Itemset (C_1) ไว้ใช้ในการหากฎความสัมพันธ์ใหม่ โดยทำการหาค่า Support ของ Itemsets ใน Candidate Itemsets (C_1) และทำการตรวจสอบตามเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_1$

คำนวณค่า Min_sup ของฐานข้อมูลใหม่ได้ดังนี้ $40\% \times 4 = 2$

$I_5.support_d = 2$ มีค่าเท่ากับค่าของ Min_sup ของฐานข้อมูลใหม่ ดังนั้น $I_5 \in C_1$

$I_6.support_d = 3$ มีค่ามากกว่าค่าของ Min_sup ของฐานข้อมูลใหม่ ดังนั้น $I_6 \in C_1$

ขั้นตอนที่ 3 ทำการอ่านค่า Support Factor ในตาราง F_1 ของ DB เพื่อดูจำนวนข้อมูลของ Itemsets ใน Candidate Itemsets (C_1) = {I_5, I_6} มาปรับปรุงร่วมกับค่า Support Factor ใน db เพื่อหากฎความสัมพันธ์ใหม่ โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ เพื่อทำการค้นหา New Large Itemsets (L'_1)

$I_5.support_{UD} = 3$ มีค่าน้อยกว่าค่าของ Min_sup รวม ดังนั้น $I_5 \notin L'_1$

$I_6.support_{UD} = 3$ มีค่าน้อยกว่าค่าของ Min_sup รวม ดังนั้น $I_6 \notin L'_1$

$\therefore$ ดังนั้นผลลัพธ์ของกฎความสัมพันธ์ใหม่ที่ได้จากการอ่านข้อมูลครั้งแรกจะได้ $L'_1 = \{I_1, I_2\}$

Input: (1) *DB*: the original database (with its size, i.e., the total number of transaction, equal to *D*); (2) L_k: the set of all large *k*-itemsets in *DB*, where *k*= 1, …, r; (3) *db*: an increment database (with its size equal to *d*); and (4) *s*: the minimum support threshold.

Output: L': The set of all large itemsets in $DB \cup db$.

Method: The 1st iteration: /* find L'_1, the set of all large 1-itemsets in $DB \cup db$ */

$W = L_1$; $C = \phi$; $L'_1 = \phi$; $P = \phi$;

/* *W*: winners, *C*: candidate sets, L'_1: initialized, *P*: for optimization */

for_all $T \in db$ do /* scan *db* */

for_all 1-itemset $X \subseteq T$ do {

if $X \in W$ then $X.support_d$++;

else {

if $X \notin C$ then { $C = C \cup \{X\}$; $X.support_d = 0$; }

/*init the support cont and add *X* into *C* */

$X.supprt_d$++; }

};

for_all $X \in W$ do /* put winners into L'_1*/

if $X.support_{UD} \geq s \times (D + d)$

then $L'_1 = L'_1 \cup \{X\}$;

for_all $X \in C$ do /*prune candidate sets in *C*/

if $X.support_d < s \times d$

then { $C = C - \{X\}$; $P = P \cup \{X\}$; } /* *P* will be used for optimization */

for_all $T \in DB$ do /* scan *DB* */

for_all 1-itemset $X \subseteq T$ do {

if $X \in C$ then $X.support_D$++;

if $X \in P$ then removes *X* from *T*; }; /* Transaction *T* is reduced */

for_all $X \in C$ do /* put winners into L'_1 */

if $X.support_{UD} \geq s \times (D + d)$ then $L'_1 = L'_1 \cup \{X\}$;

return L'_1. /* end of the 1st iteration */

รูปที่ 3.6 ส่วนที่ 4 **ของ** A Hash-Based Fast Update Algorithm (HFUP)

The k-th iteration: /* for $k = 2$ or larger, repeat this program fragment to find L'_k, the set of all large k-itemsets in the updated database, until either L'_k returned is empty or $db = \phi$ */

$W = L_k$; $L'_k = \phi$; /* W: winners; L'_k initialized */

C = apriori_gen1(L'_{k-1}) - L_k;

/* the size-k candidate sets */

for_all k-itemset $X \in W$ do /* prune off losers in W */

for_all (k-1)-itemset $Y \in L_{k-1} - L'_{k-1}$ do

if $Y \subseteq X$ then { $W = W - \{X\}$; break; }

for_all $T \in db$ do { /* scan db */

for_all $X \in$ Subset(W, T) do $X.support_d$++;

/* Subset(W,T) returns all the sets in W contained in T */

for_all $X \in$ Subset(C, T) do $X.support_d$++; /* find support of all $X \in C$ */

Reduce_db (T);

/* Some items in transactions in db can be removed, discussed in next section */

}

for_all $X \in W$ do /* put the winners from W into L'_k */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

for_all $X \in C$ do /* prune candidate sets in C */

if $X.support_d < s \times d$ then $C = C - \{X\}$;

for_all $T \in DB$ do { /* scan DB */

for_all $X \in$ Subset(C, T) do $X.support_D$++;

Reduce_Db(T); }

/* Some items in transactions in DB can be removed, discussed in next section */

for_all $X \in C$ do /* put the winners from C into L'_k */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

return L'_k. /* The end of the k-th iteration */

รูปที่ 3.7 ส่วนที่ 5 ของ A Hash-Based Fast Update Algorithm (HFUP)

หลังจากที่เราหาค่า Large 1-Itemset (L'_1) ตามส่วนที่ 4 ของ HFUP อัลกอริทึม (รูปที่ 3.7) เรียบร้อยแล้ว เราก็จะทำการหากฎความสัมพันธ์ของ Large Itemsets (L'_2) ในฐานข้อมูลทั้งหมด ตามส่วนที่ 5 ของ HFUP อัลกอริทึม ซึ่งจะสามารถเทียบได้กับการอ่านข้อมูลรอบสองของ FUP อัลกอริทึมนั่นเอง โดยมี Large Itemsets (L'_1) ของฐานข้อมูลทั้งหมด = $\{I_1, I_2\}$ จากนั้นทำการหา Large Itemsets (L'_2) ที่เป็นไปตามเงื่อนไขตามค่า Support Factor ที่ถูกกำหนด = 40% ของข้อมูลทั้งหมด โดยมีขั้นตอนการทำงานดังนี้

ขั้นตอนที่ 1 ทำการลบ Itemset ใน Large Itemsets (L_2) = $\{I_1I_2, I_2I_4\}$ ที่ไม่ได้เป็นไปตามเงื่อนไข $X \in L_2$, $Y \in L_1\text{-}L'_1$ จะเห็นว่า I_3 และ $I_4 \in L_1\text{-}L'_1$ ดังนั้น I_2I_4 จึงถูกลบออกจาก Large Itemsets (L_2) จากนั้นทำการอ่านข้อมูลใน db เพื่อทำการปรับปรุงค่า Support ($X.support_{UD}$) ของ Large Itemsets (L_2) ในฐานข้อมูลทั้งหมด เพื่อดูว่ากฎความสัมพันธ์เดิมยังเป็นไปตามเงื่อนไขหรือไม่เมื่อฐานข้อมูลมีการเปลี่ยนแปลง ซึ่งเราจะทำการตรวจสอบเงื่อนไข $X.support_d \geq s(D + d)$ โดยที่ $X \in L_2$

$I_1I_2.support_{UD} = 2$ มีค่าน้อยกว่าค่า Min_sup รวม ดังนั้น $I_1I_2 \notin L'_2$

ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น โดยนำ Large Itemsets (L'_1) ของฐานข้อมูลทั้งหมด = $\{I_1, I_2\}$ มาทำการ join ข้อมูลเพื่อทำการหา Candidate Itemset (C_2) โดยที่ Itemsets ที่ได้ $\notin L_2$ ดังนั้น $C_2 = \{I_1I_2\}$ แต่ $I_1I_2 \in L_2$ จึงไม่ถูกแสดงใน C_2 ดังนั้น $C_2 = \phi$ จึงไม่จำเป็นต้องอ่านข้อมูลในฐานข้อมูล (db) เพื่อหาค่า Support ($X.support_d$) ของ Itemsets ใน Candidate Itemset (C_2) เพื่อตรวจสอบเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_2$ แต่เนื่องจาก $C_2 = \phi$ เราจึงสามารถข้ามขั้นตอนนี้ไปได้เลย เพื่อไปทำงานในขั้นตอนต่อไป

ขั้นตอนที่ 3 ทำการอ่านค่า Support Factor ในตาราง F_2 ของ DB เพื่อดูจำนวนข้อมูลของ Itemsets ใน Candidate Itemsets (C_2) มาปรับปรุงร่วมกับค่า Support ใน db เพื่อหาค่า Support Count ($X.support_{UD}$) ของ Itemsets ที่อยู่ใน Candidate Itemset (C_2) ของฐานข้อมูลทั้งหมด และนำข้อมูลที่ได้ไปหากฎความสัมพันธ์เมื่อมีข้อมูลใหม่เพิ่มเข้ามาในฐานข้อมูล จากนั้นทำการตรวจสอบเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in C_2$ แต่เนื่องจาก $C_2 = \phi$ เราจึงสามารถข้ามขั้นตอนนี้ไปได้เลย เพื่อไปทำงานในขั้นตอนต่อไป

ซึ่งจากทั้ง 3 ขั้นตอนข้างต้น จะเห็นว่าไม่มี Itemsets ใดผ่านเงื่อนไขมาเป็น Large Itemsets (L'_2) ได้เลย $\therefore$ เราจะสรุปผลลัพธ์ที่ได้จากการอ่านข้อมูลรอบที่สองจะได้ $L'_2 = \phi$ ทำให้เราไม่สามารถนำค่า Large k-Itemsets ไปสร้างกฎความสัมพันธ์ใดๆ ได้เลยหลังจากมีการเพิ่มของข้อมูลใหม่เข้ามา ซึ่งกรณีนี้อาจจะเกิดขึ้นในชีวิตจริงของเราก็ได้ แต่โอกาสที่จะพบน้อยมากเนื่องจากข้อมูลที่เรานำมาวิเคราะห์นั้นมีขนาดใหญ่มากนั่นเอง

บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลอง และผลการทดลอง ซึ่งงานวิจัยนี้จะใช้โปรแกรม Microsoft Visual Studio 2010 และ Access Database Engine 2007 ในการทดลอง และสร้างชุดข้อมูลที่ใช้ในการทดลองโดยใช้โปรแกรม Matlab 7

4.1 วัตถุประสงค์ของการทดลอง

ในงานวิจัยนี้จะนำเสนอแนวทางในการค้นหากฎความสัมพันธ์ของข้อมูล โดยใช้อัลกอริทึม HFUP ซึ่งอัลกอริทึมนี้เป็นการประยุกต์เอาคุณสมบัติของ Direct Hashing and Pruning (DHP) มาใช้ร่วมกับอัลกอริทึม FUP เพื่อช่วยลดจำนวนของ Candiadte Itemsets และทำให้เวลาในการประมวลผลข้อมูลลดน้อยลง โดยจะทำการทดลองและเปรียบเทียบประสิทธิภาพในการค้นหากฎความสัมพันธ์ระหว่างอัลกอริทึม HFUP และอัลกอริทึม FUP ซึ่งจะกำหนดตัววัดประสิทธิภาพในการทำงานของอัลกอริทึมทั้ง 2 อัลกอริทึม ดังนี้

1. เวลาในการประมวลผล โดยในส่วนของอัลกอริทึม HFUP จะแสดงเวลาในการสร้างตารางแฮช, เวลาในการอ่านค่าจากตารางแฮชมาให้แต่ละ Candidate Itemsets เพื่อใช้เปรียบเทียบกับค่า Minimum Support เพื่อหา Candidate 2-Itemsets, เวลาในการสแกนหาค่า Support ให้กับ Candidate Itemsets แต่ละตัว เพื่อนำไปเปรียบเทียบกับค่า Minimum Support ในการหา Large Itemsets และเวลารวมของการประมวลผลทั้งหมด ในส่วนของอัลกอริทึม FUP จะแสดงเวลาในการสแกนหาค่า Support ให้แต่ละ Candidate Itemsets เพื่อนำมาเปรียบเทียบกับค่า Minimum Support ในการหา Large Itemsets และเวลารวมในการทำงานทั้งหมด
2. จำนวนของ Candidate Itemsets ทั้งหมดของแต่ละอัลกอริทึม

4.2 แผนการทดลอง

ในการทดลองนี้เราจะแบ่งการทดลองออกเป็น 3 การทดลอง ตามปัจจัยที่ต้องการศึกษา โดยที่แต่ละการทดลองเราจะกำหนดเงื่อนไขให้มีเพียงแค่ตัวแปรเดียวที่จะมีการเปลี่ยนแปลงค่า ในขณะที่ตัวแปรอื่นๆ เราจะกำหนดให้มีค่าคงที่ภายในแต่ละการทดลอง ซึ่งแต่ละการทดลองจะมีรายละเอียดในการกำหนดค่าเพื่อใช้ในการทดลองดังนี้

การทดลองที่ 1 เพื่อศึกษาผลของขนาดของฐานข้อมูลเพิ่มขยายที่มีต่อเวลาในการทำงานของอัลกอริทึม HFUP โดยกำหนดจำนวนทรานแซคชั่นของฐานข้อมูลเดิม(Original Database) จำนวน 10,000 ทรานแซคชั่น ค่า Minimum Support มีค่าเท่ากับ 0.03 และขนาดของตารางแฮชให้มีขนาดเท่ากับ 100 สำหรับการทดลองนี้เราจะใช้จำนวนของทรานแซคชั่นของฐานข้อมูลใหม่ (Incremental Database) ที่มีจำนวนไม่เท่ากันเป็นตัวแปรในการเปรียบเทียบ ซึ่งเราได้กำหนดชุดข้อมูลของฐานข้อมูลใหม่ไว้ทั้งหมด 5 ชุดคือ

- ชุดที่ 1 ให้ฐานข้อมูลใหม่มีจำนวน 2,000 ทรานแซคชั่น
- ชุดที่ 2 ให้ฐานข้อมูลใหม่มีจำนวน 4,000 ทรานแซคชั่น
- ชุดที่ 3 ให้ฐานข้อมูลใหม่มีจำนวน 6,000 ทรานแซคชั่น
- ชุดที่ 4 ให้ฐานข้อมูลใหม่มีจำนวน 8,000 ทรานแซคชั่น
- ชุดที่ 5 ให้ฐานข้อมูลใหม่มีจำนวน 10,000 ทรานแซคชั่น

การทดลองที่ 2 เพื่อศึกษาผลของขนาดของตารางแฮชที่มีต่อเวลาในการทำงานของอัลกอริทึม HFUP โดยกำหนดจำนวนทรานแซคชั่นของฐานข้อมูลเดิม(Original Database) จำนวน 10,000 ทรานแซคชั่น ค่า Minimum Support มีค่าเท่ากับ 0.03 และจำนวนทรานแซคชั่นของฐานข้อมูลใหม่จำนวน 6,000 ทรานแซคชั่น สำหรับการทดลองนี้เราจะใช้ขนาดของตารางแฮชเป็นตัวแปรในการเปรียบเทียบ โดยกำหนดให้มีขนาดของตารางแฮชที่ใช้ในการทดลองทั้งหมด 7 ค่า คือ 50 ช่อง, 75 ช่อง, 100 ช่อง, 150 ช่อง, 350 ช่อง, 500 ช่อง และ 650 ช่อง

การทดลองที่ 3 เพื่อศึกษาผลของค่าสนับสนุนต่ำสุด (Minimum Support) ที่มีต่อเวลาในการทำงานของอัลกอริทึม HFUP โดยกำหนดจำนวนทรานแซคชั่นของฐานข้อมูลเดิม(Original Database) จำนวน 10,000 ทรานแซคชั่น ค่าขนาดของตารางแฮชเท่ากับ 100 และจำนวนทรานแซคชั่นของฐานข้อมูลใหม่จำนวน 6,000 ทรานแซคชั่น สำหรับการทดลองนี้เราจะใช้ค่า Minimum Support เป็นตัวแปรในการเปรียบเทียบ โดยกำหนดให้มีค่า Minimum Support ที่ใช้ในการทดลองทั้งหมด 7 ค่า คือ 0.01, 0.02, 0.03, 0.04, 0.05, 0.06 และ 0.07

4.2.1 การสร้างชุดข้อมูลสังเคราะห์

ได้มีการนำเสนอการสร้างชุดข้อมูลสังเคราะห์ไว้ในเอกสาร Fast Algorithm for Mining Association Rules [2] เพื่อใช้สำหรับประเมินประสิทธิภาพในการทำงานของอัลกอริทึม โดยใช้หลักการทางสถิติต่างๆ มาประยุกต์ใช้ในการสร้างชุดข้อมูลสังเคราะห์ ซึ่งจะนำไปสุ่มค่าความน่าจะเป็นเพื่อหาขนาดของไอเทมเซต, ขนาดของทรานแซคชั่น และการสุ่มค่าไอเทมเพื่อนำไปใส่ในทรานแซคชั่นด้วยวิธีการแจกแจงแบบต่างๆ เช่น การแจกแจงแบบยูนิฟอร์ม, การแจกแจงแบบปกติ, การแจกแจงแบบปัวส์ซอง และการแจกแจงแบบเอ็กซ์โพแนนเชียล เป็นต้น โดยลักษณะของชุดข้อมูลสังเคราะห์จะเป็นทรานแซคชั่นของลูกค้าแต่ละคนที่ไปซื้อสินค้าตามซุปเปอร์มาร์เก็ตต่างๆ และไอเทมต่างๆ ในทรานแซคชั่นจะเปรียบเสมือนรายการสินค้าแต่ละรายการที่ลูกค้าซื้อไป

ดังนั้นขนาดของทรานแซคชั่นที่สร้างจะได้มาจากการจัดกลุ่มของไอเทมต่างๆ ด้วยค่าเฉลี่ย ทำให้จำนวนของ Large Itemset สูงสุดอาจจะประกอบด้วยขนาดของไอเทมที่ต่างกัน ซึ่งชุดข้อมูลสังเคราะห์จะมรพารามิเตอร์ต่างๆ ที่สำคัญดังนี้

- N หมายถึงจำนวนของไอเทมที่มีในฐานข้อมูลทั้งหมด ในการสร้างชุดข้อมูลสังเคราะห์นี้ N = 100 ซึ่งจะแสดงในรูปของตัวเลข 1 – 100

- |D| หมายถึงจำนวนของทรานแซคชั่นที่มีทั้งหมดในฐานข้อมูล

- |L| หมายถึงจำนวนของไอเทมเซตทั้งหมดที่สามารถเป็น Large Itemsets ได้

- |I| หมายถึงค่าเฉลี่ยของ |L| เช่น |L| = 100 และ |I| = 4 หมายความว่าค่าเฉลี่ยของ |L| จำนวน 100 ชุด จะมีค่าเฉลี่ยของ Large Itemsets มีขนาดประมาณ 4 ไอเทม

- |T| หมายถึงค่าเฉลี่ยของขนาดทรานแซคชั่น ในการสร้างชุดข้อมูลสังเคราะห์นี้จะใช้การแจกแจงแบบปัวส์ซองในการสุ่มขนาดของทรานแซคชั่น โดยจะสุ่มจำนวนของทรานแซคชั่นด้วยขนาดต่างๆ กัน

ในการทดลองนี้ได้ทำการสังเคราะห์ชุดข้อมูลของฐานข้อมูลเดิมเป็นจำนวน 10,000 ทรานแซคชั่น โดยที่แต่ละทรานแซคชั่นจะสุ่มไอเทมให้อยู่ในรูปของตัวเลข 1- 100 ซึ่งจะแสดงตัวอย่างของชุดข้อมูลที่สังเคราะห์มาตามตารางที่ 4.1

ตารางที่ 4.1 แสดงตัวอย่างข้อมูลที่ใช้ในการทดลอง

TID	ITEMS
1	18, 21, 22, 42, 44, 54, 65, 72, 79
2	4, 9, 14, 16, 18, 21, 26, 35, 53, 54, 55, 58, 65, 77, 87
3	2, 8, 11, 22, 36, 42, 73, 76, 84
4	2, 7, 8, 10, 15, 22, 39, 45, 53, 82, 94, 97
5	1, 2, 4, 10, 23, 32, 37, 38, 48, 75, 79, 84, 89, 97
6	1, 2, 18, 33, 47, 70, 71, 75, 83, 100
7	2, 7, 8, 10, 17, 32, 33, 42, 47, 54, 66, 67, 71, 78, 84, 85, 91
8	7, 14, 31, 37, 59, 78, 97
9	4, 6, 8, 31, 53, 58, 59, 61, 78, 79, 83
10	2, 3, 22, 23, 34, 35, 40, 58, 84, 93, 96, 97
11	1, 4, 23, 51, 54, 55, 63, 68
12	2, 11, 21, 24, 26, 31, 40, 54, 55, 60, 65, 67, 87, 89, 92
…	…

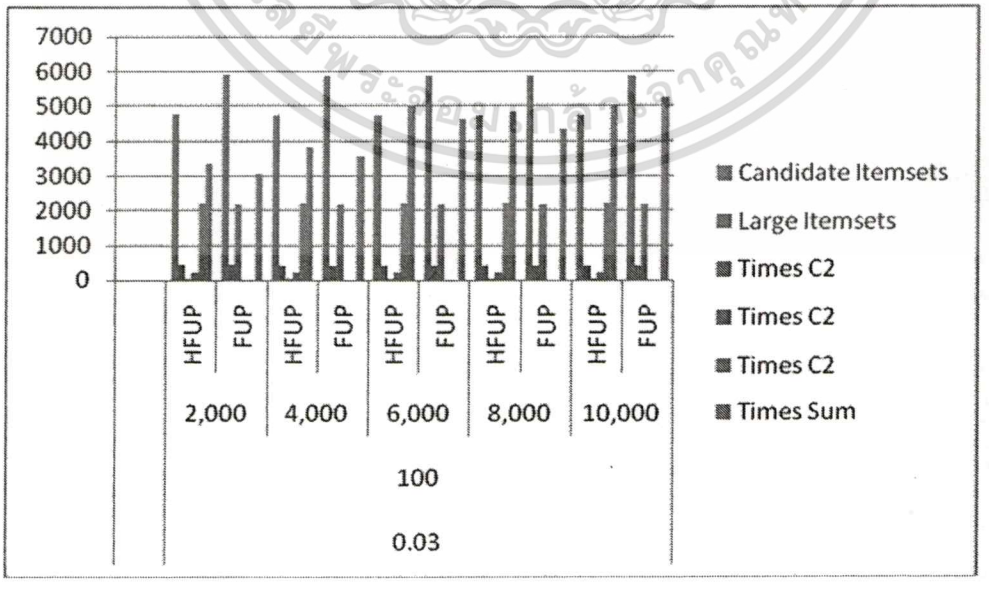
4.3 ผลการทดลอง

จากการทดลองทั้ง 3 การทดลอง ได้เปรียบเทียบด้านประสิทธิภาพในการทำงานของอัลกอริทึม HFUP และอัลกอลิทึม FUP โดยใช้เวลาในการทำงานเป็นตัวชี้วัด และแสดงให้เห็นถึงคุณสมบัติของการใช้ Hash Function ของอัลกอริทึม HFUP เพื่อลดขนาดของ Candidate Itemset ด้วย ซึ่งในแต่ละชุดการทดลองจะมีรายละเอียดดังนี้

ในการทดลองที่ 1 เราจะกำหนดค่าสนับสนุนขั้นต่ำ (min_sup) ที่ใช้ในการทดลองคือ 0.03 และขนาดของตารางแฮชเท่ากับ 100 โดยกำหนดให้จำนวนทรานแซคชั่นของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น และจำนวนทรานแซคชั่นของฐานข้อมูลใหม่ทั้งหมด 5 จำนวน โดยจะแสดงผลการทดลองตามตารางที่ 4.2

ในการทดลองที่ 2 เราจะกำหนดค่าสนับสนุนขั้นต่ำ (min_sup) ที่ใช้ในการทดลองคือ 0.03 กำหนดให้จำนวนทรานแซคชั่นของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น และจำนวนทรานแซคชั่นของฐานข้อมูลใหม่เท่ากับ 6,000 ทรานแซคชั่น ซึ่งกำหนดขนาดของตารางแฮชไว้ทั้งหมด 7 ค่า คือ ขนาด 50 ช่อง, ขนาด 75 ช่อง, ขนาด 100 ช่อง, ขนาด 150 ช่อง, ขนาด 350 ช่อง, ขนาด 500 ช่อง และขนาด 650 ช่อง โดยจะแสดงผลการทดลองตามตารางที่ 4.3

ในการทดลองที่ 3 เราจะกำหนดให้จำนวนทรานแซคชั่นของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น จำนวนทรานแซคชั่นของฐานข้อมูลใหม่เท่ากับ 6,000 ทรานแซคชั่น และขนาดของตารางแฮชเท่ากับ 100 ซึ่งกำหนดค่า Minimum Support ไว้ทั้งหมด 7 ค่า คือ 0.01, 0.02, 0.03, 0.04, 0.05, 0.06 และ 0.07 โดยจะแสดงผลการทดลองตามตารางที่ 4.4



รูปที่ 4.1 เปรียบเทียบประสิทธิภาพการทำงานของการทดลองที่ 1

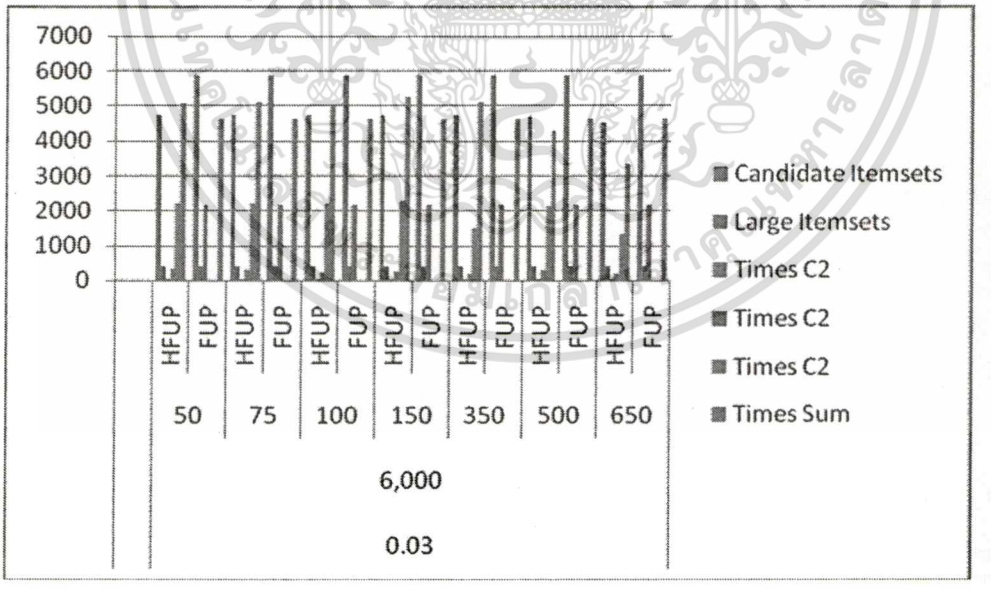
ตารางที่ 4.2 แสดงผลการทดลองที่ 1 ตามตัวแปรขนาดของฐานข้อมูลเพิ่มขยาย

Min_sup	ขนาดของตารางแฮช	ขนาดของฐานข้อมูลเพิ่มขยาย	Algorithm	Candidate Itemsets	Large Itemsets	Times (sec)			
						C_2			Sum
						สร้างตารางแฮช	ค้นหาในตารางแฮช	นับค่า Support	
0.03	100	2,000	HFUP	4,762	434	40.788	241.466	2,225.664	3,338.800
			FUP	5,893	434	2,183.523			3,045.800
		4,000	HFUP	4,748	431	40.788	241.466	2,225.664	3,810.100
			FUP	5,882	431	2,183.523			3,550.300
		6,000	HFUP	4,746	431	40.788	241.466	2,225.664	5,039.400
			FUP	5,874	431	2,183.523			4,626.800
		8,000	HFUP	4,741	415	40.788	241.466	2,225.664	4,830.200
			FUP	5,868	415	2,183.523			4,335.000
		10,000	HFUP	4,747	425	40.788	241.466	2,225.664	4,993.300
			FUP	5,872	425	2,183.523			5,257.300

หมายเหตุ – สร้างตารางแฮช คือเวลาที่ใช้ในการหาเซตย่อยของ 2-Itemset ทั้งหมดไปจนกระทั่งสร้างตารางแฮชเสร็จเรียบร้อย

- ค้นหาในตารางแฮช คือเวลาในการหาค่าจากตารางแฮชมาให้ Candidate 2-Itemsets ครบทุกตัวจนนำไปเปรียบเทียบกับค่า Min_sup จนได้เป็น C_2 ที่แท้จริง
- นับค่า Support และ C2 คือเวลาในการสแกนหาค่า Support ให้แต่ละ Candiadte 2-Itemsets จนนำไปเปรียบเทียบกับค่า Min_sup จนได้ L_2

จากตารางที่ 4.2 จะเห็นว่าเมื่อเราได้ทำการกำหนดค่าจำนวนของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น ค่า Minimum Support = 0.03 และขนาดของตารางแฮชเท่ากับ 100 เป็นค่าตายตัว โดยจะให้มีการสุ่มจำนวนฐานข้อมูลที่เพิ่มเข้ามาเป็นจำนวน 2,000 ทรานแซคชั่น, จำนวน 4,000 ทรานแซคชั่น, จำนวน 6,000 ทรานแซคชั่น, จำนวน 8,000 ทรานแซคชั่น และจำนวน 10,000 ทรานแซคชั่นไว้นั้น จากผลการทดลองเราจะเปรียบเทียบประสิทธิภาพในการทำงานของอัลกอริทึมทั้ง 2 ในเชิงของจำนวน Candidate Itemsts และเวลาในการประมวลผลทั้งหมด จะเห็นว่าจำนวนของฐานข้อมูลใหม่ที่เพิ่มมากขึ้น หรือลดน้อยลง ไม่ส่งผลกระทบต่อประสิทธิภาพในการทำงานของอัลกอริทึม HFUP ดังจะเห็นได้จากจำนวนของ Candidate Itemsets ที่มีจำนวนลดลงทั้งหมดเมื่อเทียบกับอัลกอริทึม FUP ถึงแม้ว่าเวลาในการประมวลผลของอัลกอริทึม HFUP จะมากกว่าอัลกอริทึม FUP นั้นก็ไม่ได้เป็นผลมาจากการเพิ่มหรือลดขนาดของจำนวนฐานข้อมูลใหม่ เนื่องจากข้อมูลทั้ง 5 ชุดนี้จะให้ผลในเรื่องเวลาไปในทิศทางเดียวกันคือจำนวนของฐานข้อมูลใหม่ที่เพิ่มมากขึ้นจะทำให้เวลาในการทำงานเพิ่มมากขึ้นไปด้วย เมื่อเทียบกับจำนวนของฐานข้อมูลใหม่ที่มีจำนวนน้อยกว่า ซึ่งจะสามารถสรุปได้ว่าจำนวนของฐานข้อมูลที่มีจำนวนมากขึ้น หรือลดลงไม่มีผลต่อประสิทธิภาพในการทำงานของอัลกอริทึม HFUP ซึ่งจะเห็นได้จากจำนวนของ Candidate Itemsets ที่มีจำนวนน้อยกว่าอัลกอริทึม FUP ทั้งหมด และในเชิงของเวลาที่อัลกอริทึม HFUP ใช้มากกว่าอัลกอริทึม FUP นั้นน่าจะมาจากปัจจัยอื่นเป็นสาเหตุมากกว่า

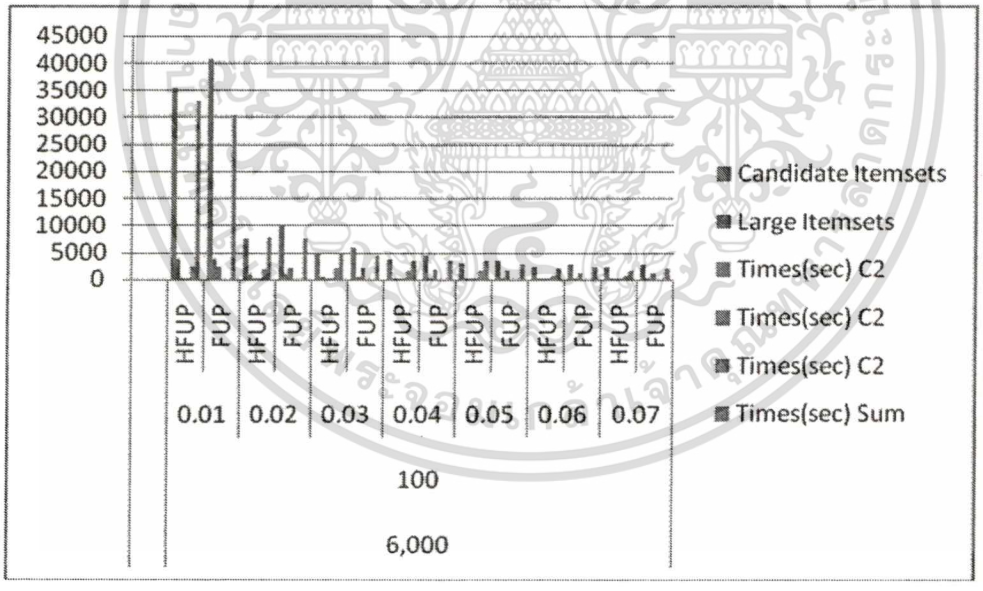


รูปที่ 4.2 เปรียบเทียบประสิทธิภาพการทำงานของการทดลองที่ 2

ตารางที่ 4.3 แสดงผลการทดลองที่ 2 ตามตัวแปรขนาดของตารางแฮช

Min_sup	ขนาดของฐานข้อมูลเพิ่มขยาย	ขนาดของตารางแฮช	Algorithm	Candidate Itemsets	Large Itemsets	Times(sec)			
						C_2			Sum
						สร้างตารางแฮช	ค้นหาในตารางแฮช	นับค่า Support	
0.03	6,000	50	HFUP	4,746	431	64.012	331.500	2,199.820	5,053.600
			FUP	5,874	431	2,183.523			4,626.800
		75	HFUP	4,746	431	49.094	300.998	2,196.830	5,087.600
			FUP	5,874	431	2,183.523			4,626.800
		100	HFUP	4,746	431	40.788	241.466	2,225.664	5,039.400
			FUP	5,874	431	2,183.523			4,626.800
		150	HFUP	4,746	431	41.671	264.193	2,271.136	5,256.300
			FUP	5,874	431	2,183.523			4,626.800
		350	HFUP	4,746	431	32.787	205.214	1,499.090	5,087.600
			FUP	5,874	431	2,183.523			4,626.800
		500	HFUP	4,689	431	55.398	297.235	2,148.215	4,302.900
			FUP	5,874	431	2,183.523			4,626.800
		650	HFUP	4,504	431	49.172	209.393	1,340.742	3,325.300
			FUP	5,874	431	2,183.523			4,626.800

จากตารางที่ 4.3 จะเห็นว่าเมื่อเราได้ทำการกำหนดค่าจำนวนของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น ค่า Minimum Support = 0.03 และจำนวนของฐานข้อมูลใหม่เท่ากับ 6,000 ทรานแซคชั่น โดยจะทำการสุ่มขนาดของตารางแฮชเพื่อใช้ในการทดลองทั้งหมด 7 ค่า คือ 50, 75, 100, 150, 350, 500 และ 650 นั้น จากผลการทดลองเราจะเปรียบเทียบประสิทธิภาพในการทำงานของอัลกอริทึมทั้ง 2 ในเชิงของจำนวน Candidate Itemsts และเวลาในการประมวลผลทั้งหมด จะเห็นว่าการกำหนดขนาดของตารางแฮชจะมีผลกระทบต่อเวลาในการประมวลของอัลกอริทึม HFUP จากตารางที่ 4.3 จะเห็นว่าถ้าขนาดของตารางแฮชเท่ากับ 50, 75, 100, 150 และ 350 เวลาในการทำงานของอัลกอริทึม HFUP จะมากกว่าอัลกอริทึม FUP แต่ถ้าขนาดของตารางแฮชเท่ากับ 500 และ 650 เวลาในการทำงานของอัลกอริทึม HFUP จะน้อยกว่าเวลาในการทำงานของอัลกอริทึม FUP และจะเห็นว่าขนาดของตารางแฮชเท่ากับ 650 จะทำให้เวลาในการทำงานของอัลกอริทึม HFUP ใช้เวลาน้อยที่สุด ดังนั้นการกำหนดขนาดของตารางแฮชให้เหมาะสมกับข้อมูลที่เราต้องการหากฎความสัมพันธ์นั้นจะมีผลกระทบต่อประสิทธิภาพในการทำงานของอัลกอริทึม HFUP ในเชิงของเวลาในการประมวลผล แต่ถ้าดูในแง่ของการลดจำนวนของ Candidate Itemsets นั้น จะเห็นว่าอัลกอริทึม HFUP มีประสิทธิภาพมากกว่าอัลกอริทึม FUP



รูปที่ 4.3 เปรียบเทียบประสิทธิภาพการทำงานของการทดลองที่ 3

ตารางที่ 4.4 แสดงผลการทดลองที่ 3 ตามตัวแปรค่าสนับสนุนต่ำสุด (Minimum Support)

ขนาดของฐานข้อมูลเพิ่มขยาย	ขนาดของตารางแฮช	Min_sup	Algorithm	Candidate Itemsets	Large Itemsets	Times(sec)			
						C_2			Sum
						สร้างตารางแฮช	ค้นหาในตารางแฮช	นับค่า Support	
6,000	100	0.01	HFUP	35,485	3,813	48.359	312.470	2,349.322	32,954.400
			FUP	40,709	3,813	2,368.608			30,471.354
		0.02	HFUP	7,649	1,126	46.446	287.743	2,034.124	7,916.600
			FUP	10,261	1,126	2,276.410			7,555.600
		0.03	HFUP	4,746	431	40.788	241.466	2,225.664	5,039.400
			FUP	5,874	431	2,183.523			4,626.800
		0.04	HFUP	3,927	197	43.438	249.256	1,759.957	3,721.200
			FUP	4,633	197	1,994.434			3,705.100
		0.05	HFUP	3,147	116	42.429	208.131	1,693.660	3,496.400
			FUP	3,659	116	1,623.002			2,985.400
		0.06	HFUP	2,460	83	34.016	132.082	898.448	2,274.600
			FUP	2,896	83	1,300.107			2,329.000
		0.07	HFUP	2,386	76	33.766	133.372	825.081	1,839.900
			FUP	2,804	76	1,268.585			2,274.200

จากตารางที่ 4.4 จะเห็นว่าเมื่อเราได้ทำการกำหนดค่าจำนวนของฐานข้อมูลเดิมเท่ากับ 10,000 ทรานแซคชั่น จำนวนของฐานข้อมูลใหม่เท่ากับ 6,000 ทรานแซคชั่น และขนาดของตารางแฮชเท่ากับ 100 โดยจะทำการสุ่มค่า Minimum Support เพื่อใช้ในการทดลองทั้งหมด 7 ค่า คือ 0.01, 0.02, 0.03, 0.04, 0.05, 0.06 และ 0.07 นั้น จากผลการทดลองเราจะเปรียบเทียบประสิทธิภาพในการทำงานของอัลกอริทึมทั้ง 2 ในเชิงของจำนวน Candidate Itemsts และเวลาในการประมวลผลทั้งหมด จะเห็นว่าค่า Minimum Support ที่เพิ่มมากขึ้นเมื่อเทียบภายในอัลกอริทึมเดียวกันจะเห็นว่าจำนวนของ Candidate Itemsets จะมีจำนวนลดลง เนื่องมาจากค่าสนับสนุนต่ำสุดที่มีค่าที่สูงขึ้นนั่นเอง ซึ่งการกำหนดค่า Minimum Support ที่มีค่ามากเกินไปอาจจะทำให้ Candidate Itemsets บางตัวที่มีคาวมสำคัญอาจจะถูกตัดออกไปด้วยได้ ทำให้กฎความสัมพันธ์ที่ได้อาจจะไม่ใช่กฎความสัมพันธ์ที่ดีที่สุดเสมอไป เราจึงควรกำหนดค่า Minimum Support ให้มีความเหมาะสมกับจำนวนของข้อมูล เพราะถ้ากำหนดค่า Minimum Support น้อยเกินไปก็จะทำให้เสียเวลาในการประมวลผลนานมาก เนื่องจากค่าไม่สามารถตัดค่า Candidate Itemsets ในแต่ละรอบให้น้อยลงได้เลย ทำให้จำนวนของ Candidate Itemsets ในแต่ละ k-Itemsets นั้นมีขนาดที่ใหญ่มาก ทำให้ต้องใช้เวลาในการประมวลมากตามไปด้วย แต่ถ้าเราจะมาเปรียบเทียบระหว่างอัลกอริทึมทั้ง 2 ก็จะเห็นว่าอัลกอริทึม HFUP สามารถลดจำนวน Candiadte Itemsets ได้มากกว่าไม่ว่าจะกำหนดค่า Minimum Support เท่าไหร่ก็ตาม แต่ถ้าเปรียบเทียบในเชิงของเวลาจะเห็นว่าเวลาในการทำงานของอัลกอริทึม HFUP จะมากกว่าอัลกอริทึม FUP เมื่อกำหนดค่า Minimum Support เท่ากับ 0.01 – 0.05 แต่เมื่อค่า Minimum Support เท่ากับ 0.06 และ 0.07 เวลาในการทำงานของอัลกอริทึม HFUP จะน้อยกว่าเวลาในการทำงานของ FUP ซึ่งที่ผลการทดลองเป็นเช่นนี้เพราะอาจจะมีสาเหตุเกี่ยวเนื่องมาจากการกำหนดค่าของตัวแปรอื่นๆ ประกอบด้วยก็เป็นได้ เช่น ขนาดของตารางแฮชที่เรากำหนดไว้เท่ากับ 100 เป็นต้น จะเห็นว่าถ้าเรามาดูช่องเวลาในการสแกน Support ของทั้ง 2 อัลกอริทึม ค่า Minimum Support ที่ทำให้เวลาในการทำงานของอัลกอริทึม HFUP มากกว่าอัลกอริทึม FUP นั้นจะใช้เวลาในการสแกนค่า Support ใกล้เคียงกัน ซึ่งเป็นผลมาจากการลดจำนวน Candidate Itemsets โดยใช้ค่าจากตารางแฮชนั้นไม่มีประสิทธิภาพเท่าที่ควรนั่นเอง

4.4 สรุปและวิเคราะห์ผลการทดลอง

เนื่องจากในการค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิมนั้นอัลกอริทึม HFUP จะใช้ Hash Function มาช่วยในการลดจำนวนของ Candidate Itemsets ของจำนวน 2-Itemsets ทำให้เวลาในการทำงานลดลงเมื่อเทียบกับวิธีการค้นหากฎความสัมพันธ์โดยใช้อัลกอริทึม FUP เนื่องจากการค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิมนั้นอัลกอริทึม FUP ยังใช้วิธีการค้นหาแบบเดียวกับอัลกอริทึมอะพิโอริอยู่นั่นเอง และเมื่อมีการเพิ่มข้อมูลเข้ามาอัลกอริทึม HFUP ก็จะนำวิธีการค้นหา

แบบเดียวกับอัลกอริทึม FUP มาใช้ร่วมกับ Hash Function ด้วย ซึ่งจะเห็นว่าประสิทธิภาพในการประมวลผลข้อมูลนั้นจะใช้เวลาในการทำงานลดลง แต่จากการทดลองจะเห็นว่าขนาดของตารางแฮชมีผลต่อเวลาในการประมวลผลของอัลกอริทึม HFUP เป็นอย่างมาก

จากผลการทดลองของทั้ง 3 การทดลอง สามารถสรุปได้เป็น 2 ปัจจัยหลักๆ ดังนี้

1. สรุปโดยดูจากค่า Minimum Support ที่แตกต่างกัน

จากผลการทดลองจะเห็นว่าเมื่อค่า Minimum Support มีค่าที่เพิ่มมากขึ้นเมื่อเทียบกับขนาดของตารางแฮชที่เท่ากัน จะทำให้เวลาในการทำงานลดลง และจำนวนของ Candidate Itemset มีจำนวนลดลงด้วย เมื่อเทียบกับค่า Minimum Support ที่น้อยกว่า เนื่องมาจากค่า Minimum Support ที่มากขึ้นสามารถตัดค่า Candidate Itemset เพื่อที่จะกลายเป็น Large Itemset ได้มากกว่าตามเงื่อนไขที่ว่า Candidate Itemset ที่จะสามารถกลายเป็น Large Itemseet ได้นั้นจะต้องมีค่า Support Factor มากกว่าหรือเท่ากับค่า Minimum Support นั่นเอง จากเงื่อนไขนี้เมื่อค่า Minimum Support มากขึ้นอาจจะส่งผลให้กฎที่มีความสำคัญถูกตัดออกไปตามเงื่อนไขได้ ทำให้การนำข้อมูลเหล่านี้ไปใช้ไม่เกิดประโยชน์สูงสุดต่อธุรกิจ

2. สรุปโดยดูจากค่าขนาดของตารางแฮชที่มีขนาดแตกต่างกัน

จากการทดลองจะเห็นว่าเมื่อขนาดของตารางแฮชแตกต่างกันเมื่อเทียบกับค่า Minimum Support ที่เท่ากันนั้น เราไม่สามารถสรุปได้อย่างตายตัวเหมือนค่า Minimum Support ว่าขนาดของตารางแฮชที่น้อยกว่า หรือมากกว่าจะส่งผลให้ประสิทธิภาพในการทำงานของอัลกอริทึม HFUP มีประสิทธิภาพสูงสุดได้ เนื่องจากค่าที่จะส่งผลต่อการเปลี่ยนแปลงตำแหน่งของตารางแฮชในแต่ละ Itemsets นั้นนอกจากขนาดของตารางแฮชแล้ว น่าจะยังมีปัจจัยอื่นๆ เข้ามาเกี่ยวข้องอีก ซึ่งในงานวิจัยนี้ไม่ได้ทำการทดลองจนสามารถหาข้อสรุปได้ว่าต้องทำการกำหนดขนาดของตารางแฮชเท่าไหร่จึงจะทำให้อัลกอริทึม HFUP มีประสิทธิภาพในการทำงานสูงที่สุด แต่ถ้าเราสังเกตเฉพาะเวลาที่ Candidate 2-Itemsets ทำการสแกนหาค่า Support Factor ของแต่ละ Itemsets เพื่อนำไปเปรียบเทียบกับค่า Minimum Support ในการสร้าง Large 2-Itemsets ของทั้งอัลกอริทึม HFUP และอัลกอริทึม FUP นั้นจะเห็นว่าขนาดของตารางแฮชที่ส่งผลให้เวลาในการทำงานของอัลกอริทึม HFUP มากกว่าเวลาในการทำงานของอัลกอริทึม FUP นั้นไม่สามารถลดจำนวนของ Candidate 2-Itemsets หรือลดได้น้อยมาก เนื่องจากเวลาในการสแกนค่า Support Factor ของอัลกอริทึมทั้งสองจะถือว่ามีความใกล้เคียงกันมากนั่นเอง แต่ถ้าเราสังเกตเวลาในการสแกนค่า Support Factor ของขนาดตารางแฮชที่ทำให้เวลาในการทำงานของอัลกอริทึม HFUP ลดลงนั้น จะเห็นว่าเวลาในการสแกนค่า Support Factor ของอัลกอริทึม HFUP น้อยกว่าเวลาในการสแกนค่า Support Factor ของอัลกอริทึม FUP มาก ซึ่งน่าจะเป็นเพราะตารางแฮชได้ทำการลดจำนวนของ Candidate 2-Itemsets ได้เป็นจำนวนมากนั่นเอง

จึงสามารถสรุปได้ว่าเมื่อนำวิธีการค้นหาแบบอัลกอริทึม FUP เดิมมาประยุกต์ใช้งานร่วมกับ Hash Function นั้นจะทำให้ประสิทธิภาพในการทำงานของอัลกอริทึม FUP พัฒนามากขึ้นกว่าเดิม แต่ต้องขึ้นอยู่กับเงื่อนไขในการกำหนดขนาดของตารางแฮชให้เหมาะสมกับข้อมูลที่จะใช้ในการทดลอง การกำหนดค่า Minimum Support และยังรวมไปถึงลักษณะของข้อมูลที่ทำการสุ่มมาใช้ในการทดลองนั้นว่ามีการกระจายตัวของข้อมูลอย่างเหมาะสมหรือไม่ เพราะปัจจัยที่กล่าวมาจะส่งผลต่อทั้งเวลาในการทำงาน และความถูกต้องของกฎความสัมพันธ์ที่เราหามาได้ด้วยว่า กฎความสัมพันธ์นั้นเป็นกฎดามสัมพันธ์ที่ดีที่สุดสำหรับชุดข้อมูลนั้นๆ หรือไม่ ซึ่งมันจะส่งผลกระทบเมื่อเรานำกฎความสัมพันธ์ที่หามาได้นี้ไปใช้กับการประกอบธุรกิจจริงๆ ก็ได้

บทที่ 5
สรุปผลการวิจัย และข้อเสนอแนะ

การทำวิจัยนี้จัดทำขึ้นเพื่อนำเสนอให้เห็นถึงประโยชน์ของการนำทฤษฎีดาต้าไมนิ่งมาประยุกต์ใช้กับธุรกิจ โดยการนำไปใช้ในการวิเคราะห์พฤติกรรมการซื้อสินค้าของลูกค้าเพื่อหาความสัมพันธ์ต่างๆ และสามารถนำกฎความสัมพันธ์ที่ได้ไปประยุกต์ใช้ในการสร้างแผนกลยุทธ์ทางการตลาดเพื่อทำให้ธุรกิจได้รับกำไรสูงสุด ซึ่งอัลกอริทึมที่นิยมใช้ในการค้นหากฎความสัมพันธ์ (Association Rule Discovery) คืออัลกอริทึมอะพิโอริ แต่เนื่องจากข้อมูลที่ใช้ในการวิเคราะห์เพื่อหากฎความสัมพันธ์นั้นจะมีการเปลี่ยนแปลงอยู่ตลอดเวลา เราจึงต้องทำการวิเคราะห์เพื่อหากฎความสัมพันธ์ใหม่ให้เหมาะสมกับข้อมูลที่มีการเปลี่ยนแปลงอยู่เสมอ เพื่อวางแผนกลยุทธ์ให้เหมาะสมกับ ณ ช่วงเวลานั้นๆ ให้มากที่สุด ซึ่งอัลกอริทึมอะพิโอริไม่เหมาะกับการประมวลผลเมื่อมีการเพิ่มข้อมูลเข้ามา เนื่องจากอัลกอริทึมอะพิโอริจะต้องทำการรวมข้อมูลเดิมและข้อมูลใหม่เข้าไว้ด้วยกันก่อน เพื่อที่จะทำการค้นหากฎความสัมพันธ์ของข้อมูลทั้งหมดใหม่ โดยที่เราจะไม่สามารถนำกฎความสัมพันธ์ที่หาได้จากข้อมูลเดิมมาใช้ให้เกิดประโยชน์ได้ ทำให้ต้องเสียเวลาในการทำงานเพื่อค้นหากฎความสัมพันธ์ใหม่ทั้งหมดอีกรอบ เราจึงทำการศึกษาอัลกอริทึม FUP เพิ่มเติม เนื่องจากอัลกอริทึม FUP นี้จะถูกนำมาใช้ในการประมวลผลเมื่อมีข้อมูลใหม่เพิ่มเข้ามา โดยในการค้นหากฎความสัมพันธ์ของข้อมูลเดิมเราสามารถใช้อัลกอริทึมอะพิโอริเหมือนเดิมได้ ในการวิจัยนี้เราจะนำเสนออัลกอริทึม HFUP ซึ่งเป็นการประยุกต์การทำงานร่วมกันระหว่างอัลกอริทึม DHP กับอัลกอริทึม FUP เพื่อเพิ่มประสิทธิภาพในการทำงานสำหรับข้อมูลที่มีการเปลี่ยนแปลงอยู่ตลอดเวลา และลดเวลาในการประมวลผลเมื่อเทียบกับอัลกอริทึม FUP จากผลการทดลองจะเห็นว่าอัลกอริทึม HFUP สามารถลดเวลาในการประมวล และลดจำนวนของ Candidate Itemsets ลงเมื่อเทียบกับอัลกอริทึม FUP แต่จะขึ้นอยู่กับการกำหนดขนาดของตารางแฮชให้เหมาะสมกับจำนวนและลักษณะการกระจายตัวของข้อมูลด้วย เพราะถ้าเรากำหนดขนาดของตารางแฮชน้อยเกินไปจะทำให้เกิดการชนกันของข้อมูลในแต่ละช่องของตารางแฮชมากเกินไป ทำให้ไม่สามารถลดจำนวนของ Candidate Itemsets ได้ซึ่งจะส่งผลให้เวลาในการทำงานเพิ่มมากขึ้นด้วยเมื่อเทียบกับอัลกอริทึม FUP แต่ถ้ากำหนดขนาดตารางแฮชมากเกินไปเราอาจจะเข้าใจผิดว่าถ้าขนาดของตารางแฮชมากน่าจะทำให้เกิดการกระจายตัวของข้อมูลไปในแต่ละช่องของตารางแฮช ซึ่งน่าจะส่งผลให้เราสามารถลดจำนวนของ Candidate Itemsets ลงได้ แต่จริงๆ แล้วค่าที่มีผลต่อตำแหน่งของตารางแฮชคือค่าที่ได้จากการคำนวณผ่าน Hash Function ต่างหาก ดังนั้นเราจึงต้องทำการศึกษาเพิ่มเติมว่าเราจะหาขนาดของตารางแฮชให้เหมาะสมกับข้อมูลแต่ละชุดได้อย่างไรเพื่อให้ประสิทธิภาพของอัลกอริทึม HFUP เพิ่มมากขึ้น

5.1 ปัญหาและข้อเสนอแนะในการทำวิจัยต่อไป

ในการจัดทำวิจัยนี้จะเห็นว่าอัลกอริทึม HFUP ที่เรานำเสนอนั้นสามารถใช้ลดขนาดของ Candidate k-Itemsets ของแต่ละทรานแซคชั่น ทำให้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม FUP เมื่อใช้กับฐานข้อมูลที่มีการเปลี่ยนแปลง ซึ่งอัลกอริทึม HFUP สามารถนำไปประยุกต์ใช้กับธุรกิจที่ต้องการวิเคราะห์ถึงข้อมูลพฤติกรรมของผู้บริโภคที่มีความเปลี่ยนแปลงอยู่ตลอดเวลาได้ เช่น ข้อมูลพฤติกรรมในการเลือกใช้บริการต่างๆ ของโทรศัพท์เคลื่อนที่ พฤติกรรมการเลือกซื้อสินค้า และการเลือกใช้บริการของโรงพยาบาลต่างๆ เป็นต้น แต่การทำงานของอัลกอริทึม HFUP จะมีประสิทธิภาพในการทำงานสูงสุดก็ต่อเมื่อเราสามารถกำหนดขนาดของตารางแฮชให้มีขนาดที่เหมาะสมกับจำนวนของข้อมูลที่เรามีได้ โดยที่ไม่เกิดการชนกันของข้อมูลในแต่ละช่องของตารางแฮชมากจนเกินไป ซึ่งอาจจะส่งผลให้การค้นหากฎความสัมพันธ์ของเราที่ได้ไม่ใช่กฎความสัมพันธ์ที่ดีที่สุด โดยเราจะนำเสนอวิธีการแก้ไขนี้ในโอกาสต่อไป

บรรณานุกรม

[1] Han J., M. Kamber, "**Data mining: Concepts and Techniques,**" Morgan Kaufmann Publishers, San Francisco, CA, pp. 227-256, 2006.

[2] R. Agrawal, R. Srikant, "**Fast Algorithm for Mining Association Rules,**" Proceedings of the 20th International Conference on Very Large Data Bases, pp.478-499, September 1994.

[3] David W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong, "**Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique,**" Proceedings of the 12th International Conference on Data Engineering, pp. 106-14, February 1996.

[4] J.S. Park, M.S. Chen, and P.S. Yu, "**An Effective Hash-Based Algorithm for Mining Association Rules,**" In Proc. 1995 ACM-SIGMOD International Conference Management of Data, San Jose. CA, May 1995.

[5] R. Agrawal, T. Imielinski, and A. Swami, "**Mining Association Rules between Sets of Items in Large Databases,**" Proceedings of ACM SIGMOD, pp. 207-216, May 1993.

ภาคผนวก

การค้นหากฎความสัมพันธ์โดยใช้อัลกอริทึมบนพื้นฐานแฮชสำหรับการเพิ่มข้อมูล

Association Rule Search Using A Hash-Based Fast Update Algorithm

แจ่มจันทร์ คงปาน (Jamjun Kongpan) และ วรพจน์ กรีสุระเดช (Worapoj Kreesuradej)
คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Nooi_6@hotmail.com, worapoj@it.kmitl.ac.th

บทคัดย่อ

เอกสารนี้จะนำเสนอการประยุกต์ใช้วิธีการที่จะใช้ในการการค้นหาข้อมูลที่มีความน่าสนใจจากฐานข้อมูลที่มีข้อมูลเพิ่มเข้ามาใหม่เพื่อสร้างเป็นกฎความสัมพันธ์ที่มีความสอดคล้องกับข้อมูลทั้งหมด ซึ่งเราจะนำเสนอ a hash-based fast update algorithm โดยที่อัลกอริทึมนี้ถูกปรับปรุงมาจากอัลกอริทึม FUP และอัลกอริทึม HFUP ยังใช้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม FUP ด้วย

คำสำคัญ: การค้นหากฎความสัมพันธ์ อัลกอริทึมเอฟยูพี อัลกอริทึมดีเอชพี

Abstract

This paper is concerned with applying an incremental association rule mining to extract interesting information from a dynamic database. The paper proposes a hash-based fast update algorithm. The algorithm is modified from FUP algorithm to deal with tracking candidate k-itemsets in each transaction is ineffective since the cardinality of such k-itemsets is very large. The proposed is shown that our algorithm has better running time than that of FUP.

Keyword: knowledge discovery fast update algorithm hashed-base algorithm incremental association rule.

1. บทนำ

ปัจจุบันการดำเนินธุรกิจต่างก็มีอัตราการแข่งขันกันสูง ทำให้นักธุรกิจจำนวนมากหันมาให้ความสำคัญกับการนำข้อมูลต่างๆ ในอดีตมาวิเคราะห์ เพื่อที่จะนำข้อมูลเหล่านี้ไปใช้ในการสร้างแผนกลยุทธ์ทางการตลาด เพื่อเพิ่มประสิทธิภาพและทำให้ธุรกิจได้รับผลกำไรเพิ่มมากขึ้น ซึ่งการค้นหากฎความสัมพันธ์ (Association Rule Discovery) ก็ถือเป็นเทคนิคหนึ่งของ Data mining ที่นักธุรกิจต่างให้ความสนใจในการนำเทคนิคนี้ไปช่วยในการค้นหากฎความสัมพันธ์ของข้อมูลในกลุ่มข้อมูลขนาดใหญ่ (Data Set) โดยจะนำไปวิเคราะห์ข้อมูลพฤติกรรมการซื้อสินค้าของลูกค้าเพื่อหาความสัมพันธ์ต่างๆ และสามารถนำมาสรุปเป็นกฎความสัมพันธ์เพื่อที่ผู้บริหารสามารถที่จะนำข้อมูลเหล่านี้ไปใช้ให้เกิดประโยชน์ได้

การค้นกากฎความสัมพันธ์ (Association Rule Discovery) เป็นการค้นหาความสัมพันธ์ระหว่างรายการในแต่ละรายการหรือกลุ่มของรายการที่ปรากฏขึ้นในฐานข้อมูล รูปแบบของกฎความสัมพันธ์ที่ได้จะอยู่ในรูปแบบ “IF X Then Y” โดยที่ X และ Y จะต้องเกิดขึ้นพร้อมกันภายในทรานแซคชั่นเดียวกัน โดยใช้สัญลักษณ์ X-->Y ซึ่งมีตัวแปรที่ใช้ในการวัดค่าอยู่ 2 ตัวแปร คือ ค่าสนับสนุน (Support Factor) และค่าความเชื่อมั่น (Confidence Factor) [1]

อัลกอริทึม Apriori [2] ถือเป็นอัลกอริทึมที่นิยมใช้ในการหากฎความสัมพันธ์ของข้อมูล แต่ถ้าฐานข้อมูลมีการเพิ่มข้อมูลเข้ามา หรือเกิดมีการเปลี่ยนแปลงข้อมูล อัลกอริทึม Apriori จะต้องนำข้อมูลทั้งหมดมารวมกันก่อน แล้วจึงจะสามารถนำข้อมูลทั้งหมดไปค้นหากฎความสัมพันธ์ใหม่ทั้งหมด โดยไม่สามารถนำกฎความสัมพันธ์ที่หาได้จากกลุ่มข้อมูลเก่าก่อนหน้า

มาใช้ให้เกิดประโยชน์ได้ ทำให้เสียเวลาในการทำงานเพื่อค้นหากฎความสัมพันธ์ใหม่ทั้งหมด

ดังนั้นจึงเกิดเทคนิค Incremental Association Rule Discovery เพื่อใช้ในการค้นหากฎความสัมพันธ์ในฐานข้อมูลที่มีการเปลี่ยนแปลง โดยใช้อัลกอริทึม FUP [3] มาช่วยในการค้นหากฎความสัมพันธ์ ซึ่งหลักการทำงานของ FUP คือจะสร้าง Candidate Itemset ขนาดเล็กเพื่อค้นหา New Large Itemsets และทำการปรับปรุงฐานข้อมูลโดยการนำข้อมูลเดิมมารวมเข้ากับข้อมูลใหม่ที่เพิ่มขึ้นเพื่อหา New Large Itemsets ทำให้การค้นหากฎความสัมพันธ์มีประสิทธิภาพมากขึ้น และลดเวลาในการค้นหากฎความสัมพันธ์ได้ดีกว่าอัลกอริทึม Apriori ซึ่งเรายังได้นำวิธีการค้นหากฎความสัมพันธ์แบบ Direct Hashing and Pruning (DHP) [4] มาปรับใช้ร่วมกับอัลกอริทึม FUP เพื่อเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ และลดเวลาในการค้นหาอีกด้วย

2. ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 นิยามเบื้องต้นของกฎความสัมพันธ์

$I = \{ i_1, i_2, \ldots, i_n \}$ คือกลุ่มของข้อมูลแต่ละตัวที่ใช้ในการหากฎความสัมพันธ์ เรียกว่าไอเทม (Items)

$D = \{ T_1, T_2, \ldots, T_m \}$ คือฐานข้อมูลที่ประกอบไปด้วยทรานแซคชั่น (T)

T คือเซตของไอเทมก็ต่อเมื่อ $T \subseteq I$

TID เป็นหมายเลขหรือรหัสประจำแต่ละทรานแซคชั่นที่ไม่ซ้ำกัน (unique identifier)

X คือเซตของไอเทมในแต่ละทรานแซคชั่นก็ต่อเมื่อ $X \subseteq T$

k-Itemset คือเซตของ Items ที่มีจำนวนสมาชิก k Items

โดยปกติแล้วกฎความสัมพันธ์สามารถจัดให้อยู่ในรูปของ $X \rightarrow Y$ คือ IF X THEN Y โดยที่ $X \subset I$ กับ $Y \subset I$ และ $X \cap Y = \phi$

2.2 Apriori Algorithm [2]

ขั้นตอนวิธีของ Apriori มีลักษณะการทำงานดังนี้คือ สร้าง Candidate Itemset ขึ้นมาก่อน โดยจะเริ่มจาก Candidate Itemsets ที่มีขนาดเท่ากับ k-Itemsets โดยที่ $k = \{1, 2, 3, \ldots, n\}$ ซึ่ง Candidate Itemsets ในระดับที่ k จะได้มาจากการ join กันระหว่าง Large Itemsets ในระดับที่ k-1 เช่น Itemsets {ABC} เป็นผลมาจากการนำ Itemset {AB} join กับ Itemset {AC} จากนั้นจึงทำการหา Large Itemets ในระดับที่ k โดยจะทำการอ่านข้อมูลจากฐานข้อมูลเพื่อทำการนับค่าสนับสนุน (Support Factor) ของ Candidate Itemsets แต่ละตัวในระดับที่ k นั้น จากนั้นจึงทำการตรวจสอบว่า Candidate Itemsets ใดมีค่าสนับสนุน (Support Factor) มากกว่าหรือเท่ากับค่าสนับสนุนน้อยสุด (min_sup) จะถือว่า Candidate Itemsets นั้นมีคุณสมบัติเป็น Large Itemsets ในระดับที่ k จากนั้นจะทำการสร้าง Candidate Itemsets หา Large itemsets ในระดับที่ k+1 ต่อไปโดยจะทำเช่นนี้ไปเรื่อยๆ จนกว่าจะไม่สามารถที่จะสร้าง Candidate Itemsets ในระดับที่ k+1 ได้

2.3 Fast Update Algorithm (FUP) [3]

FUP เป็นอัลกอริทึมที่พัฒนามาจากอัลกอริทึม Apriori โดยมีหลักการทำงานคือ จะทำการจัดเก็บ Large Itemsets ที่ค้นพบในฐานข้อมูลเดิม จากนั้นทำการบันทึกและตรวจสอบทรานแซคชั่นใหม่ที่เพิ่มเข้ามาในฐานข้อมูล อัลกอริทึม FUP จะสร้าง Candidate Set ขนาดเล็กเพื่อค้นหา New Large Itemsets และทำการปรับปรุงฐานข้อมูลโดยการนำข้อมูลในฐานข้อมูลเดิมมารวมเข้ากับข้อมูลใหม่ที่เพิ่มขึ้นเพื่อหา New Large Itemsets ทั้งหมดที่อยู่ในฐานข้อมูล

ขั้นตอนการทำงานของ Fast Update Algorithm (FUP) ดังนี้

1) นำ Itemset ที่ได้จากฐานข้อมูลที่เพิ่มขึ้น (db) มาหาค่า Support Factor ใหม่หลังจากทำการรวมฐานข้อมูลทั้งหมดไว้ด้วยกันแล้ว โดยที่ $X \in L_k$ แล้วนำมาตรวจสอบเงื่อนไข $X.support_{UD} \geq s(D + d)$ ถ้าเป็นไปตามเงื่อนไขจะถือว่าเป็น New Large k-Itemset (L'_k) ถ้าไม่เป็นไปตามเงื่อนไข จะถูกตัดออกจาก L_k

2) สร้าง C_k จากฐานข้อมูลที่เพิ่ม (db) โดยการหาค่า Support Factor ของ Itemset ที่ไม่อยู่ใน L_k แล้วนำมาตรวจสอบเงื่อนไข $X.support_d \geq s \times d$ ถ้าเป็นไปตามเงื่อนไขจะเก็บไว้ที่ C_k แต่ถ้าไม่เป็นไปตามเงื่อนไขจะถูกตัดออกจาก C_k

3) อ่านข้อมูลในฐานข้อมูล (DB) เพื่อนำค่า Support Factor มารวมกับค่า Support Factor ของฐานข้อมูลที่เพิ่มเข้ามา ของ Itemsets ใน C_k แล้วนำมาตรวจสอบเงื่อนไข $X.support_{UD} \geq s(D + d)$ ถ้าเป็นไปตามเงื่อนไขจะถือว่าเป็น New Large k-

Itemsets (L'_k) แต่ถ้าไม่เป็นไปตามเงื่อนไขจะถูกตัดออกจาก C_k และไม่ถือว่าเป็น L'_k

4) วนการทำการแบบนี้ไปเรื่อยๆ จนกระทั่งไม่สามารถหา L'_k ได้อีก

3. A Hash-Based Fast Update Algorithm

ในงานวิจัยนี้เราจะนำเสนออัลกอริทึม HFUP ซึ่งเป็นการนำเอาคุณสมบัติของอัลกอริทึม DHP มาใช้ร่วมกับอัลกอริทึม FUP เพื่อช่วยเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ให้เหมาะสมกับฐานข้อมูลที่มีการเปลี่ยนแปลง ซึ่งจะทำให้เราสามารถนำกฎความสัมพันธ์ที่ได้นี้ไปใช้ให้เกิดประโยชน์สูงสุดต่อไป โดยรายละเอียดของ HFUP อัลกอริทึมมีขั้นตอนต่างๆ ดังนี้

1) ในขั้นตอนแรกเรา สามารถหากฎความสัมพันธ์ โดยการใช้วิธีของ HFUP ซึ่งจะทำการสแกนฐานข้อมูลรอบแรกเพื่อนับค่า Support Factor ของ 1-Itemsets (F_1) และก็จะสร้างเซตย่อยของ 2-Itemsets ทั้งหมดที่เป็นไปได้ก็จะนำเซตย่อยของ 2-Itemses มาผ่าน Hash Function ทีละชุดเพื่อหา Hash Address แล้วจะบวก 1 เพิ่มเข้าไปใน Bucket ของ Hash Table พร้อมกับการเก็บค่า Support Factor ของ F_2 จากนั้นก็จะสร้าง L_1 จากค่า Support Factor ของ 1-Itemsets ที่มีค่ามากกว่าหรือเท่ากับ min_sup ตามภาพที่ 1

Input: (1) DB: ฐานข้อมูลเดิม; (2) L_k: เซตทั้งหมดของ large k-itemsets ในฐานข้อมูลเดิม ที่ซึ่ง k = 1,...,r; (3) db: ข้อมูลที่เพิ่มขึ้น; และ (4) s: ค่าสนับสนุนน้อยสุด

Output: L': เซตของ large itemsets ทั้งหมดใน DB $\cup$ db.

```
/* Part 1 */
    s = a minimum support;
    set all the buckets of H2 to zero;  /* สร้างตารางแฮช*/
    forall transaction t ∈ D do begin
        insert and count 1-Items occurrences in a hash tree;
        forall 2-Subsets x of t do
            F2(x) ++; /* เก็บค่า Support ของแต่ละ Subsets */
            H2[h2(x)] ++;
    end
L1 = {c|c.count ≥ s, c is in a leaf node of the hash tree};
Dk = D;          /* database for large k-Itemsets */
for (int k=2; L[k-1] != ϕ; k++) {
    /* make a hash table */
    gen_candidate(Lk-1, Hk, Ck);
    set all the buckets of Hk+1 to zero;
    Dk+1 = ϕ;
    forall transactions t ∈ Dk do begin
        forall candidate k-Itemsets ∈ Ck do begin
            Fk+1(x) ++; /* นับค่า Support ของแต่ละ Ck */
            if (k=2) then
                all the 2-subsets of this transaction are
                generated and hashed into a hash table H2 in
                such a way that when a 2-subset is hashed to
                bucket i, the value of bucket i is increased by
                one; (Hk+1[hk+1(x)] ++;)
        end
    end
    if (k=2) then
        Ck = { x ∈ Ck| Hk[hk(x)] ≥ s};
    Lk = {x ∈ Ck| F2(x) ≥ s};
}
```

ภาพที่ 1: อัลกอริทึม HFUP ส่วนที่ 1

2) ในขั้นตอนถัดมาอัลกอริทึม HFUP ก็จะสร้าง C_k จากการนำ L_{k-1} มา join กันแล้วใช้ Hash Table ที่สร้างไว้ในรอบก่อนหน้านี้ช่วยในการกำหนดว่า Itemsets ชุดใดควรจะนำไปสร้างเป็น C_k จากนั้นก็จะนำค่า Support Factor ที่เรานับเก็บไว้ระหว่างสร้าง Hash Table มาเช็คดูว่ามี Itemsets ใดใน C_k ที่มีค่า Support Factor มากกว่าหรือเท่ากับ min_sup เพื่อนำ Itemsets เหล่านั้นมาสร้างเป็น L_k และสร้าง Hash Table ที่จะนำไปใช้ในการสร้าง C_k ในรอบถัดไป และก็ทำการเก็บค่า Support Factor ไปด้วยในระหว่างการสร้าง Hash Table เพื่อนำมาใช้ในขั้นตอนการคัดเลือก C_k ไปเป็น L_k โดยขั้นตอนนี้จะทำงานวนลูปไปเรื่อยๆ จนกระทั่งไม่สามารถสร้าง L_k ได้อีก ตามภาพที่ 1

จากนั้นเมื่อมีข้อมูลเพิ่มเข้ามาในฐานข้อมูลเราก็จะทำการตรวจสอบกฎความสัมพันธ์เดิมว่ายังสามารถใช้ได้อีกหรือไม่ และทำการหากฎความสัมพันธ์ใหม่เพิ่ม โดยมีขั้นตอนการทำงานดังนี้

1) ขั้นตอนที่ 1 ทำการอ่านข้อมูลในส่วนที่เพิ่มเข้ามาเพื่อปรับปรุงค่า Support Factor ของแต่ละ Item ที่เราเก็บไว้ที่ F_k จากนั้นก็ทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$

2) ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น ที่ไม่ได้อยู่ใน Large Itemsets (L_1) เพื่อหา Candidate Itemset (C_1) ไว้ใช้ในการหากฎความสัมพันธ์ใหม่ โดยทำการหาค่า Support ของ Itemsets ใน Candidate Itemsets และทำการตรวจสอบตามเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_1$

3) ขั้นตอนที่ 3 ทำการอ่านค่า Support Factor ในตาราง F_1 ของ DB เพื่อดูจำนวนข้อมูลของ Itemsets ใน Candidate Itemsets (C_1) มาปรับปรุงร่วมกับค่า Support Factor ใน db เพื่อหากฎความสัมพันธ์ใหม่ โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ เพื่อทำการค้นหา New Large Itemsets (L'_1)

Method: The 1st iteration: /* find L'_1, the set of all large 1-itemsets in $DB \cup db$ */

$W = L_1$; $C = \phi$; $L'_1 = \phi$; $P = \phi$;

/* W: winners, C: candidate sets, L'_1: initialized, P: for optimization */

for_all $T \in db$ do /* scan db */

for_all 1-itemset $X \subseteq T$ do {

if $X \in W$ then $X.support_d$++;

else {

if $X \notin C$ then { $C = C \cup \{X\}$; $X.support_d = 0$; }

/*init the support cont and add X into C */

$X.supprt_d$++; }

};

for_all $X \in W$ do /* put winners into L'_1*/

if $X.support_{UD} \geq s \times (D + d)$

then $L'_1 = L'_1 \cup \{X\}$;

for_all $X \in C$ do /*prune candidate sets in C*/

if $X.support_d < s \times d$

then { $C = C - \{X\}$; $P = P \cup \{X\}$; }

/* P will be used for optimization */

for_all $T \in DB$ do /* scan DB */

for_all 1-itemset $X \subseteq T$ do {

if $X \in C$ then $X.support_D$++;

if $X \in P$ then removes X from T;

/* Transaction T is reduced */

};

for_all $X \in C$ do /* put winners into L'_1 */

ภาพที่ 2: อัลกอริทึม HFUP ส่วนที่ 2

The k-th iteration: /* for $k = 2$ or larger, repeat this program fragment to find L'_k, the set of all large k-itemsets in the updated database, until either L'_k returned is empty or $db = \phi$ */

$W = L_k$; $L'_k = \phi$; /* W: winners; L'_k initialized */

C = **apriori_gen1**(L'_{k-1}) - L_k; /* refer to [2] */

/* the size-k candidate sets */

for_all k-itemset $X \in W$ do /* prune off losers in W */

for_all (k-1)-itemset $Y \in L_{k-1} - L'_{k-1}$ do

if $Y \subseteq X$ then { $W = W - \{X\}$; break; }

for_all $T \in db$ do { /* scan db */

for_all $X \in$ Subset(W, T) do $X.support_d$++;

/* Subset(W,T) returns all the sets in W contained in T */

for_all $X \in$ Subset(C, T) do $X.support_d$++;

/* find support of all $X \in C$ */

Reduce_db (T);

/* Some items in transactions in db can be removed */

}

for_all $X \in W$ do /* put the winners from W into L'_k */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

for_all $X \in C$ do

/* prune candidate sets in C. Since if the k-th iteration for k = 2, **DHP** used to reduce the number of candidate 2-itemsets */

if $X.support_d < s \times d$ then $C = C - \{X\}$;

for_all $T \in DB$ do { /* scan DB */

for_all $X \in$ Subset(C, T) do $X.support_D$++;

Reduce_Db(T); }

/* Some items in transactions in DB can be removed */

for_all $X \in C$ do

/* put the winners from C into L'_k. Since if the k-th iteration for k = 2, **DHP** used to reduce the number of candidate 2-itemsets */

if $X.support_{UD} \geq s \times (D + d)$

then $L'_k = L'_k \cup \{X\}$;

return L'_k. /* The end of the k-th iteration */

ภาพที่ 3: อัลกอริทึม HFUP ส่วนที่ 3

หลังจากที่เราได้ New Large Itemsets มาแล้วให้ทำการสแกนหาทรานแซคชั่นในฐานข้อมูลใหม่ที่มี $X \in L'_1$ ทั้งหมดมาสร้างเซตย่อยของ 2-Itemsets ทั้งหมดที่เป็นไปได้ จากนั้นนำเซตย่อยของ 2-Itemses มาผ่าน Hash Function ทีละชุดเพื่อหา Hash Address แล้วจะบวก 1 เพิ่มเข้าไปใน Bucket ของ Hash Table ที่เราได้ทำการสร้างไว้แล้วจากการหา Large Itemsets ของฐานข้อมูลเดิม เพื่อที่เราจะสามารถนำไปใช้ในการลดขนาดของ Candidate Itemsets ในขั้นตอนถัดไป โดยจะการทำงานจะวนซ้ำแบบนี้ไปเรื่อยๆ จนไม่สามารถหา New Large Itemsets (L'_1) ได้อีก จากนั้นก็จะนำ New Large Itemsets (L'_1) ทั้งหมดมาสร้างเป็นกฎความสัมพันธ์ โดยที่เราจะถือกฎความสัมพันธ์ที่มีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นน้อยสุดนั้นเป็นกฎความสัมพันธ์ที่มีความน่าเชื่อถือ สามารถนำไปใช้เพื่อให้เกิดประโยชน์สูงสุดได้ (ตามภาพที่ 2 และ 3)

4. ตัวอย่างการทดลอง

ในขั้นตอนแรกเรา สามารถหากฎความสัมพันธ์ โดยใช้วิธี HFUP ซึ่งจะทำการสแกนฐานข้อมูล เดิม (ภาพที่ 4) โดยจะทำตามขั้นตอนที่ได้อธิบายไว้ข้างต้นตามภาพที่ 1 ซึ่งสามารถสร้างเป็นตาราง Hash Table ดังภาพที่ 5

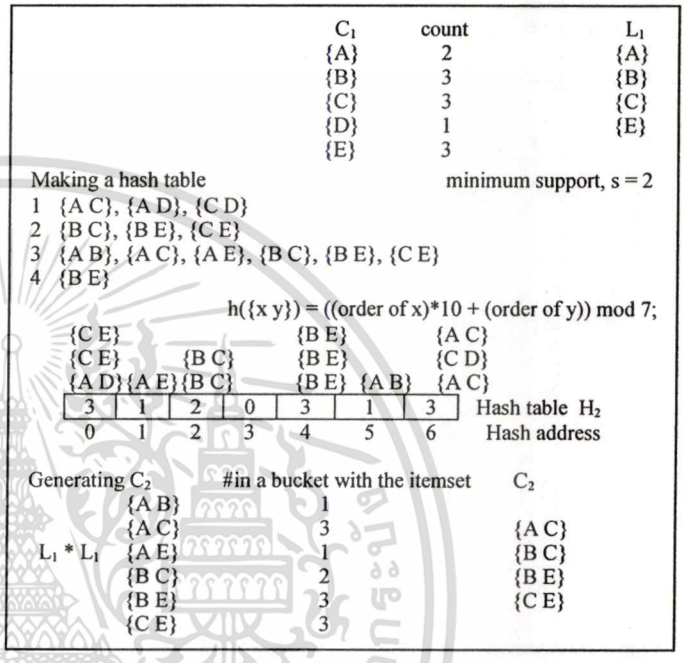
Database *DB*

TID	Items
1	A C D
2	B C E
3	A B C E
4	B E

Database *db*

TID	Items
1	A E F
2	C F
3	A C E

ภาพที่ 4: แสดงฐานข้อมูลเดิม (DB) และข้อมูลที่เพิ่มเข้ามา (db)



ภาพที่ 5: แสดงการสร้าง Hash Table และ C_2 ด้วย HFUP

จากภาพที่ 5 เราจะได้ L_1 คือ {A, B, C, E} และ L_2 คือ {(AC), (BC), (BE), (CE)} และจะทำการวนหา Large k-Itemsets ไปเรื่อยๆ จนไม่สามารถหา Large k-Itemsets ได้อีกแล้ว หลังจากนั้นจึงนำ Large k-Itemsets ทั้งหมดมาสร้างเป็นกฎความสัมพันธ์ โดยที่กฎความสัมพันธ์ที่จะสามารถนำไปใช้ประโยชน์ได้ต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นน้อยสุด เมื่อมีข้อมูลใหม่เพิ่มเข้ามาเราจะทำการหา New Large Itemsets ตามขั้นตอนดังภาพที่ 2 และ 3 ดังนี้

- ทำการอ่านข้อมูลใน New Transaction (ภาพที่ 4) เพื่อนับจำนวนข้อมูลที่เกิดขึ้นของแต่ละ Large Itemsets (L_1) = {A, B, C, E} ใน New Transaction เพื่อนำมาปรับปรุงค่า Support Factor ของแต่ละ Large Itemsets (L_1) ในฐานข้อมูลทั้งหมด เพื่อดูว่ากฎความสัมพันธ์เดิมยังเป็นไปตามเงื่อนไขหรือไม่ เมื่อฐานข้อมูลมีการเปลี่ยนแปลง โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ เมื่อค่า minimum

support = 50% เช่น ค่า Support Factor ของ Item(A) ในฐานข้อมูลทั้งหมดมีค่า (2+2) = 4 เมื่อเทียบกับค่า min_sup ของฐานข้อมูลรวมซึ่งหาได้จากนำทรานแซคชั่นทั้งหมดคูณกับค่า % ของ min_sup จากตัวอย่างจะได้ค่า min_sup รวมเท่ากับ (4+3) × 50% = 3.5 แต่เราจะปัดให้ค่าเป็นจำนวนเต็มเสมอ ซึ่งในที่นี้ค่า min_sup จะมีค่าเท่ากับ 4 และเมื่อนำมาตรวจสอบตามเงื่อนไขข้างต้นจะเห็นว่าเงื่อนไขเป็นจริงจะเห็นว่า $A \in L'_1$ ทำการตรวจเช็คเช่นนี้ไปเรื่อยๆ จนครบทุก Item ใน L_1

- ขั้นตอนที่ 2 ทำการอ่านข้อมูลใน db เพื่อหา Itemsets ใหม่ที่เกิดขึ้น ที่ไม่ได้อยู่ใน Large Itemsets (L_1) จากภาพที่ 4 คือ {F} เพื่อนำ Itemset นี้มาหา Candidate Itemset (C_1) ไว้ใช้ในการหากฎความสัมพันธ์ใหม่ โดยทำการหาค่า Support Factor ของ Itemsets ใน Candidate Itemsets (C_1) และทำการตรวจสอบตามเงื่อนไข $X.support_d \geq s \times d$ โดยที่ $X \in C_1$เช่น จากภาพที่ 4 จะเห็นว่า Item(F) นั้นมีค่า Support Factor เท่ากับ 2 และค่า min_sup ของ db สามารถคำนวณได้ดังนี้ $3 \times 50\% = 1.5$ ซึ่งจะถือว่ามีค่าเท่ากับ 2 นั่นเอง ดังนั้นจะเห็นว่า $F \in C_1$ เนื่องจากค่า Support Factor ของ Item(F) มีค่าเท่าค่า min_sup ซึ่งทำให้เงื่อนไขเป็นจริง เราจะทำการตรวจเช็คอย่างนี้จนครบทุก Itemsets ที่ไม่ได้เป็นสมาชิกของ L_1

- ขั้นตอนที่ 3 ทำการอ่านค่า Support Factor ในตาราง F_1 ของ DB เพื่อดูจำนวนข้อมูลของ Itemsets ใน Candidate Itemsets (C_1) = {F} มาปรับปรุงร่วมกับค่า Support Factor ใน db เพื่อหากฎความสัมพันธ์ใหม่ โดยทำการตรวจสอบตามเงื่อนไข $X.support_{UD} \geq s(D + d)$ โดยที่ $X \in L_1$ เพื่อทำการค้นหา New Large Itemsets (L'_1) เช่น จะเห็นว่า Item(F) มีค่า Support Factor ของฐานข้อมูลรวมเท่ากับ 2 และค่า min_sup ของฐานข้อมูลรวมทั้งหมดสามารถคำนวณได้เท่ากับ 4 (ตามตัวอย่างการคำนวณข้างต้น) ดังนั้น $F \notin L'_1$ เนื่องจากเงื่อนไขไม่เป็นจริง

- โดยที่เราจะทำการวนหา New Large k-Itemsets อย่างนี้ไปเรื่อยๆ จนไม่สามารถหาค่า New Large k-Itemsets ได้อีก จะนำ New Large k-Itemsets ทั้งหมดที่หาได้มาสร้างเป็นกฎความสัมพันธ์ โดยจะนำกฎความสัมพันธ์ที่ได้มาเปรียบเทียบค่าความเชื่อมั่นว่ามีค่ามากกว่าหรือเท่ากับค่าความเชื่อมั่นน้อยสุด (min_conf) หรือไม่ ถ้าเป็นจริงจะถือว่ากฎความสัมพันธ์นั้นมีความน่าเชื่อถือ ดังนั้นสามารถนำกฎความสัมพันธ์นั้นไปใช้ให้เกิดประโยชน์สูงสุดได้

5. สรุป

ในเอกสารนี้จะเห็นว่า HFUP ที่เรานำเสนอนั้นสามารถใช้ลดขนาดของ Candidate k-Itemsets ของแต่ละทรานแซคชั่น ทำให้เวลาในการประมวลผลน้อยกว่า FUP เมื่อใช้กับฐานข้อมูลที่มีการเปลี่ยนแปลง ซึ่ง HFUP สามารถนำไปประยุกต์ใช้กับธุรกิจที่ต้องการวิเคราะห์ถึงข้อมูลพฤติกรรมของผู้บริโภคที่มีความเปลี่ยนแปลงอยู่ตลอดเวลาได้ เช่น ข้อมูลพฤติกรรมในการเลือกใช้บริการต่างๆ ของโทรศัพท์เคลื่อนที่ พฤติกรรมการเลือกซื้อสินค้า และการเลือกใช้บริการของโรงพยาบาลต่างๆ เป็นต้น แต่ DHP จะมีประสิทธิภาพสูงสุดก็ต่อเมื่อเราสามารถกำหนดจำนวนช่องของตารางแฮชให้มีค่าเหมาะสมกับจำนวนข้อมูลที่เรามีได้โดยที่ไม่เกิดการชนกันของข้อมูลในแต่ละช่องของตารางแฮชมากจนเกินไป ซึ่งอาจจะส่งผลให้การค้นหากฎความสัมพันธ์ของเราที่ได้ไม่ใช่กฎที่ดีที่สุด โดยเราจะนำเสนอวิธีการแก้ไขนี้ในโอกาสต่อไป

6. เอกสารอ้างอิง

[1] Han J., M. Kamber, "Data mining: Concepts and Techniques," *Morgan Kaufmann Publishers, San Francisco, CA,* pp. 227-256, 2006.

[2] R. Agrawal, R. Srikant, "Fast Algorithm for Mining Association Rules," *Proceedings of the 20th International Conference on Very Large Data Bases,* pp.478-499, September 1994.

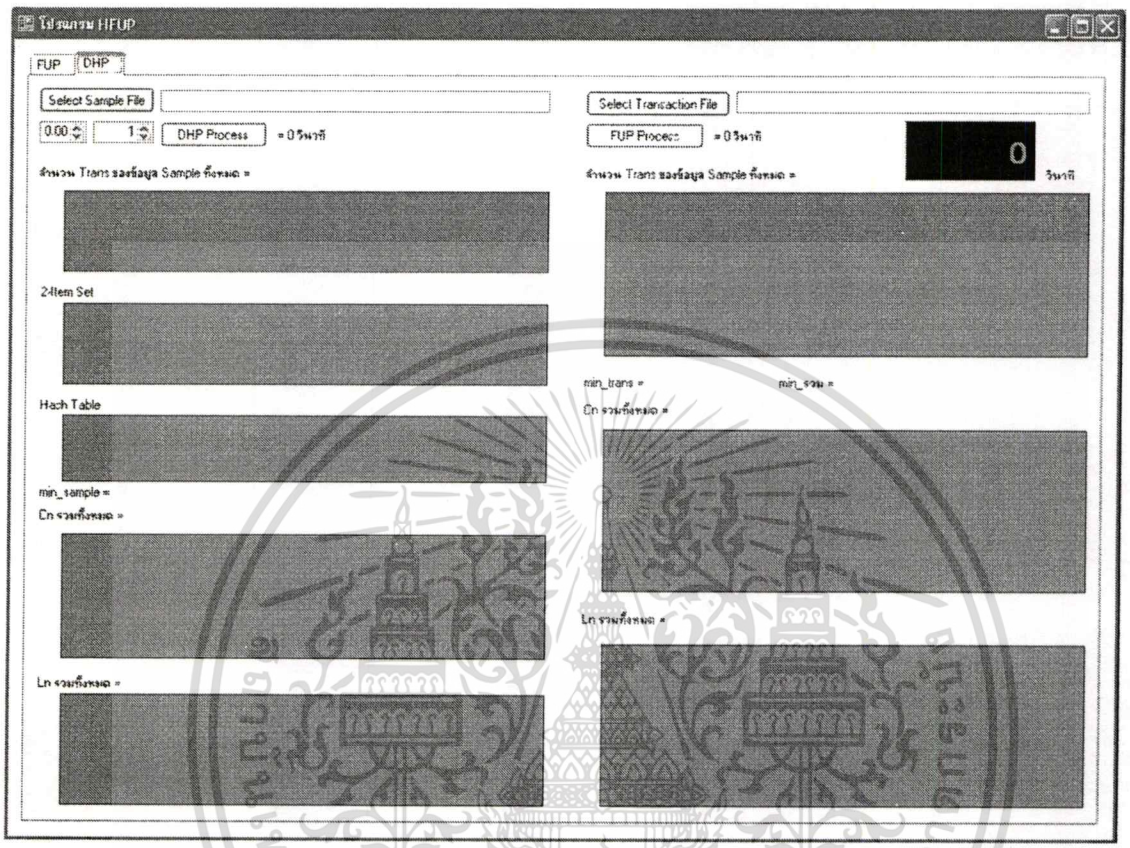
[3] David W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proceedings of the 12th International Conference on Data Engineering,* pp. 106-14, February 1996.

[4] J.S. Park, M.S. Chen, and P.S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," *In Proc. 1995 ACM-SIGMOD International Conference Management of Data,* San Jose. CA, May 1995.

[5] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proceedings of ACM SIGMOD,* pp. 207-216, May 1993.

ภาคผนวก ก.2

การ Implement A Hash-Based Fast Update Algorithm (HFUP)



รูปที่ ก.2.1 แสดงหน้าจอโปรแกรมในส่วนของอัลกอริทึม HFUP

ขั้นตอนการทำงานของโปรแกรม HFUP

1. คลิกที่ปุ่ม Select Sample File เพื่อทำการเลือกไฟล์ฐานข้อมูลเดิมที่ต้องการค้นหากฎความสัมพันธ์ของข้อมูล
2. กำหนดค่า Minimum Support
3. กำหนดขนาดของตารางแฮช
4. คลิกที่ปุ่ม DHP Process จะเป็นเริ่มต้นการค้นหากฎความสัมพันธ์ของชุดข้อมูลที่ทำการเลือกมาในข้อ 1
5. หลังจากเสร็จสิ้นกระบวนการค้นหากฎความสัมพันธ์ของชุดฐานข้อมูลเดิมแล้วนั้น จะได้ Large Itemsets ทั้งหมด เพื่อนำไปสร้างเป็นกฎความสัมพันธ์ต่อไป
6. หากมีข้อมูลใหม่เพิ่มเข้ามา และต้องการนำมาประมวลร่วมกับชุดฐานข้อมูลเดิมที่เราได้ทำการประมวลผลเพื่อค้นหากฎความสัมพันธ์เสร็จสิ้นไปแล้วนั้น เราจะทำการคลิก

ที่ปุ่ม Select Transaction File เพื่อเลือกไฟล์ของชุดฐานข้อมูลใหม่ที่ต้องการนำมาประมวลผลร่วมกับชุดฐานข้อมูล

7. คลิกที่ปุ่ม FUP Process เพื่อทำการประมวลเพื่อหา New Large Itemsets ทั้งหมด เมื่อมีชุดของฐานข้อมูลใหม่เพิ่มเข้ามา
8. เมื่อประมวลผลเสร็จเราก็จะได้ New Large Itemsets ทั้งหมด เพื่อนำไปสร้างเป็นกฎความสัมพันธ์ต่อไป

จากขั้นตอนการทำงานของโปรแกรมจะเห็นว่า Source Code ในการ Implement อัลกอริทึม HFUP จะแบ่งออกเป็น 2 ส่วนหลักๆ คือ DHP Process และ FUP Process ซึ่งเราได้แสดงรายละเอียดของโมดูลการทำงานต่างๆ ของทั้ง 2 Process ตามรูปที่ ก.2.2 และรูปที่ ก.2.3 และสามารถดูรายละเอียด Code ของแต่ละโมดูลย่อยๆ ดังนี้

ก.2.1 แสดง Code ของแต่ละโมดูลย่อยใน DHP Process

1. btnDHPProcess_Click() หลังจากที่เราเลือกไฟล์ที่ต้องการหากฎความสัมพันธ์ของข้อมูล กำหนดค่า Minimum Support และขนาดของตารางแฮชเรียบร้อย เราจะเริ่มต้นกระบวนการในหาค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิม โดยคลิกที่ปุ่ม DHP Process เพื่อเริ่มการประมวล

```
private void btnDHPProcess_Click(object sender, EventArgs e)
{
  if (nudMinDHP.Value == 0)
  {
    MessageBox.Show("กรุณากรอกค่า min_sup", "คำเตือน", MessageBoxButtons.OK,
                                                    MessageBoxIcon.Warning);
  }
  else if (txtSampleDHPDIR.Text == string.Empty)
  {
    MessageBox.Show("กรุณาเลือกไฟล์ตัวอย่างข้อมูล", "คำเตือน", MessageBoxButtons.OK,
                                                    MessageBoxIcon.Warning);
  }
  else
  {
    Thread progressThread = new Thread(delegate()
    {
      Form objForm = new frmLoadingProcess();
        objForm.ShowDialog();
```

```
    });
    findDHPAlgorithm();
    showResultDHPAlgorithm();
  }
}
```

2. findDHPAlgorithm() จากโมดูลแรกจะเห็นว่ามีการเรียกโมดูลย่อย ซึ่งจะเป็นส่วนที่ใช้ในการค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิม

```
private void findDHPAlgorithm()
{
  double tempTime = 0;
  dtCTime = new DataTable();
  dtCTime.Columns.Add("ลำดับที่ C");
  dtCTime.Columns.Add("เวลา (วินาที)");
  dtLTime = new DataTable();
  dtLTime.Columns.Add("ลำดับที่ L");
  dtLTime.Columns.Add("เวลา (วินาที)");
  dgvDHPSampleData.DataSource = null;
  dgvTwoItemSet.DataSource = null;
  dgvHashTable.DataSource = null;
  dgvSampleCDHPResult.DataSource = null;
  dgvSampleLDHPResult.DataSource = null;
  dgvDHPCTime.DataSource = null;
  dgvDHPLTime.DataSource = null;
  successStatus = false;
  iiNumber = 0;
  dsCandidateDHP = new DataSet();
  dsLargeDHP = new DataSet();
  subProcessTime.Reset();
  subProcessTime.Start();
  dsCandidateDHP.Tables.Add(getCandidateItemSet(dtSampleDHP));
  subProcessTime.Stop();
  tempTime = subProcessTime.ElapsedMilliseconds;
```

```
tempTime = tempTime / 1000;
DataRow insertTime = dtCTime.NewRow();
insertTime[0] = "C1";
insertTime[1] = tempTime;
dtCTime.Rows.Add(insertTime);
subProcessTime.Reset();
subProcessTime.Start();
dsLargeDHP.Tables.Add(getLargeItemSet(dsCandidateDHP.Tables[0],
                                Convert.ToDouble(nudMinDHP.Value), dtSampleDHP));
subProcessTime.Stop();
tempTime = subProcessTime.ElapsedMilliseconds;
tempTime = tempTime / 1000;
insertTime = dtLTime.NewRow();
insertTime[0] = "L1";
insertTime[1] = tempTime;
dtLTime.Rows.Add(insertTime);
subProcessTime.Reset();
subProcessTime.Start();
dtTwoItemSet = getTwoItemSet(dtSampleDHP).Copy();
subProcessTime.Stop();
 tempTime = subProcessTime.ElapsedMilliseconds;
tempTime = tempTime / 1000;
lbl2ItemSet.Text = "2-Item Set = " + tempTime.ToString() + " วินาที";
subProcessTime.Reset();
subProcessTime.Start();
dtHash = getHashTable(dtTwoItemSet, Convert.ToInt32(nudHashValue.Value));
subProcessTime.Stop();
tempTime = subProcessTime.ElapsedMilliseconds;
tempTime = tempTime / 1000;
lblHashTable.Text = "Hash Table = " + tempTime.ToString() + " วินาที";
while (!successStatus)
{
```

```
if (iiNumber != 0)
{
  if (iiNumber == 1 && dsLargeDHP.Tables[iiNumber - 1].Rows.Count != 0)
  {
    subProcessTime.Reset();
    subProcessTime.Start();
    dsCandidateDHP.Tables.Add(getCandidateItemSetDHP_TwoItemSet
                              (dsLargeDHP.Tables[iiNumber - 1], dtHash, iiNumber - 1,
                               Convert.ToDouble(nudMinDHP.Value), dtSampleDHP));
    subProcessTime.Stop();
    tempTime = subProcessTime.ElapsedMilliseconds;
    tempTime = tempTime / 1000;
    insertTime = dtCTime.NewRow();
    insertTime[0] = "C" + (iiNumber + 1).ToString();
    insertTime[1] = tempTime;
    dtCTime.Rows.Add(insertTime);
    subProcessTime.Reset();
    subProcessTime.Start();
    dsLargeDHP.Tables.Add(getLargeItemSetN
                              (dsCandidateDHP.Tables[iiNumber],
                               Convert.ToDouble(nudMinDHP.Value), dtSampleDHP));
    subProcessTime.Stop();
    tempTime = subProcessTime.ElapsedMilliseconds;
    tempTime = tempTime / 1000;
    insertTime = dtLTime.NewRow();
    insertTime[0] = "L" + (iiNumber + 1).ToString();
    insertTime[1] = tempTime;
    dtLTime.Rows.Add(insertTime);
  }
  else if (dsLargeDHP.Tables[iiNumber - 1].Rows.Count > 1)
  {
    subProcessTime.Reset();
```

```
        subProcessTime.Start();
        dsCandidateDHP.Tables.Add(getCandidateItemSetN
                                (dsLargeDHP.Tables[iiNumber - 1], dtSampleDHP, iiNumber - 1));
        subProcessTime.Stop();
        tempTime = subProcessTime.ElapsedMilliseconds;
        tempTime = tempTime / 1000;
        insertTime = dtCTime.NewRow();
        insertTime[0] = "C" + (iiNumber + 1).ToString();
        insertTime[1] = tempTime;
        dtCTime.Rows.Add(insertTime);
        subProcessTime.Reset();
        subProcessTime.Start();
        dsLargeDHP.Tables.Add(getLargeItemSetN
                                (dsCandidateDHP.Tables[iiNumber],
                                Convert.ToDouble(nudMinDHP.Value), dtSampleDHP));
        subProcessTime.Stop();
        tempTime = subProcessTime.ElapsedMilliseconds;
        tempTime = tempTime / 1000;
        insertTime = dtLTime.NewRow();
        insertTime[0] = "L" + (iiNumber + 1).ToString();
        insertTime[1] = tempTime;
        dtLTime.Rows.Add(insertTime);
    }
    else if (dsLargeDHP.Tables[iiNumber - 1].Rows.Count == 1)
    {
        successStatus = true;
    }
    else
    {
        successStatus = true;
        dsLargeDHP.Tables.RemoveAt(iiNumber - 1);
    }
```

```
    }
    iiNumber++;
  }
}
```

3. getCandidateItemSet() จากโมดูลที่ 2 จะมีการเรียกโมดูลย่อยเพื่อสแกนหา C_1 จากฐานข้อมูลพร้อมนับค่า Support

```
private DataTable getCandidateItemSet(DataTable pSampleTrans)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    foreach (DataRow objDR in pSampleTrans.Rows)
    {
        for (int iCol = 1; iCol <= pSampleTrans.Columns.Count - 1; iCol++)
        {
            if (objDR[iCol].ToString().Trim() != string.Empty)
            {
                if (outResult.Rows.Count == 0)
                {
                    DataRow insertRow = outResult.NewRow();
                    insertRow["item_set"] = objDR[iCol].ToString();
                    outResult.Rows.Add(insertRow);
                }
                else
                {
                    var checkItems = (from q in outResult.AsEnumerable() where
                        q["item_set"].ToString() == objDR[iCol].ToString() select q).Count();
                    if (checkItems == 0)
                    {
                        DataRow insertRow = outResult.NewRow();
                        insertRow["item_set"] = objDR[iCol].ToString();
                        outResult.Rows.Add(insertRow);
```

```
                    }
                }
            }
        }
    }
    foreach (DataRow objDR in outResult.Rows)
    {
        int numberOfItems = 0;
        foreach (DataRow countDR in pSampleTrans.Rows)
        {
            for (int iCol = 1; iCol <= pSampleTrans.Columns.Count - 1; iCol++)
            {
                if (countDR[iCol].ToString().Trim() != string.Empty)
                {
                    if (countDR[iCol].ToString() == objDR["item_set"].ToString())
                    {
                        numberOfItems += 1;
                    }
                }
            }
        }
        DataRow updateRow = objDR;
        updateRow["support"] = numberOfItems;
    }
    DataView sortTable = outResult.DefaultView;
    sortTable.Sort = "item_set asc";
    return sortTable.ToTable();
}
```

4. getLargeItemSet() จากโมดูลที่ 2 หลังจากที่เราได้สแกนหา C_1 พร้อมกับค่า Support ของ 1-Itemset ทั้งหมดแล้วเราจะนำ C_1 มาเปรียบเทียบค่า min_sup เพื่อหา L_1

```
private DataTable getLargeItemSet(DataTable pCandidateItemSet, double pMinSup, DataTable
                                                                                pSampleTrans)
```

```
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    int minSup = calculateMinSup((pMinSup * pSampleTrans.Rows.Count).ToString());
    var largeItems = from q in pCandidateItemSet.AsEnumerable() where
                                Convert.ToInt32(q["support"].ToString()) >= minSup select q;
    foreach (DataRow largeItem in largeItems)
    {
        DataRow insertRow = outResult.NewRow();
        insertRow["item_set"] = largeItem["item_set"];
        insertRow["support"] = largeItem["support"];
        outResult.Rows.Add(insertRow);
    }
    outResult.AcceptChanges();
    return outResult;
}
```

5. getTwoItemSet() หลังจากที่เราได้ค่า L_1 มาแล้วนั้น เราจะสร้างตารางแฮชโดยเริ่มต้นสร้างเซทย่อยของ 2-Itemsets ที่เป็นไปได้ทั้งหมดก่อน

```
private DataTable getTwoItemSet(DataTable pSampleTrans)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    string markJoin = string.Empty;
    foreach (DataRow loopDR in pSampleTrans.Rows)
    {
        DataRow insertRow = outResult.NewRow();
        for (int iMark = 1; iMark <= loopDR.ItemArray.Length - 1; iMark++)
        {
            for (int iCol = iMark; iCol <= loopDR.ItemArray.Length - 1; iCol++)
            {
                if (loopDR[iCol].ToString().Trim() != string.Empty)
```

```
                {
                    if (iCol == iMark)
                    {
                        markJoin = loopDR[iCol].ToString();
                    }
                    else
                    {
                        insertRow["item_set"] = insertRow.IsNull("item_set") ? markJoin + "/" +
                                loopDR[iCol].ToString() : insertRow["item_set"].ToString() + "," +
                                markJoin + "/" + loopDR[iCol].ToString();
                    }
                }
            }
        }
        outResult.Rows.Add(insertRow);
    }
    return outResult;
}
```

6. getHashTable() เราจะนำเซทย่อยมาคำนวณหาค่า Address เพื่อสร้างตารางแฮช

```
private DataTable getHashTable(DataTable pTwoItemSet, int pHashNumber)
{
    DataTable outResult = new DataTable();
    for (int addCol = 0; addCol < pHashNumber; addCol++)
    {
        outResult.Columns.Add(addCol.ToString());
    }
    foreach (DataRow inputToHash in pTwoItemSet.Rows)
    {
        foreach (string calData in inputToHash[0].ToString().Split(",".ToCharArray()))
        {
            string[] iData = calData.Split("/".ToCharArray());
            int hashAddress = calculateHashAddress(Convert.ToInt32(iData[0]),
```

```
                                        Convert.ToInt32(iData[1]), pHashNumber);
            if (outResult.Rows.Count == 0)
            {
               outResult.Rows.Add();
            }
            DataRow updateRow = outResult.Rows[0];
            updateRow[hashAddress] = updateRow.IsNull(hashAddress) ? calData :
                                    updateRow[hashAddress].ToString() + "," + calData;
        }
    }
    return outResult;
}
```

7. จากโมดูลที่ 2 เมื่อทำงานมาถึงตรงนี้เราจะมีการตรวจสอบค่า k ว่ามีค่าเท่ากับ 2 หรือไม่ ถ้าเท่ากับ 2 จะไปทำงานที่โมดูลในข้อ 8 แต่ถ้าไม่เท่ากับ 2 จะไปทำงานที่โมดูลในข้อ 9

8. getCandidateItemSetDHP_TwoItemSet() สร้าง C_k ด้วยการจอย L_k มาเปรียบเทียบกับค่าในตารางแฮชเพื่อลดจำนวนของ C_k แล้วจะไปทำงานต่อที่ข้อ 10

```
private DataTable getCandidateItemSetDHP_TwoItemSet(DataTable pLItemSet, DataTable
                                pHashTable, int pCol, double pMinSup, DataTable pSampleTrans)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    int jNumber = 1;
    for (int iNumber = 0; iNumber <= pLItemSet.Rows.Count - 2; iNumber++)
    {
        for (int iRow = jNumber; iRow <= pLItemSet.Rows.Count - 1; iRow++)
        {
            DataRow insertRow = outResult.NewRow();
            insertRow["item_set"] = pLItemSet.Rows[iNumber][0].ToString() + "," +
                                                    pLItemSet.Rows[iRow][0].ToString();
            outResult.Rows.Add(insertRow);
        }
```

```
        jNumber++;
    }
    foreach (DataRow valueDR in outResult.Rows)
    {
        string compareHash = string.Empty;
        foreach (string iCompare in valueDR["item_set"].ToString().Split(",".ToCharArray()))
        {
            compareHash += compareHash == string.Empty ? iCompare : "/" + iCompare;
        }
        int hashAddress = findHashAddress(compareHash, pHashTable);
        valueDR["support"] = hashAddress == -1 ? 0 :
                        pHashTable.Rows[0][hashAddress].ToString().Split(",".ToCharArray()).
                                                        GetUpperBound(0) + 1;
}
int minSup = calculateMinSup((pMinSup * pSampleTrans.Rows.Count).ToString());
var largeItems = from q in outResult.AsEnumerable() where
                        Convert.ToInt32(q["support"].ToString()) >= minSup select q;
DataTable tempResult = new DataTable();
tempResult = largeItems.CopyToDataTable();
foreach (DataRow valueDR in tempResult.Rows)
{
    int supportValue = 0;
    string[] tempValue = valueDR["item_set"].ToString().Split(",".ToCharArray());
    foreach (DataRow sampleDR in pSampleTrans.Rows)
    {
        string[] tempSample = new string[sampleDR.ItemArray.Length - 1];
        for (int iCol = 0; iCol < sampleDR.ItemArray.Length - 1; iCol++)
        {
            tempSample[iCol] = sampleDR[iCol + 1].ToString();
        }
        var intersectResult = tempValue.Intersect(tempSample).ToArray<string>();
        bool checkEqual = false;
```

```
            if (tempValue.GetUpperBound(0) == intersectResult.GetUpperBound(0))
            {
               checkEqual = compareArray(tempValue, intersectResult);
            }
            if (checkEqual)
            {
               supportValue++;
            }
         }
         valueDR["support"] = supportValue;
      }
      return tempResult;
}
```

9. getCandidateItemSetN() สร้าง C_k โดยการเอา L_k มาจอยกับ L_k แล้วจะไปทำงานที่ข้อ 10

```
private DataTable getCandidateItemSetN(DataTable pLItemSet, DataTable pSampleTrans, int
                                                                                    pCol)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    if (pCol == 0)
    {
        int jNumber = 1;
        for (int iNumber = 0; iNumber <= pLItemSet.Rows.Count - 2; iNumber++)
        {
            for (int iRow = jNumber; iRow <= pLItemSet.Rows.Count - 1; iRow++)
            {
               DataRow insertRow = outResult.NewRow();
               insertRow["item_set"] = pLItemSet.Rows[iNumber][0].ToString() + "," +
                                                    pLItemSet.Rows[iRow][0].ToString();
               outResult.Rows.Add(insertRow);
            }
```

```
            jNumber++;
        }
    }
    else
    {
        int currentRecord = 0;
        string[] markJoin = null;
        string[] currentMark = null;
        foreach (DataRow dataDR in pLItemSet.Rows)
        {
            int runRecord = 0;
            while (runRecord < pLItemSet.Rows.Count - currentRecord)
            {
                if (runRecord == 0)
                {
                    markJoin = dataDR["item_set"].ToString().Split(",".ToCharArray());
                }
                else
                {
                    currentMark = pLItemSet.Rows[currentRecord +
                                                 runRecord][0].ToString().Split(",".ToCharArray());
                    bool joinStatus = false;
                    for (int iMark = 0; iMark <= pCol - 1; iMark++)
                    {
                        joinStatus = markJoin[iMark] == currentMark[iMark] ? true : false;
                        if (!joinStatus)
                        {
                            break;
                        }
                    }
                    if (joinStatus)
                    {
```

```
                var joinResult = markJoin.Union(currentMark).ToArray<string>();
                DataRow insertRow = outResult.NewRow();
                foreach (string joinDR in joinResult)
                {
                    insertRow["item_set"] = insertRow.IsNull("item_set") ? joinDR :
                                                    insertRow["item_set"] += "," + joinDR;
                }
                outResult.Rows.Add(insertRow);
            }
            else
            {
                break;
            }
        }
        runRecord++;
    }
    currentRecord++;
  }
}
foreach (DataRow valueDR in outResult.Rows)
{
  int supportValue = 0;
  string[] tempValue = valueDR["item_set"].ToString().Split(",".ToCharArray());
  foreach (DataRow sampleDR in pSampleTrans.Rows)
  {
    string[] tempSample = new string[sampleDR.ItemArray.Length - 1];
    for (int iCol = 0; iCol < sampleDR.ItemArray.Length - 1; iCol++)
    {
      tempSample[iCol] = sampleDR[iCol + 1].ToString();
    }
    var intersectResult = tempValue.Intersect(tempSample).ToArray<string>();
    bool checkEqual = false;
```

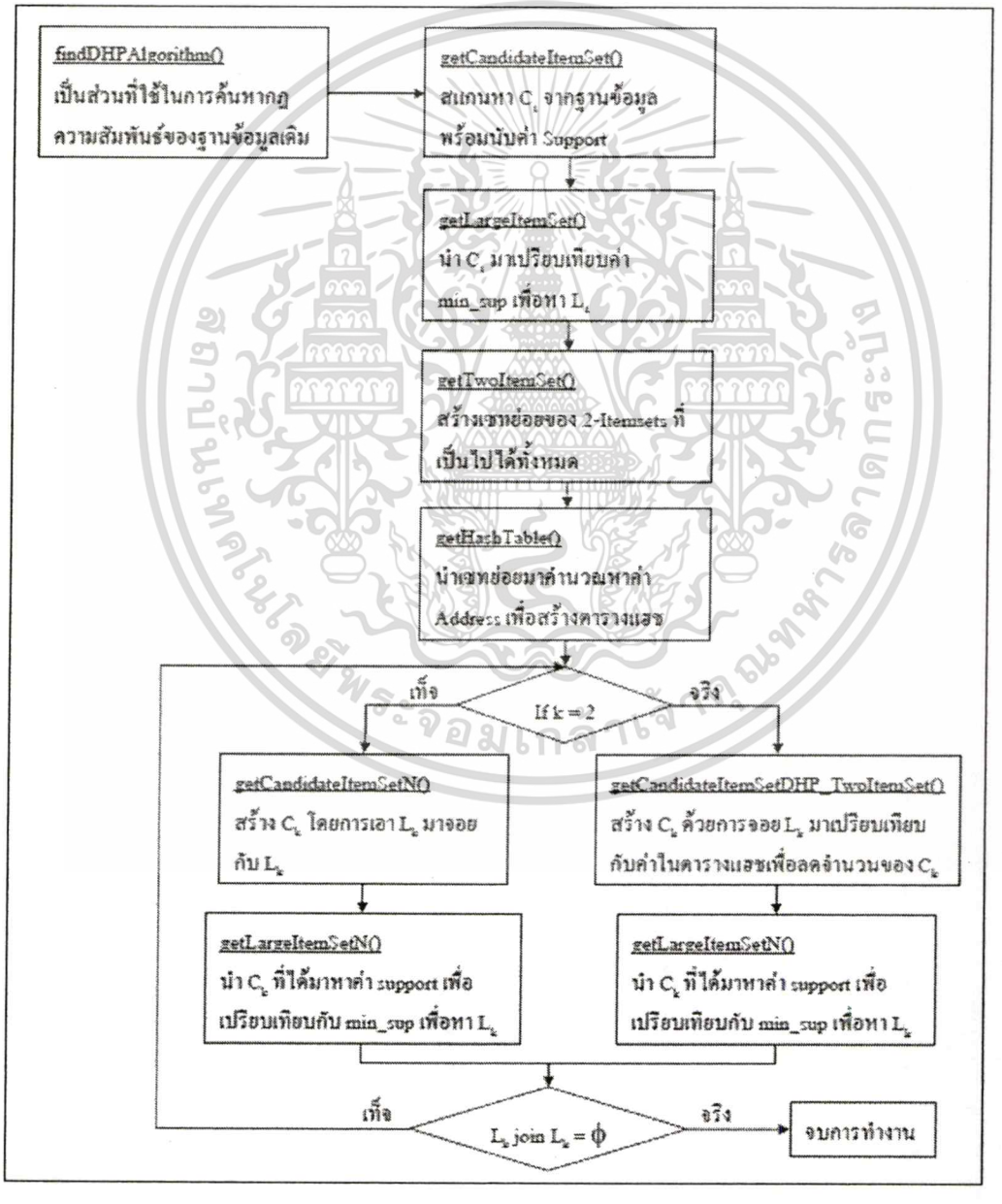
```
            if (tempValue.GetUpperBound(0) == intersectResult.GetUpperBound(0))
            {
               checkEqual = compareArray(tempValue, intersectResult);
            }
            if (checkEqual)
            {
               supportValue++;
            }
         }
         valueDR["support"] = supportValue;
      }
      outResult.AcceptChanges();
      return outResult;
}
```

10. getLargeItemSetN() นำ C_k ที่ได้มาหาค่า support เพื่อเปรียบเทียบกับ min_sup เพื่อหา L_k หลังจากที่ทำงานที่โมดูลนี้เสร็จจะไปตรวจสอบเงื่อนไขเพื่อทำงานต่อที่ข้อ 11

```
private DataTable getLargeItemSetN(DataTable pCItemSet, double pMinSup, DataTable
                                                                    pSampleTrans)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    int minSup = calculateMinSup((pMinSup * pSampleTrans.Rows.Count).ToString());
    var largeItems = from q in pCItemSet.AsEnumerable() where
                                Convert.ToInt32(q["support"].ToString()) >= minSup select q;
    foreach (DataRow largeItem in largeItems)
    {
        DataRow insertRow = outResult.NewRow();
        insertRow["item_set"] = largeItem["item_set"];
        insertRow["support"] = largeItem["support"];
        outResult.Rows.Add(insertRow);
    }
```

```
    outResult.AcceptChanges();
    return outResult;
}
```

11. จะทำการตรวจสอบโดยนำค่า L_k ที่หามาได้มาจอยกันเพื่อดูว่าหลังจากจอยกันแล้วเซตของ C_{k+1} มีค่าเท่ากับเซตว่างจริงหรือไม่ ถ้าจริงจะถือเราไม่สามารถหาค่า Large Itemsets ได้อีกแล้ว จะถือว่าการค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิมเสร็จสิ้นแล้ว เราสามารถนำค่า Large Itemsets ที่หามาได้ทั้งหมดนี้ไปสร้างเป็นกฎความสัมพันธ์ได้ แต่ถ้าเงื่อนไขข้างต้นเป็นเท็จเราจะวนกลับไปทำงานที่ข้อ 7



รูปที่ ก.2.2 โมดูลการทำงานของอัลกอริทึม HFUP กับฐานข้อมูลเดิม

ก.2.2 แสดง Code ของแต่ละโมดูลย่อยใน FUP Process

1. btnFUPDHPProcess_Click() หลังจากที่เราเลือกไฟล์ที่ต้องการหากฎความสัมพันธ์ของข้อมูลที่เพิ่มเข้ามาใหม่ เราจะเริ่มต้นกระบวนการในหาค้นหากฎความสัมพันธ์ของข้อมูลทั้งหมด โดยคลิกที่ปุ่ม FUP Process เพื่อเริ่มการประมวล

```
private void btnFUPDHPProcess_Click(object sender, EventArgs e)
{
   if (dsCandidateDHP.Tables.Count == 0)
   {
      MessageBox.Show("กรุณาทำ Process ข้อมูล Sample ก่อน", "คำเตือน",
                                        MessageBoxButtons.OK, MessageBoxIcon.Warning);
   }
   else if (txtTransDHPFileDIR.Text == string.Empty)
   {
       MessageBox.Show("กรุณาเลือกไฟล์ Transaction", "คำเตือน", MessageBoxButtons.OK,
                                                        MessageBoxIcon.Warning);
   }
   else
  {
       Thread progressThread = new Thread(delegate()
       {
           Form objForm = new frmLoadingProcess();
           objForm.ShowDialog();
       });
       progressThread.Start();
       processTime.Reset();
       processTime.Start();
       findFUPDHPAlgorithm();
       processTime.Stop();
       showResultFUPDHPAlgorithm();
       dsCandidateTrans.Reset();
       progressThread.Abort();
       processTwoTime = processTime.ElapsedMilliseconds;
```

```
        processTwoTime = processTwoTime / 1000;
        lblDHPFUPProcessTime.Text = "= " + processTwoTime.ToString() + " วินาที";
        allPrcessTime = processOneTime + processTwoTime;
        lblDHPProcessTime.Text = allPrcessTime.ToString();
    }
}
```

2. findFUPDHPAlgorithm() จากโมดูลแรกจะเห็นว่ามีการเรียกโมดูลย่อย ซึ่งจะเป็นส่วนที่ใช้ในการค้นหากฎความสัมพันธ์เมื่อมีข้อมูลเพิ่มเข้ามาใหม่

```
private void findFUPDHPAlgorithm()
{
    dgvTransCDHP.DataSource = null;
    dgvTransLDHP.DataSource = null;
    dgvTransDHPRecord.DataSource = null;
    dgvDHPFUPCTime.DataSource = null;
    dgvDHPFUPLTime.DataSource = null;
    double tempTime = 0, calTime = 0;
    successStatus = false;
    iiNumber = 0;
    int minAll = 0;
    int minTrans = 0;
    dtCTime = new DataTable();
    dtCTime.Columns.Add("ลำดับที่ C");
    dtCTime.Columns.Add("เวลา (วินาที)");
    dtLTime = new DataTable();
    dtLTime.Columns.Add("ลำดับที่ L");
    dtLTime.Columns.Add("เวลา (วินาที)");
    dsCandidateTransDHP = new DataSet();
    dsLargeTransDHP = new DataSet();
    dtTempC = new DataTable();
    subProcessTime.Reset();
    subProcessTime.Start();
    dtTempC = getCandidateItemSet(dtTransDHP).Copy();
```

```
subProcessTime.Stop();
calTime = subProcessTime.ElapsedMilliseconds;
calTime = calTime / 1000;
minAll = calculateMinSup((Convert.ToDouble(nudMinDHP.Value) *
                    (dtSampleDHP.Rows.Count + dtTransDHP.Rows.Count)).ToString());
lblMinAllDHP.Text = "min_รวม = " + minAll.ToString();
minTrans = calculateMinSup((Convert.ToDouble(nudMinDHP.Value) *
                                                    dtTransDHP.Rows.Count).ToString());
lblMinTransDHP.Text = "min_trans = " + minTrans.ToString();
while (!successStatus)
{
    if (dsLargeDHP.Tables.Count == 0)
    {
        successStatus = true;
    }
    else if (iiNumber == 0)
    {
        subProcessTime.Reset();
        subProcessTime.Start();
        dsLargeTransDHP.Tables.Add(getLargeItemSetTrans(dtTempC,
                                    dsLargeDHP.Tables[iiNumber], minAll, minTrans));
        subProcessTime.Stop();
        tempTime = subProcessTime.ElapsedMilliseconds;
        tempTime = tempTime / 1000;
        tempTime = tempTime + calTime;
        DataRow insertTime = dtCTime.NewRow();
        insertTime[0] = "C1";
        insertTime[1] = tempTime;
        dtCTime.Rows.Add(insertTime);
        insertTime = dtLTime.NewRow();
        insertTime[0] = "L1";
        insertTime[1] = tempTime;
```

```
        dtLTime.Rows.Add(insertTime);
        dsCandidateTransDHP.Tables.Add(dtNewC);
    }
    else if (dsLargeTransDHP.Tables[iiNumber - 1].Rows.Count > 1 && iiNumber <
                                                        dsLargeDHP.Tables.Count)
    {
        subProcessTime.Reset();
        subProcessTime.Start();
        dtTempC = getCandidateItemSetN(dsLargeTransDHP.Tables[iiNumber - 1],
                                                        dtTransDHP, iiNumber - 1);
        dsLargeTransDHP.Tables.Add(getLargeItemSetTrans(dtTempC,
                                dsLargeDHP.Tables[iiNumber], minAll, minTrans));
        subProcessTime.Stop();
        tempTime = subProcessTime.ElapsedMilliseconds;
        tempTime = tempTime / 1000;
        DataRow insertTime = dtCTime.NewRow();
        insertTime[0] = "C" + (iiNumber + 1).ToString();
        insertTime[1] = tempTime;
        dtCTime.Rows.Add(insertTime);
        insertTime = dtLTime.NewRow();
        insertTime[0] = "L" + (iiNumber + 1).ToString();
        insertTime[1] = tempTime;
        dtLTime.Rows.Add(insertTime);
        dsCandidateTransDHP.Tables.Add(dtNewC);
    }
    else if (dsLargeTransDHP.Tables[iiNumber - 1].Rows.Count == 1)
    {
        successStatus = true;
    }
    else
    {
        successStatus = true;
```

```
        }
        iiNumber++;
      }
    }
```

3. getCandidateItemSet() สแกนหา C_1 จากฐานข้อมูลที่เพิ่มเข้ามาใหม่พร้อมนับค่า Support

```
private DataTable getCandidateItemSet(DataTable pSampleTrans)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    foreach (DataRow objDR in pSampleTrans.Rows)
    {
        for (int iCol = 1; iCol <= pSampleTrans.Columns.Count - 1; iCol++)
        {
            if (objDR[iCol].ToString().Trim() != string.Empty)
            {
                if (outResult.Rows.Count == 0)
                {
                    DataRow insertRow = outResult.NewRow();
                    insertRow["item_set"] = objDR[iCol].ToString();
                    outResult.Rows.Add(insertRow);
                }
                else
                {
                    var checkItems = (from q in outResult.AsEnumerable() where
                        q["item_set"].ToString() == objDR[iCol].ToString() select q).Count();
                    if (checkItems == 0)
                    {
                        DataRow insertRow = outResult.NewRow();
                        insertRow["item_set"] = objDR[iCol].ToString();
                        outResult.Rows.Add(insertRow);
```

```
                }
            }
        }
    }
    foreach (DataRow objDR in outResult.Rows)
    {
        int numberOfItems = 0;
        foreach (DataRow countDR in pSampleTrans.Rows)
        {
            for (int iCol = 1; iCol <= pSampleTrans.Columns.Count - 1; iCol++)
            {
                if (countDR[iCol].ToString().Trim() != string.Empty)
                {
                    if (countDR[iCol].ToString() == objDR["item_set"].ToString())
                    {
                        numberOfItems += 1;
                    }
                }
            }
        }
        DataRow updateRow = objDR;
        updateRow["support"] = numberOfItems;
    }
    DataView sortTable = outResult.DefaultView;
    sortTable.Sort = "item_set asc";
    return sortTable.ToTable();
}
```

4. getLargeItemSetTrans() หา New Large Itemsets ตามวิธีการของอัลกอริทึม FUP ทั้ง 3 ขั้นตอนโดยจะนำค่า Large Itemset จากการค้นหากฎความสัมพันธ์เดิมมาพิจารณาร่วมกับข้อมูลใหม่ที่เพิ่มเข้ามาด้วย

```
private DataTable getLargeItemSetTrans(DataTable pTempCandidate, DataTable pLargeSample,
                                                                    int pMinAll, int pMinTrans)
```

```
{
    DataTable outResult = new DataTable();
    DataTable newC = new DataTable();
    DataTable tempResult = new DataTable();
    tempResult = pLargeSample.Copy();
    foreach (DataRow intersectDR in tempResult.Rows)
    {
        intersectDR["support"] = Convert.ToInt32(intersectDR["support"].ToString()) +
                                 findItemValue(pTempCandidate, intersectDR["item_set"].
                                 ToString(), "item_set", "support");
    }
    newC = tempResult.Copy();
    int minAll = pMinAll;
    foreach (DataRow deleteDR in tempResult.Rows)
    {
        if (Convert.ToInt32(deleteDR["support"].ToString()) < minAll)
        {
            deleteDR.Delete();
        }
    }
    outResult = tempResult.Copy();
    var dataOld = newC.AsEnumerable();
    var dataNew = pTempCandidate.AsEnumerable();
    var exceptDatas = from q1 in dataNew
    join q2 in dataOld on q1["item_set"] equals q2["item_set"] into j
    from r in j.DefaultIfEmpty()
    where r == null
    select q1;
    int minTrans = pMinTrans;
    foreach (DataRow checkExceptData in exceptDatas)
    {
        if (Convert.ToInt32(checkExceptData["support"].ToString()) >= minTrans)
```

```
            {
                DataRow insertC = newC.NewRow();
                insertC["item_set"] = checkExceptData["item_set"];
                insertC["support"] = checkExceptData["support"];
                newC.Rows.Add(insertC);
                if (Convert.ToInt32(checkExceptData["support"].ToString()) >= minAll)
                {
                    DataRow insertL = outResult.NewRow();
                    insertL["item_set"] = checkExceptData["item_set"];
                    insertL["support"] = checkExceptData["support"];
                    outResult.Rows.Add(insertL);
                }
            }
        }
        DataView sortTable = newC.DefaultView;
        sortTable.Sort = "item_set asc";
        dtNewC = new DataTable();
        dtNewC = sortTable.ToTable();
        sortTable = outResult.DefaultView;
        sortTable.Sort = "item_set asc";
        return sortTable.ToTable();
    }
```

5. getCandidateItemSetN() สร้าง C_k โดยการเอา L_k มาจอยกับ L_k ทำตามขั้นตอนของอัลกอริทึม FUP เพื่อหา C_k ก่อนแล้วนำ C_k ที่ได้ไปหา New Large Itemsets ต่อไป

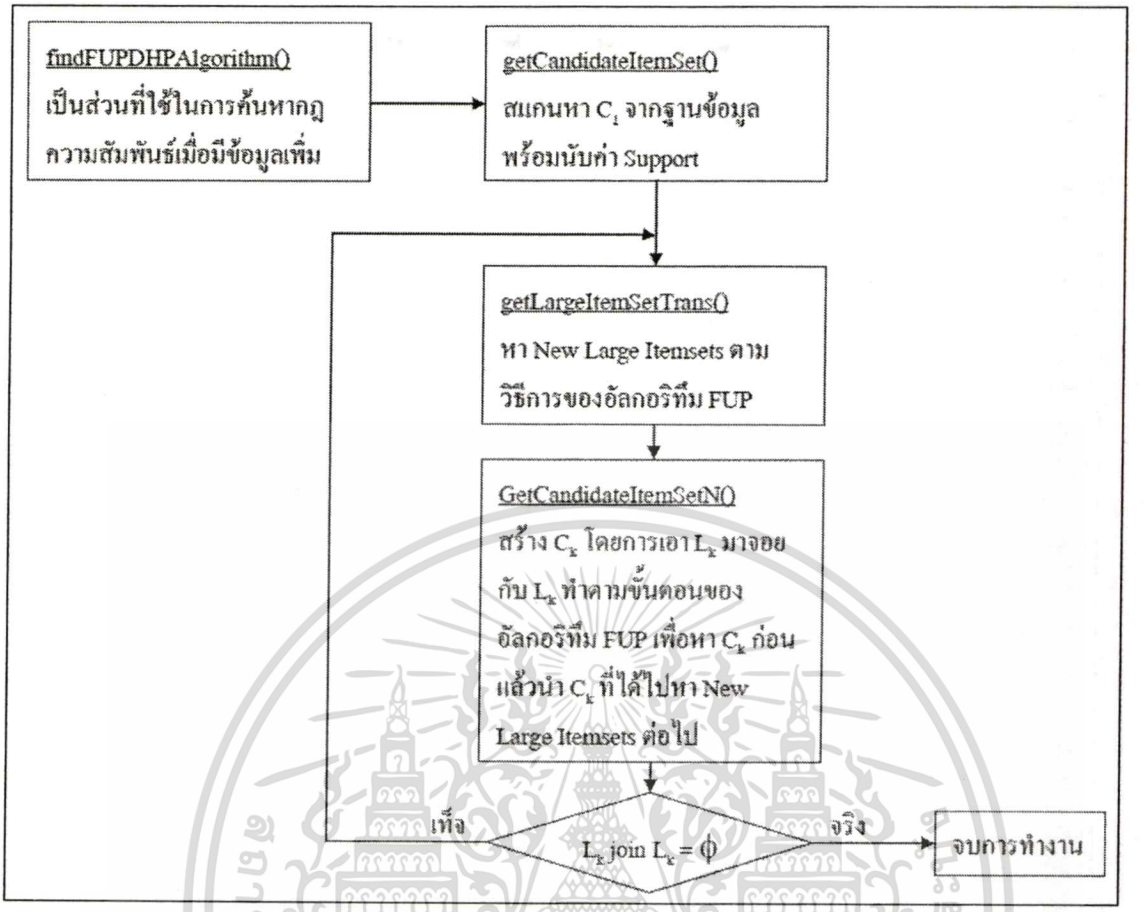
```
private DataTable getCandidateItemSetN(DataTable pLItemSet, DataTable pSampleTrans, int
                                                                                    pCol)
{
    DataTable outResult = new DataTable();
    outResult.Columns.Add("item_set");
    outResult.Columns.Add("support");
    if (pCol == 0)
    {
```

```
        int jNumber = 1;
        for (int iNumber = 0; iNumber <= pLItemSet.Rows.Count - 2; iNumber++)
        {
            for (int iRow = jNumber; iRow <= pLItemSet.Rows.Count - 1; iRow++)
            {
                DataRow insertRow = outResult.NewRow();
                insertRow["item_set"] = pLItemSet.Rows[iNumber][0].ToString() + "," +
                                                    pLItemSet.Rows[iRow][0].ToString();
                outResult.Rows.Add(insertRow);
            }
            jNumber++;
        }
    }
    else
    {
        int currentRecord = 0;
        string[] markJoin = null;
        string[] currentMark = null;
        foreach (DataRow dataDR in pLItemSet.Rows)
        {
            int runRecord = 0;
            while (runRecord < pLItemSet.Rows.Count - currentRecord)
            {
                if (runRecord == 0)
                {
                    markJoin = dataDR["item_set"].ToString().Split(",".ToCharArray());
                }
                else
                {
                    currentMark = pLItemSet.Rows[currentRecord +
                                                runRecord][0].ToString().Split(",".ToCharArray());
                    bool joinStatus = false;
```

```
            for (int iMark = 0; iMark <= pCol - 1; iMark++)
            {
               joinStatus = markJoin[iMark] == currentMark[iMark] ? true : false;
               if (!joinStatus)
               {
                  break;
               }
            }
            if (joinStatus)
            {
               var joinResult = markJoin.Union(currentMark).ToArray<string>();
               DataRow insertRow = outResult.NewRow();
               foreach (string joinDR in joinResult)
               {
                  insertRow["item_set"] = insertRow.IsNull("item_set") ? joinDR :
                                                   insertRow["item_set"] += "," + joinDR;
               }
               outResult.Rows.Add(insertRow);
            }
            else
            {
               break;
            }
         }
         runRecord++;
      }
      currentRecord++;
   }
}
foreach (DataRow valueDR in outResult.Rows)
{
   int supportValue = 0;
```

```
            string[] tempValue = valueDR["item_set"].ToString().Split(",".ToCharArray());
            foreach (DataRow sampleDR in pSampleTrans.Rows)
            {
               string[] tempSample = new string[sampleDR.ItemArray.Length - 1];
               for (int iCol = 0; iCol < sampleDR.ItemArray.Length - 1; iCol++)
               {
                  tempSample[iCol] = sampleDR[iCol + 1].ToString();
               }
               var intersectResult = tempValue.Intersect(tempSample).ToArray<string>();
               bool checkEqual = false;
               if (tempValue.GetUpperBound(0) == intersectResult.GetUpperBound(0))
               {
                  checkEqual = compareArray(tempValue, intersectResult);
               }
               if (checkEqual)
               {
                  supportValue++;
               }
            }
            valueDR["support"] = supportValue;
         }
         outResult.AcceptChanges();
         return outResult;
      }
```

6. จะทำการตรวจสอบโดยนำค่า L_k ที่หามาได้มาจอยกันเพื่อดูว่าหลังจากจอยกันแล้วเซตของ C_{k+1} มีค่าเท่ากับเซตว่างจริงหรือไม่ ถ้าจริงจะถือเราไม่สามารถหาค่า New Large Itemsets ได้อีกแล้ว จะถือว่าการค้นหากฎความสัมพันธ์ของฐานข้อมูลเดิมเสร็จสิ้นแล้ว เราสามารถนำค่า New Large Itemsets ที่หามาได้ทั้งหมดนี้ไปสร้างเป็นกฎความสัมพันธ์ได้ แต่ถ้าเงื่อนไขข้างต้นเป็นเท็จเราจะวนกลับไปทำงานที่ข้อ 4



รูปที่ ก.2.3 โมดูลการทำงานของอัลกอริทึม HFUP กับข้อมูลที่เพิ่มเข้ามา

ประวัติผู้เขียน

ชื่อ-นามสกุล	นางสาวแจ่มจันทร์ คงปาน
วัน เดือน ปีเกิด	6 พฤษภาคม 2523 ที่พิษณุโลก
ที่อยู่	96/5 หมู่ 5 ต.อรัญญิก อ.เมือง จ.พิษณุโลก 65000 โทร.081-534-8150
	Email: nooi_6@hotmail.com
ประวัติการศึกษา	2545 คณะวิทยาศาสตร์ สาขาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยเชียงใหม่