รายงานการวิจัยฉบับสมบูรณ์

การค้นกฎความสัมพันธ์แบบเพิ่มขยาย
Incremental Association Rule Discovery

รองศาสตราจารย์ ดร.วรพจน์ กรีสุระเดช

ได้รับทุนสนับสนุนงานวิจัยจากเงินงบประมาณรายได้ ประจำปีงบประมาณ 2554
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อโครงการ การค้นหากฎความสัมพันธ์แบบเพิ่มขยาย
แหล่งเงิน แหล่งเงินรายได้
ประจำปีงบประมาณ 2554 **จำนวนเงินที่ได้รับการสนับสนุน** 50,000 บาท
ระยะเวลาทำการวิจัย 1 ปี 6 เดือน **ตั้งแต่** ตุลาคม 2554 **ถึง** มีนาคม 2556
ชื่อ-สกุล หัวหน้าโครงการ รองศาสตราจารย์ ดร.วรพจน์ กรีสุระเดช
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

งานวิจัยฉบับนี้ นำเสนออัลกอริทึมสำหรับการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย โดยเป็นการปรับปรุงประสิทธิภาพอัลกอริทึม FUP (Fast UPdate Algorithm) ซึ่งมีวัตถุประสงค์เพื่อใช้ในการค้นหากฎความสัมพันธ์เมื่อฐานข้อมูลมีการเปลี่ยนแปลง เมื่อมีการเพิ่มชุดข้อมูลใหม่จำนวนหนึ่งเขามาในฐานข้อมูลเดิมจะมีผลกระทบต่อกฎความสัมพันธ์ที่เคยได้ทำการค้นหาไว้แล้วก่อนหน้า นั่นคือฟรีเคว้นท์ไอเท็มเซตที่เคยถูกนำไปสร้างกฎความสัมพันธ์ในช่วงระยะเวลาก่อนหน้าอาจจะไม่เป็นไอเท็มเซ็ตตัวที่น่าสนใจสำหรับเวลาปัจจุบัน เพื่อแก้ไขปัญหาการเพิ่มขยายการค้นหากฎความสัมพันธ์เมื่อมีการเพิ่มข้อมูลชุดใหม่เข้ามา งานวิจัยฉบับนี้จึงนำแนวคิดและหลักการทำงานของอัลกอริทึม FUP มาปรับปรุงให้สามารถประมวลผลได้โดยใช้ระยะเวลาน้อยลงแต่ยังได้ความครบถ้วนและถูกต้องของการค้นหาฟรีเคว้นท์ไอเท็มเซ็ตได้เหมือนเดิม ผลการทดลองพบว่าอัลกอริทึม BFUP ซึ่งเป็นอัลกอริทึมที่ปรับปรุงประสิทธิภาพของอัลกริทึม FUP สามารถทำงานได้อย่างมีประสิทธิภาพโดยให้ความถูกต้องและใช้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม FUP เนื่องจาก BFUP มีการสแกนแคนดิเดตไอเท็มเซ็ตในฐานข้อมูลเดิมเพียง 1 รอบ ในขณะที่ FUP จำเป็นต้องสแกนแคนดิเดตไอเท็มเซ็ตในฐานข้อมูลเดิมจำนวนหลายรอบ

คำสำคัญ : การค้นหากฎความสัมพันธ์แบบเพิ่มขยาย การค้นหากฎความสัมพันธ์ ดาต้าไมนิ่ง

Research Title: Incremental Association Rule Discovery
Researcher: Associate Professor Dr.Worapoj Kreesuradej
Faculty: Faculty of Information Technology

ABSTRACT

The most important target of mining incremental association rule is rules maintenance. Since new transactions are inserted into original database, the existing rules may be change. How to maintain association rules with high performance and good efficiency is the challenge for many researchers in this field. A lot of previous purposed algorithms, researchers introduced techniques to reduce a number of times to scan original database such as using infrequent itemsets, collects from original database, to limit that scanning times. This paper presents a simple algorithm of an incremental association mining rule. The algorithm needs only one scanning original database and infrequent itemsets from a previous mining are not required. Furthermore, this purposed algorithm guarantee that all frequent itemsets in an updated database are absolutely detected.

Keywords: Incremental Association Rule Discovery, Association Rule Mining, Data Mining

กิตติกรรมประกาศ

งานวิจัยฉบับนี้สำเร็จได้ด้วยดี โดยการวิจัยครั้งนี้ได้รับทุนสนับสนุนการวิจัยจากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง จากแหล่งทุนประเภทรายได้ ประจำปีงบประมาณ พ.ศ. 2554 ผู้วิจัยต้องขอขอบพระคุณผู้ให้การสนับสนุนทุนวิจัยมา ณ โอกาสนี้

รองศาสตราจารย์ ดร.วรพจน์ กรีสุระเดช

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ
(Introduction)

1.1 ที่มาและความสำคัญของปัญหา

ข้อมูลเป็นองค์ประกอบหนึ่งของระบบเทคโนโลยีสารสนเทศและเป็นทรัพยากรที่มีความสำคัญต่อองค์กรทุกองค์กร โดยเฉพาะอย่างยิ่งในปัจจุบันซึ่งเป็นยุคที่เทคโนโลยีสารสนเทศมีความเจริญเติบโตก้าวหน้าสูง ไม่ว่าจะเป็นความก้าวหน้าของอุปกรณ์เก็บข้อมูลที่สามารถจัดเก็บข้อมูลปริมาณมากได้ด้วยต้นทุนที่ไม่แพงมากนัก อีกทั้งเทคโนโลยีและเครื่องมือต่างๆ ที่ช่วยในการวิเคราะห์ประมวลผลข้อมูลจำนวนมหาศาลนั้นได้ภายในระยะเวลาอันสั้น หากองค์กรใดสามารถนำข้อมูลมาใช้ได้อย่างรวดเร็วและเกิดประโยชน์สูงสุดย่อมสร้างความได้เปรียบในการแข่งขันให้แก่องค์กรนั้นๆ อย่างไรก็ดี แม้ว่าในปัจจุบันเทคโนโลยีการจัดเก็บข้อมูลจะช่วยให้องค์กรสามารถจัดเก็บข้อมูลจำนวนมากได้อย่างมีประสิทธิภาพ แต่อัตราการใช้ประโยชน์จากข้อมูลนั้นยังคงมีน้อยมากเมื่อเทียบกับจำนวนข้อมูล ซึ่งองค์ความรู้ (Knowledge) อีกมากที่ยังไม่ได้นำมาใช้ยังคงถูกซ่อนอยู่ในกลุ่มข้อมูลดังกล่าว

เทคนิคการทำเหมืองข้อมูล (Data Mining) ซึ่งบางครั้งจะถูกกล่าวถึงว่า การค้นหาองค์ความรู้ในฐานข้อมูล (Knowledge Discovery in Database: KDD) เป็นกระบวนการที่ถูกนำมาใช้เพื่อค้นหารูปแบบ (pattern) หรือความสัมพันธ์ที่ไม่เคยค้นพบมาก่อนที่ถูกซ่อนอยู่ในข้อมูลจำนวนมหาศาลโดยอัตโนมัติ รูปแบบข้อมูลที่ค้นพบดังกล่าวจะช่วยให้องค์กรสามารถสร้างความได้เปรียบในการแข่งขันได้ อาทิ การนำเทคนิคการทำเหมืองข้อมูลเข้ามาช่วยในการวิเคราะห์กลุ่มลูกค้าเงินกู้ชั้นดีของธนาคาร การจำแนกกลุ่มลูกค้าที่มีความสามารถในการซื้อสินค้าราคาสูงจำพวกบ้านหรือรถยนต์ และการวิเคราะห์รูปแบบการซื้อสินค้าของลูกค้าเป็นต้น

การค้นหากฎความสัมพันธ์ (Association Rules Mining) จัดเป็นเทคนิคที่สำคัญเทคนิคหนึ่งของการทำเหมืองข้อมูล โดยการค้นหากฎความสัมพันธ์จะเป็นกระบวนการค้นหารูปแบบความสัมพันธ์ของข้อมูลที่เกี่ยวข้องกันระหว่างรายการต่างๆ ที่ถูกจัดเก็บไว้ในฐานข้อมูล ทั้งนี้ความสัมพันธ์ที่ได้จะแสดงออกมาในรูปแบบของกฎความสัมพันธ์ A $\rightarrow$ B ซึ่งหมายถึง เหตุการณ์ B จะเกิดขึ้นหลังจากที่เกิดเหตุการณ์ A แล้ว รูปแบบกฎความสัมพันธ์ดังกล่าวเป็นที่นิยมในองค์กรธุรกิจค้าปลีกที่นำไปวิเคราะห์หารูปแบบพฤติกรรมการเลือกซื้อสินค้าของลูกค้า

การค้นหากฎความสัมพันธ์โดยทั่วไปจะประกอบด้วยขั้นตอน 2 ขั้นหลัก ได้แก่ 1) การค้นหา large itemset ซึ่งจะใช้ค่าสนับสนุนต่ำสุด (minimum support threshold) เป็นค่าที่ใช้ในการตรวจสอบว่า ไอเท็มเซตใดบ้างที่มีโอกาสจะเป็น Large itemset ได้ และ 2) การสร้างกฎความสัมพันธ์จาก Large itemset ที่หาได้จากขั้นตอนที่หนึ่ง โดยจะใช้ค่าความเชื่อมั่นต่ำสุด (minimum confidence threshold) ในการตรวจสอบว่ากฎใดควรจัดอยู่ในกลุ่มของกฎที่น่าสนใจ ทั้งนี้ ค่าสนับสนุนต่ำสุด และค่าความเชื่อมั่นต่ำสุด ผู้ใช้จะเป็นผู้กำหนด

ในระยะแรกของการค้นหากฎความสัมพันธ์จะเป็นการค้นหากฎความสัมพันธ์ภายใต้สมมติฐานว่าฐานข้อมูลไม่มีการเปลี่ยนแปลง (static database) นั่นคือไม่มีการเพิ่ม ลบ หรือแก้ไขข้อมูลภายในฐานข้อมูล หากเมื่อคำนึงถึงความเป็นจริง ธรรมชาติของฐานข้อมูลนั้นมักจะมีการเพิ่ม ลบ และแก้ไขอยู่

ตลอดเวลา (dynamic database) ซึ่งจะมีผลให้ Large itemset ที่เคยค้นพบมีการเปลี่ยนแปลงตามการแก้ไขฐานข้อมูล ดังนั้นจึงจำเป็นต้องมีการประมวลผลข้อมูลทั้งเก่าและใหม่ร่วมกันเพื่อให้กฎความสัมพันธ์มีการปรับปรุงให้มีความทันสมัยอยู่ตลอดเวลา อย่างไรก็ดีการประมวลผลข้อมูลทั้งเก่าและใหม่ร่วมกันทำให้สิ้นเปลืองระยะเวลาในการประมวลผลเป็นอย่างมาก

จากปัญหาดังกล่าวจึงเป็นที่มาของงานวิจัยด้านการเพิ่มขยายการค้นหากฎความสัมพันธ์ (Incremental Mining on Association Rules) ที่พยายามคิดค้นอัลกอริทึมที่ช่วยลดระยะเวลาในการประมวลฐานข้อมูลเดิม ซึ่งงานวิจัยฉบับนี้จะนำเสนออัลกอริทึมการค้นหากฎความสัมพันธ์แบบเพิ่มขยายที่ลดระยะเวลาในการประมวลผลฐานข้อมูลเดิมโดยมุ่งประเด็นไปที่การลดจำนวนรอบของการสแกนฐานข้อมูลเดิมจากจำนวนหลายรอบให้เหลือเพียงรอบเดียว

1.2 วัตถุประสงค์ของการศึกษา

งานวิจัยนี้มีวัตถุประสงค์เพื่อ

1. ศึกษาค้นคว้าอัลกอริทึมที่เกี่ยวข้องกับการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย
2. ออกแบบและทดลองเพื่อปรับปรุงอัลกอริทึมการค้นหากฎความสัมพันธ์แบบเพิ่มขยายให้ได้ผลลัพธ์ที่ดีขึ้น

1.3 ขอบเขตของการวิจัย

งานวิจัยนี้ มุ่งศึกษาและพัฒนาปรับปรุงประสิทธิภาพของอัลกอริทึมทางด้านการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย (Incremental mining on association rule) ใช้ระยะเวลาในการโครงการรวมทั้งสิ้น 18 เดือน

1.4 ขั้นตอนการศึกษา

ขั้นตอนในการศึกษางานวิจัยการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย ประกอบด้วย 3 ขั้นตอนหลัก ดังนี้

1. กำหนดหัวข้อ วัตถุประสงค์และขอบเขตการดำเนินการวิจัย
2. ศึกษา แนวคิดและบทความงานวิจัยที่เกี่ยวข้องกับงานวิจัย
3. เรียบเรียงและจัดทำเอกสาร

1.5 นิยามศัพท์

ในงานวิจัยนี้ มีการใช้คำศัพท์เฉพาะในการศึกษาอัลกอริทึมด้านการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย (Incremental Association Rule) เพื่อให้มีความเข้าใจตรงกัน ผู้วิจัยได้นิยามศัพท์ที่สำคัญและนิยมใช้ในงานวิจัย ดังนี้

Association Rule หรือ กฎความสัมพันธ์ เป็นกระบวนการหารูปแบบของข้อมูลที่น่าสนใจในฐานข้อมูลแล้วแสดงออกมาในรูปของกฎความสัมพันธ์ X → Y

Original Database หมายถึง ฐานข้อมูลดั้งเดิมที่ยังไม่ถูกดำเนินการเพิ่ม ลบ หรือปรับปรุงข้อมูลในฐานข้อมูล ในงานวิจัยนี้จะใช้สัญลักษณ์เป็น DB

Increment Database หมายถึง ชุดข้อมูลหรือฐานข้อมูลใหม่ที่จะถูกเพิ่มเข้าไปในฐานข้อมูลเดิม (Original Database) ในงานวิจัยนี้จะใช้สัญลักษณ์เป็น db

Updated Database หมายถึง ฐานข้อมูลที่ได้รับการปรับปรุงแล้ว นั่นคือ ฐานข้อมูลที่มีการเพิ่มข้อมูลใหม่เข้าไป หรืออาจมีชุดข้อมูลบางส่วนที่ถูกลบหรือปรับปรุงแก้ไข ในงานวิจัยนี้จะใช้สัญลักษณ์ UD

Frequent Itemset หรือ Large Itemset หมายถึง เซตของไอเท็มที่มีความถี่ที่เรียกว่าค่าสนับสนุน (support count) มากกว่าหรือเท่ากับค่าที่สนับสนุน (min_sup) ที่ผู้ใช้กำหนดในตอนเริ่มต้น ซึ่งไอเท็มเซ็ตที่ถูกเรียกว่า Frequent Itemset นี้ จะถูกนำไปสร้างกฎความสัมพันธ์ต่อไป

Infrequent Itemset หรือ Small Itemset หมายถึง เซตของไอเท็มที่มีความถี่ที่เรียกว่าค่าสนับสนุน (support count) น้อยกว่าค่าที่สนับสนุน (min_sup) ที่ผู้ใช้กำหนดในตอนเริ่มต้น ซึ่งไอเท็มเซ็ตที่ถูกเรียกว่า Infrequent Itemset นี้ จะไม่ถูกนำไปสร้างกฎความสัมพันธ์

Candidate Itemset หมายถึง ไอเท็มเซตตัวเลือก เป็นชุดของไอเท็มเซ็ตที่ได้จากการเชื่อมความสัมพันธ์ของไอเท็มเซตที่เป็น frequent itemset ก่อนหน้า โดยแคนดิเดตไอเท็มเซ็ตที่ได้จะถูกนำไปทดสอบกับค่าสนับสนุนต่ำสุดที่ผู้ใช้กำหนด หากแคนดิเดตไอเท็มเซตตัวใดที่มีค่าสนับสนุนมากกว่ากว่าหรือเท่าค่าสนับสนุนต่ำสุด จะถูกเรียกว่า frequent itemset แต่ถ้ามีค่าสนับสนุนน้อยกว่าค่าสนับสนุนต่ำสุด แคนดิเดตไอเท็มเซตตัวนั้นจะถูกตัดทิ้ง (prune) ไป

Minimum Support หรือ min_sup หมายถึง ค่าสนับสนุนต่ำสุดที่ผู้ใช้เป็นผู้กำหนดเพื่อใช้ในการทดสอบความสามารถในการเป็น frequent itemset ของแต่ละไอเท็ม

Minimum Confidence หรือ min_conf หมายถึง ค่าความเชื่อมั่นต่ำสุดที่ผู้ใช้เป็นผู้กำหนดเพื่อใช้ในการทดสอบว่ากฎความสัมพันธ์ใดจะเป็นกฎที่มีความน่าสนใจ

k-itemset หมายถึง เซตของไอเท็มที่ประกอบด้วยไอเทมจำนวน k ตัว โดยที่ $k \geq 1$ ตัวอย่างเช่น

k = 1 หรือ 1-itemset หมายถึงเซตของไอเท็มที่ประกอบด้วยไอเท็ม 1 ตัว นิยมเขียนอยู่ในรูป {A, B , C , D} เมื่อ A B C และ D คือไอเท็ม

k = 2 หรือ 2-itemset หมายถึงเซตของไอเท็มที่ประกอบด้วยไอเท็ม 2 ตัว นิยมเขียนอยู่ในรูป {{A,B}, {A,C}, {A,D}, {B,C}, {B,D}, {C,D}} เป็นต้น

บทที่ 2

ทฤษฎีพื้นฐานที่ใช้ในงานวิจัย และงานวิจัยที่เกี่ยวข้อง (Literature Reviewed)

การค้นหากฎความสัมพันธ์ (discovery of association rules) นิยมใช้กันอย่างแพร่หลายในหลายสาขา โดยเฉพาะอย่างยิ่งการวิเคราะห์การซื้อสินค้าของลูกค้า (market basket analysis) ที่ศึกษาถึงพฤติกรรรมการซื้อสินค้าของลูกค้าว่า เมื่อลูกค้าซื้อสินค้าชนิดหนึ่งแล้ว จะซื้อสินค้าใดควบคู่ไปด้วย แต่ด้วยธรรมชาติของข้อมูลที่มีการเพิ่มขึ้นอย่างต่อเนื่อง ทำให้กฎความสัมพันธ์เดิมที่เคยได้รับต้องมีการเปลี่ยนแปลงตามไปด้วย

ในบทนี้จะกล่าวถึงทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องกับงานวิจัยนี้ ซึ่งประกอบด้วย

1. การไมนิ่งกฎความสัมพันธ์ (Association rules Mining)
2. การเพิ่มขยายการค้นหากฎความสัมพันธ์ (Incremental mining on association rules)

โดยในแต่ละทฤษฎีมีรายละเอียดดังนี้

2.1 การไมนิ่งกฎความสัมพันธ์ (Association rules mining)

การไมนิ่งกฎความสัมพันธ์ (association rules mining) ได้ถูกนำเสนอเมื่อปี 1993 [1] เพื่อใช้ในการค้นหาความสัมพันธ์ที่น่าสนใจระหว่างไอเท็มในชุดข้อมูลที่เป็นแบบรายการ (transactional dataset) ทั้งนี้โมเดลทางคณิตศาสตร์ที่ได้นำเสนอใน [1] ที่ใช้กระบวนการไมนิ่งกฎความสัมพันธ์เป็นดังนี้

กำหนดให้

I เป็นเซตของไอเท็ม โดย $I = \{i_1, i_2, \ldots, i_m\}$

T เป็น รายการ (transaction) ซึ่งแต่ละ transaction เป็นเซตของไอเท็ม นั่นคือ $T \subseteq I$ และ T แต่ละตัวจะสัมพันธ์กับตัวระบุ (identifier) ที่เรียกว่า TID (Transaction Identifier) ทั้งนี้จำนวนการซื้อของแต่ละไอเท็มจะไม่ถูกนำมาพิจารณา นั่นคือ แต่ละไอเท็มจะมีค่าเป็นลักษณะ binary variable คือ ซื้อ หรือ ไม่ซื้อ

D เป็นเซตของ รายการ (transaction) ในฐานข้อมูล

สมมติกำหนดให้ ไอเท็ม X เป็นไอเท็มในรายการหนึ่งๆ นั่นคือ $X \subseteq T$

กฎความสัมพันธ์จะแสดงอยู่ในรูปของ IF...THEN rule ซึ่งจะแสดงในแบบฟอร์มของ $X \Rightarrow Y$ โดย $X \subset I$, $Y \subset I$ และ $X \cap Y = \varnothing$

Itemsetใดๆ ที่สามารถนำไปแสดงเป็นกฎความสัมพันธ์ $X \Rightarrow Y$ จะต้องมีค่า support s ซึ่งเป็นจำนวนเปอร์เซ็นต์ของข้อมูล transaction ใน D ที่มีทั้งไอเท็ม A และ B ($A \cup B$) นั่นคือ เป็นค่าความน่าจะเป็นที่จะเกิดไอเท็ม A และ B พร้อมกัน ($P(A \cup B)$)

ส่วนกฎ $X \Rightarrow Y$ ใดๆ ที่จะเป็นกฎที่น่าสนใจ จะต้องมีค่า confidence c ในเซตของข้อมูล transaction ใน D เมื่อ c เป็นเปอร์เซ็นต์ของข้อมูล transaction ใน D ที่มี A แล้ว จะต้องมี B ด้วย นั่นคือความน่าจะเป็นแบบมีเงื่อนไข ($P(B|A)$)

ทั้งนี้ แนวคิดเกี่ยวกับค่า support และค่า confidence สำหรับการหากฎความสัมพันธ์สามารถแสดงให้อยู่ในรูปของความน่าจะเป็นได้ดังนี้

$$Support(X \Rightarrow Y) = P(X \cup Y)$$
$$confidence(X \Rightarrow Y) = P(X|Y)$$

สำหรับค่า support และค่า confidence หรือบางครั้งเรียกว่า minimum support และ minimum confidence

ในการหากฎความสัมพันธ์ จะต้องหากฎความสัมพันธ์ที่กฎนั้นจะต้องมีค่า confidence และค่า support มากกว่าค่า confidence และ support ที่กำหนดมาในตอนเริ่มต้น ทั้งนี้ การหากฎความสัมพันธ์ประกอบด้วยการทำงาน 2 ขั้นตอน ได้แก่

1. การหา frequent itemset ทั้งหมด (Find all frequent itemset)

ไอเท็ม X ใดๆ ที่สามารถเป็น frequent itemset หรือ large itemset ได้จะต้องมีค่า support ที่มากกว่าค่า minimum support ที่กำหนดไว้แต่เริ่มแรก นั่นคือ $\{X \mid X.support \geq S_{min}\}$ เป้าหมายของขั้นตอนนี้คือการหา Large k-itemset หรือ $\{L_1, L_2, \ldots, L_k\}$ นั่นเอง

2. การสร้างกฎจาก frequent itemset ที่ได้มาจากการคำนวณ (Generate association rules from the frequent itemset)

ในขั้นตอนนี้เป็นการสร้างกฎความสัมพันธ์จาก Frequent itemset หรือ large itemset ที่จากขั้นตอนที่ 1

ในงานวิจัยทางด้านการค้นหากฎความสัมพันธ์ (Association rules mining) อัลกอริทึมที่ได้รับความนิยมที่นำมาใช้ในการหากฎความสัมพันธ์คือ Apriori [2] สามารถอธิบายได้ดังนี้

Apriori Algorithm [2]

อัลกอริทึม Apriori เป็นอัลกอริทึมที่ค่อนข้างมีบทบาทและเป็นที่นิยมในงานวิจัยด้านการmining association rules อัลกอริทึมนี้จะมีการคำนวณหา large itemset แสดงได้ดังภาพที่ 2.1 ในการทำงาน Apriori จะ scan แต่ละ transaction ในฐานข้อมูล โดยมีการวนซ้ำหลายครั้ง โดย large k-itemset (L_k) ที่คำนวณได้แต่ละรอบ จะคำนวณจาก candidate k-1 itemset (C_{k-1}) ซึ่งเป็นในลักษณะการทำงานของ level-wise-step ในการคำนวณหา large itemset ของอัลกอริทึมนี้จะแบ่งการทำงานออกเป็น 2 ขั้นตอนหลัก ได้แก่

1. ขั้นตอนการ Join (join step)

เป็นขั้นตอนในการนำ L_{k-1} ($k \geq 2$) มาทำการ join operation เพื่อสร้าง C_k ซึ่งจะใช้ในการคำนวณหา L_k ต่อในรอบถัดไป เช่น C_2 ได้มาจากการทำ join operation ระหว่าง $L_1 * L_1$ จากนั้นจึงนำ C_2 ที่คำนวณได้ไปทำการหา L_2 (โดยการเปรียบเทียบกับค่า minimum support) เมื่อได้ L_2 ก็จะทำ join operation ระหว่าง $L_2 * L_2$ เพื่อให้ได้ C_3 เพื่อนำไปคำนวณหา L_3 เป็นลำดับถัดไป และจะทำเช่นไปเรื่อยๆ จนกระทั่ง $C_k = \varnothing$ การทำ join operation แสดงได้ดังภาพที่ 2.2

2. ขั้นตอนการ prune (prune step)

เป็นขั้นตอนที่ใช้ในการ prune itemset ของ C_k เพื่อพิจารณาหา itemset ที่มีคุณสมบัติเป็น L_k นั่นคือ itemset ใดๆ ใน C_k ที่ X.support < min_sup จะถูก prune ออกไป และ itemset ที่เหลือ (ซึ่งมีค่า X.support ≥ min_sup) จะจัดเป็น L_k ขั้นตอนการ prune step แสดงได้ดังภาพที่ 2.3 ทั้งนี้ยังได้มีการกำหนดคุณสมบัติของ itemset ไว้ดังนี้ “ถ้าซับเซต ของ k-1 itemset ใดๆ ใน C_k ที่ไม่ได้เป็นสมาชิกของ L_{k-1} แล้ว itemset นั้นๆ จะไม่สามารถเป็น L_k ได้ ซึ่งสามารถ prune itemset นั้นๆ ออกจาก C_k ได้”

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

ภาพที่ 2.1 การคำนวณหา Large itemset ของApriori Algorithm

```
insert into Ck
select p.item1, p.item2, ..., p.itemk-1, q.itemk-1
from Lk-1 p, Lk-1 q
where p.item1 = q.item1, ..., p.itemk-2 = q.itemk-2,
      p.itemk-1 < q.itemk-1;
```

ภาพที่ 2.2 ขั้นตอนการทำ join step

```
forall itemsets c ∈ Ck do
   forall (k−1)-subsets s of c do
      if (s ∉ Lk-1) then
         delete c from Ck;
```

ภาพที่ 2.3 ขั้นตอนการทำ prune step

เมื่อเสร็จสิ้นขั้นตอนการหา Large itemset แล้ว ขั้นตอนถัดไปคือการหากฎความสัมพันธ์ที่เป็นตามค่า minimum support และ ค่า minimum confidence ที่กำหนดไว้ ซึ่งค่าทั้งสองดังกล่าวจะมีช่วงระหว่าง 0 – 100% หากกฎความสัมพันธ์ใดๆ ที่เป็นไปตามค่า min_sup และ min_conf ดังกล่าว จะจัดว่ากฎนั้นเป็นกฎที่น่าสนใจ หรือกฎที่เข้มแข็ง (strong rule) ในทางกลับกัน หากกฎความสัมพันธ์ใดๆ ที่ไม่เป็นไปตามค่า min_sup และ min_conf ดังกล่าว จะจัดว่ากฎนั้นเป็นกฎที่ไม่น่าสนใจหรือเป็นกฎที่อ่อนแอ (weak rule)

ข้อเสียของ Apriori

1. จำเป็นต้องมีการ scan ฐานข้อมูลหลายครั้ง และมีการสร้าง candidate itemset จำนวนมาก ในการคำนวณแต่ละรอบ ซึ่งทำให้ใช้เวลาในการประมวลผลค่อนข้างนาน จึงทำให้มีการคิดค้นหาอัลกอริทึมที่ช่วยลดการสร้าง candidate itemset และ ลดการ scan ฐานข้อมูล
2. ไม่เหมาะสมสำหรับการสร้างกฎความสัมพันธ์ ที่ข้อมูลเป็นลักษณะ Numeric หรือ Boolean
3. หากมีการเพิ่มรายการข้อมูลใหม่ เข้าไปในฐานข้อมูล จะต้องหา กฎความสัมพันธ์ใหม่ด้วยการ scan ฐานข้อมูลใหม่ทั้งหมด

2.2 การเพิ่มขยายกฎความสัมพันธ์ (Incremental mining on association rules)

ธรรมชาติของ Dynamic Database รายการข้อมูล (transaction) จะถูกเพิ่มเข้ามาในฐานข้อมูลอยู่ตลอดเวลา จึงทำให้เกิดกฎความสัมพันธ์ใหม่ที่น่าสนใจเกิดขึ้น ขณะเดียวกัน กฎความสัมพันธ์เดิมอาจจะกลายเป็นกฎที่ไม่น่าสนใจอีกต่อไป เหตุการณ์ที่เกิดขึ้นเมื่อมีการเพิ่มข้อมูลเข้ามาใหม่ประกอบด้วย 4 เหตุการณ์ [3] ดังนี้

1. itemset ที่เป็น Large itemset ใน ฐานข้อมูลเดิม ยังคงเป็น Large itemset ใน ฐานข้อมูลใหม่เช่นเดิม
2. itemset ที่เป็น Large itemset ใน ฐานข้อมูลเดิม เปลี่ยนเป็น Small itemset ใน ฐานข้อมูลใหม่
3. itemset ที่เป็น Small itemset ใน ฐานข้อมูลเดิม เปลี่ยนเป็น Large itemset ใน ฐานข้อมูลใหม่
4. itemset ที่เป็น Small itemset ใน ฐานข้อมูลเดิม ยังคงเป็น Small itemset ใน ฐานข้อมูลใหม่เช่นเดิม

ในการรักษาไว้ซึ่งความถูกต้องของกฎความสัมพันธ์ (maintenance of association rules) จึงเป็นปัญหาสำคัญ ที่มีผู้วิจัยหลายรายพยายามศึกษาหาวิธีทางแก้ไขปัญหาดังกล่าว ซึ่งงานวิจัยฉบับนี้ ได้ศึกษางานวิจัยที่เกี่ยวข้องกับการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย ดังนี้

FUP [4]

อัลกอริทึม FUP (Fast UPdate Algorithm) เป็นงานวิจัยแรกที่นำเสนอเทคนิคการทำ incremental updating technique เพื่อ maintenance association rules เมื่อมีการเพิ่มข้อมูลใหม่เข้ามาในฐานข้อมูล การทำงานของ FUP อาศัยหลักการเดียวกับ Apriori ซึ่งจะมีการทำงานหลายรอบ โดยรอบแรกจะเริ่มตั้งแต่ 1-itemset ไปจนถึง k-itemset ทั้งนี้อัลกอริทึมนี้จะทำงานภายใต้การอนุมานว่าค่า minimum support และค่า minimum confidence คงที่

ความหมายของสัญลักษณ์ต่างๆ ที่ใช้ในอัลกอริทึม FUP มีรายละเอียดดังนี้

DB หมายถึง original Database

db หมายถึง increment database

D หมายถึง จำนวน transaction ที่มีอยู่ใน original database

d หมายถึง จำนวน transaction ที่มีอยู่ใน increment database

s หมายถึง ค่า minimum support

L_k หมายถึง Large k-itemset ใน original database เมื่อ k=1,2,...

L'_k หมายถึง Large k-itemset ใน updated database ($DB \cup db$) เมื่อ k=1,2,...

$X.support_D$ หมายถึง ค่าความถี่ (support) ของ ไอเท็ม X ใน original database

$X.support_d$ หมายถึง ค่าความถี่ (support) ของ ไอเท็ม X ใน increment database

$X.support_{UD}$ หมายถึง ค่าความถี่ (support) ของ ไอเท็ม X ใน updated database

FUP จะแบ่งการทำงานออกเป็น 2 ส่วนหลักคือ First iteration และ Second iteration and beyond รายละเอียดขั้นตอนการทำงานของ FUP ทั้งสองส่วน อธิบายดังนี้

1. First iteration

การทำงานในส่วนนี้ จะเป็นการหา prune ไอเท็มที่เป็น loser item และ หาไอเท็มที่เป็น winner item ซึ่งเป็น Large 1-itemset

1.1 scan db เพื่อทำการปรับปรุงค่า support ของไอเท็ม เพื่อ prune ไอเท็มที่เป็น loser ออกไป และหาไอเท็มที่เป็น winner โดยมีหลักการพิจารณาดังนี้

1.1.1 กรณี $X \in L_1$ ให้นำค่า support ของไอเท็ม X ใน DB และ db มารวมกัน จะได้ $X.support_{UD} = X.support_D + X.support_d$ จากนั้นทำการตรวจสอบค่า support ที่ได้ โดย

ถ้า $X.support_{UD} \geq s \times (D+d)$ แสดงว่า ไอเท็ม X นั้นๆ เป็น winner item และให้ $X \in L'_1$

ถ้า $X.support_{UD} < s \times (D+d)$ แสดงว่า ไอเท็ม X นั้นๆ เป็น loser item และจะ prune ไอเท็ม X นั้นทิ้งไป

1.1.2 กรณี $X \notin L_1$ จะมีการพิจารณา 2 ส่วน คือ

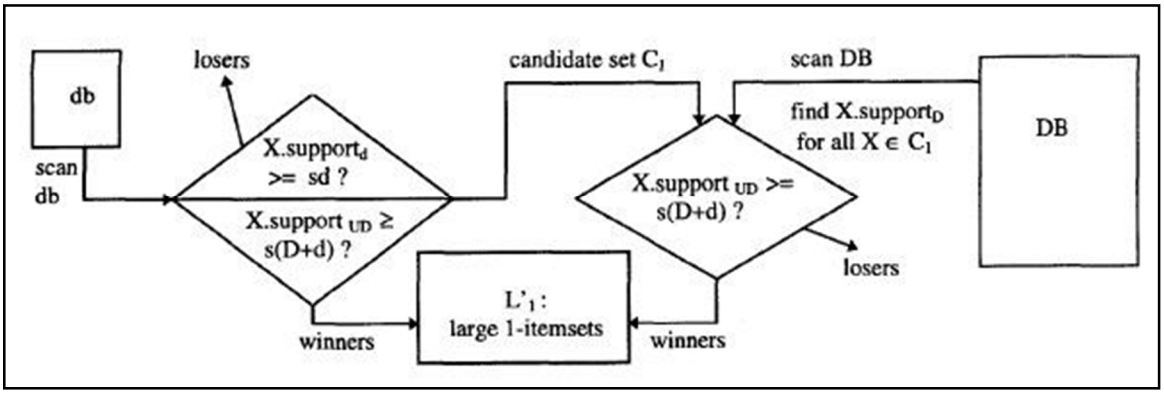
1.1.2.1 prune ไอเท็มที่ไม่มีโอกาสเป็น L'_1 ได้ โดยพิจารณาจาก $X \notin L_1$ และ $X.support_d < s \times d$ แสดงว่าไอเท็ม X นั้น เป็น loser item จะทำการ prune ไอเท็ม X นั้นทิ้งไป ซึ่งจะช่วยลดจำนวนการ scan ไอเท็ม ในฐานข้อมูลเดิม

1.1.2.2 ตรวจสอบไอเท็มที่มีโอกาสเป็น L'_1 โดยพิจารณาจาก $X \notin L_1$ และ $X.support_d \geq s \times d$ ให้นำ ไอเท็ม X ดังกล่าวไป scan ใน DB จากนั้นจึงนำค่า $X.support_{UD}$ ที่ได้มาตรวจสอบว่าเป็น loser item หรือ winner item โดย

ถ้า $X.support_{UD} \geq s \times (D+d)$ แสดงว่า ไอเท็ม X นั้นๆ เป็น winner item และให้ $X \in L_1'$

ถ้า $X.support_{UD} < s \times (D+d)$ แสดงว่า ไอเท็ม X นั้นๆ เป็น loser item จะทำการ prune ไอเท็ม X นั้นทิ้งไป

เมื่อเสร็จสิ้นกระบวนการทำงานในรอบนี้ จะได้ Large 1-itemset (L'_1) ใน ฐานข้อมูลที่ปรับปรุงแล้ว (updated database) ทั้งนี้กระบวนการทำงานในรอบ first iteration เพื่อหา large 1-itemset ของ FUP แสดงได้ดังภาพที่ 2.4 และอัลกอริทึมแสดงการทำงานในรอบ นี้แสดงได้ดังภาพที่ 2.5



ภาพที่ 2.4 กระบวนการทำงานในรอบ First iteration ของ FUP

2. Second iteration and beyond

การทำงานในส่วนนี้จะเป็นการหา Large k-itemset เมื่อ $k \geq 2$ ซึ่งจะมีส่วนหนึ่ง ที่มีหลักการทำงานคล้าย first iteration นั่นคือการหาไอเท็มที่เป็น loser ที่ไม่สามารถเป็น L_k เพื่อลดการ scan k-itemset เมื่อ $k \geq 2$ ใน db โดยอาศัยแนวคิดที่ว่า “ถ้าไอเท็ม X ใดๆ ที่เป็น loser item ในการประมวลผลรอบที่ k-1 (เมื่อ $k \geq 2$) แล้ว itemset ใดๆ ของ L_k ในฐานข้อมูลเดิมที่มีไอเท็ม X ดังกล่าวเป็น subset อยู่จะไม่สามารถเป็น winer item ในรอบที่ k” จากแนวคิดดังกล่าว ใน second iteration and beyond จะมีการทำงานดังนี้

2.1 หา itemset ที่เป็นสมาชิกของ L_2 และ L'_2 นั่นคือ หา itemset ที่เป็น large itemset ทั้งในฐานข้อมูลเดิม และฐานข้อมูลที่ปรับปรุงแล้ว

ขั้นตอนนี้จะ prune loser item ที่ไม่สามารถเป็น L_2 เพื่อลดจำนวนไอเท็มที่จะ scan ใน db โดยพิจารณาจาก $Y = L_1 - L'_1$(ไอเท็ม Y ใดๆ ที่เป็นสมาชิกของ L_1 แต่ไม่เป็นสมาชิกของ L'_1) นั่นคือเมื่อ $X \in L_2$ ที่มีไอเท็ม Y เป็นซับเซต จะไม่สามารถเป็น Large itemset ได้ ดังนั้น ไอเท็ม X ดังกล่าวจะถูก prune ทิ้งไป

ตัวอย่าง

$L_1 = \{A,B,C\}$ $L'_1 = \{A,C,D\}$ $L_2 = \{AB,AC\}$

$L_1 - L'_1 = \{B\}$

จากตัวอย่าง จะเห็นได้ {B} เป็นสมาชิกใน L_1 แต่ไม่เป็นสมาชิกใน L'_1 ดังนั้น itemset ใด ๆ ใน L_2 ที่มี {B} เป็นสมาชิก itemset นั้นๆ จะเรียกว่า loser item และถูก prune ออกไปโดยไม่ต้องนำไป scan ใน db ซึ่งจากตัวอย่าง itemset ใน L_2 ที่มี {B} เป็นซับเซตคือ {AB} ดังนั้น {AB} จะถูก prune ออกจาก L_2 และ L_2 จะเหลือสมาชิกอยู่คือ {AC} ซึ่ง {AC} จะถูกนำไป scan ใน db เพื่อปรับปรุงค่า support

ในการ scan สมาชิกตัวที่เหลือใน L_2 หากพบว่า

- $X.support_{UD} \geq s \times (D+d)$
  itemset นั้นๆ จะเรียกว่า winner item แล้วจะถูกนำไปเก็บใน L'_2
- $X.support_{UD} < s \times (D+d)$
  Itemset นั้นๆ จะเรียกว่า loser item ก็จะถูก prune ทิ้งไป

2.2 หา new Large 2-itemset (L'_2)

ในขั้นตอนนี้จะเริ่มคำนวณ candidate 2-itemset (C_2) ด้วยการ join operation ระหว่าง $L'_1 * L'_1$ เพื่อนำไป scan ใน db และหา L'2 โดยพิจารณาดังนี้

2.2.1 กรณี $X \in C_2$ และ $X \in L'_2$

ไม่ต้องนำ ไอเท็ม X ดังกล่าวไป scan ใน db เนื่องจาก X เป็นสมาชิกของ L'_2 แล้ว (ซึ่งคำนวณได้จากขั้นตอนที่ 2.1)

2.2.2 กรณี $X \in C_2$ และ $X \notin L'_2$

ให้นำ ไอเท็ม X ดังกล่าวไป scan ใน db แล้วตรวจสอบ

- ถ้า $X.support_d \geq s \times d$

ให้นำ ไอเท็ม X ดังกล่าวไป scan ใน DB แล้วปรับปรุงค่า support แล้วตรวจสอบหากพบว่า $X.support_{UD} \geq s \times (D+d)$ แล้ว ให้เพิ่มไอเท็ม X นั้นเป็นสมาชิกของ L'_2

- ถ้า $X.support_d < s \times d$

ให้ prune ไอเท็ม X ออกจาก C_2 และไม่ต้องนำ X ไป scan ใน DB

เมื่อเสร็จสิ้นขั้นตอนที่ 2.2 ผลลัพธ์ที่ได้คือ L'_2

2.3 ทำการวนซ้ำเช่นเดียวกับข้อที่ 2.1 เพื่อหา k-itemset (k≥3)ในรอบถัดไป

จุดเด่นของ FUP

1. มีการเก็บผลลัพธ์จากการไมนิ่งในฐานข้อมูลเดิมมาใช้ร่วมกับฐานข้อมูลใหม่ที่เพิ่มเข้ามา
2. สามารถลดจำนวน Candidate itemset ที่จะนำไปใช้ scan ในฐานข้อมูลเดิม

ข้อเสียของ FUP

1. สามารถใช้งานได้ในกรณีของการเพิ่มข้อมูลใหม่เข้าไปในฐานข้อมูลเดิมเท่านั้น ยังไม่สามารถใช้ได้กับกรณีการลบ และการ modification ในฐานข้อมูลได้
2. ยังจำเป็นต้องมีการ scan ฐานข้อมูลเดิม เพื่อหา new large k-itemset ในทุกรอบ k

Negative border [5]

อัลกอริทึม Negative border เป็นงานวิจัยที่ศึกษาเกี่ยวกับปัญหาการ maintenance association rules โดยอัลกอริทึมนี้สามารถกระทำได้ใน 2 กรณีคือ การเพิ่มข้อมูลใหม่เข้าและ และการลบข้อมูลเก่าออกไปจากฐานข้อมูลเดิม ทั้งนี้อัลกอริทึมนี้จะทำงานภายใต้การอนุมานว่า ค่า minimum support และค่า minimum confidence คงที่

หลักการของ Negative border คือ จะมีการเก็บค่า C_k ทั้ง $C_k \in L_k$ และ $C_k \notin L_k$ โดย $C_k \notin L_k$ จะเรียกว่า negative border ตัวอย่างเช่น

$C_1 = \{A, B, C, D, E\}$ $L_1 = \{A, B, E\}$

ดังนั้น negative border ของ L_1 หรือ $NBd(L_1) = \{C,D\}$

จากตัวอย่าง สามารถเขียนในรูปของสมการได้ดังนี้

$NBd(L_k) = C_k - L_k$ หรือ

$C_k = L_k \cup NBd(L_k)$

ความหมายของสัญลักษณ์ที่ใช้ในอัลกอริทึม Negative border มีรายละเอียดดังนี้

DB	หมายถึง	original database
db	หมายถึง	increment database
DB^+	หมายถึง	updated database
L^{DB}	หมายถึง	large itemset ใน original database
L^{db}	หมายถึง	large itemset ใน increment database
L^{DB+}	หมายถึง	large itemset ใน updated database
$NBd(L_k)$	หมายถึง	Negative border ของ large k-itemset เมื่อ $k \geq 1$
$NBd(L^{DB})$	หมายถึง	Negative border ใน original database
$NBd(L^{db})$	หมายถึง	Negative border ใน increment database
$NBd(L^{DB+})$	หมายถึง	Negative border ใน updated database
s	หมายถึง	itemset ใดๆ ในฐานข้อมูลทั้ง DB, db และ DB^+
s.count	หมายถึง	ค่าความถี่ของไอเท็ม s ที่เกิดขึ้น
$t_{DB}(s)$	หมายถึง	จำนวน transaction ใน DB ที่มีไอเท็ม s เป็นสมาชิก
$t_{db}(s)$	หมายถึง	จำนวน transaction ใน db ที่มีไอเท็ม s เป็นสมาชิก

ขั้นตอนการทำงานของ Negative border จะแบ่งการทำงานออกเป็น 2 ส่วน คือส่วนของการเพิ่มข้อมูลใหม่ (Addition of new transaction) และในส่วนของการลบข้อมูลเก่าออกไป (Deletion of existing transaction) อัลกอริทึม Negative border แสดงดังภาพที่ 2.7 โดยในแต่ละส่วนมีรายละเอียดการทำงานดังนี้

1. Addition of new transactions

อัลกอริทึม Negative border แสดงดังภาพที่ 2.7 ซึ่งมีรายละเอียดการทำงานดังนี้

1. ปรับปรุงค่า support ให้กับ itemset ที่เป็นสมาชิกของ L_k และ $NBd(L_k)$ ในฐานข้อมูลเดิม

เริ่มจากการ scan transaction ที่เข้ามาในฐานข้อมูลเพื่อ คำนวณหา large itemset ใน db (L_k^{db}) ในขณะเดียวกันให้ทำการปรับปรุงค่า support ของ itemset ที่เป็น L_k และ $NBd(L_k)$ ในฐานข้อมูลเดิมด้วย จากนั้นให้ทำการพิจารณาดังนี้

1.1 กรณี $s \in L^{DB}$

ถ้า $t_{DB}(s) + t_{db}(s) < min\_sup \times (t_{DB}+t_{db})$ ให้ prune ไอเท็ม s นั้นออกจาก L^{DB}

ถ้า $t_{DB}(s) + t_{db}(s) \geq min\_sup \times (t_{DB}+t_{db})$ ให้เพิ่มไอเท็ม s เข้าเป็นสมาชิกของ L^{DB+}

1.2 กรณี $s \in L^{db}$ และ $s \notin L^{DB}$ และ $s \in NBd(L^{DB})$

ถ้า $t_{DB}(s) + t_{db}(s) \geq min\_sup \times (t_{DB}+t_{db})$ ให้เพิ่มไอเท็ม s เข้าเป็นสมาชิกของ L^{DB+}

2. หลังจากทำการปรับปรุงค่า support ให้แก่ L_k และ $NBd(L_k)$ ในฐานข้อมูลเดิมเรียบร้อยแล้ว (จากขั้นตอนที่ 1) ผลลัพธ์ที่ได้คือ large itemset ในฐานข้อมูลปรับปรุง (L^{DB+}) จากนั้นจะทำการเปรียบความแตกต่างของ itemset ใน L^{DB} และ L^{DB+} ดังนี้

2.1 กรณี $L^{DB} = L^{DB+}$ (ไม่มีการเปลี่ยนของ itemset ใน original database และ updated database) หมายถึง itemset ที่เป็นสมาชิกของ negative border ไม่มีการเปลี่ยนแปลง นั่นคือ $NBd(L^{DB}) = NBd(L^{DB+})$

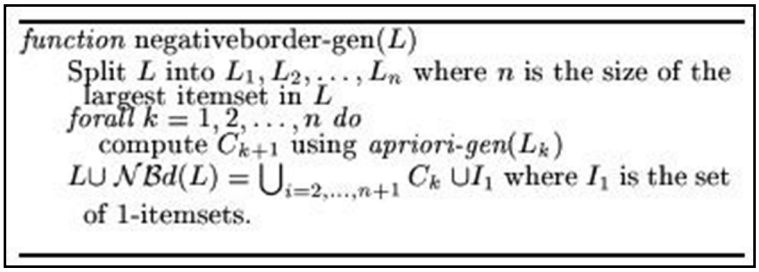
2.2 กรณี $L^{DB} \neq L^{DB+}$ (มีการเปลี่ยนของ itemset ใน original database และ updated database) หมายถึง itemset ที่เป็นสมาชิกของ negative border มีการเปลี่ยนแปลง นั่นคือ $NBd(L^{DB}) \neq NBd(L^{DB+})$ ให้ทำการคำนวณค่า ใหม่โดยใช้ฟังก์ชั่น Negativeborder-gen(L^{DB+}) แสดงได้ดังภาพ 2.8 โดยการนำเอา L_k^{DB+} ไปคำนวณหา $NBd(L_k^{DB+})$ ในแต่ละรอบ k ตาม level-wise

3. จากขั้นตอนที่ 2 เมื่อคำนวณหาค่า $NBd(L_k^{DB+})$ เรียบร้อยแล้ว จะนำค่า $L_k^{DB} \cup NBd(L_k^{DB})$ ของฐานข้อมูลเดิมมาเปรียบ $L_k^{DB+} \cup NBd(L_k^{DB+})$ เทียบกับฐานข้อมูลใหม่ เพื่อดูความเปลี่ยนแปลง ซึ่งถ้า $L_k^{DB} \cup NBd(L_k^{DB}) \neq L_k^{DB+} \cup NBd(L_k^{DB+})$ จะทำการหาค่า negative border closure ของ LDB+ และจะทำการ scan ฐานข้อมูลเดิมอีกครั้งเพื่อปรับปรุงค่า L_k^{DB} และ $NBd(L_k^{DB})$

function Update-Large-Itemset(L^{DB}, $\mathcal{NB}d(L^{DB})$, *db*)

//*DB* and *db* denote the number of transactions in the original database and the increment database respectively.
Compute L^{db}
for each itemset $s \in L^{DB} \cup \mathcal{NB}d(L^{DB})$ *do*
  $t_{db}(s)$ = number of transactions in *db* containing *s*
$L^{DB+} = \phi$
for each itemset $s \in L^{DB}$ *do*
  if $(t_{DB}(s) + t_{db}(s)) \geq minsup * (DB + db)$ *then*
    $L^{DB+} = L^{DB+} \cup s$
for each itemset $s \in L^{db}$ *do*
  if $s \notin L^{DB}$ and $s \in \mathcal{NB}d(L^{DB})$ and $(t_{DB}(s) + t_{db}(s)) \geq minsup * (DB + db)$ *then*
    $L^{DB+} = L^{DB+} \cup s$
if $L^{DB} \neq L^{DB+}$ *then*
  $\mathcal{NB}d(L^{DB+})$ = negativeborder-gen(L^{DB+})
else $\mathcal{NB}d(L^{DB+}) = \mathcal{NB}d(L^{DB})$
if $L^{DB} \cup \mathcal{NB}d(L^{DB}) \neq L^{DB+} \cup \mathcal{NB}d(L^{DB+})$ *then*
  $S = L^{DB+}$
  repeat
    compute $S = S \cup \mathcal{NB}d(S)$
  until *S* does not grow
  $L^{DB+} = \{x \in S | support(x) \geq minsup\}$
  //$support(x)$ is the support count of x in $DB \cup db$
  $\mathcal{NB}d(L^{DB+})$ = negativeborder-gen(L^{DB+})

ภาพที่ 2.5 อัลกอริทึมของ Negative Border



ภาพที่ 2.8 ฟังก์ชั่น Negative Border-gen

2. Deletion of existing transactions

ในกรณีที่มีข้อมูลถูกลบออกจาก ฐานข้อมูลเดิม จะทำการคำนวณค่า L_k^{DB-} และ $NBd(L_k^{DB-})$ ใหม่ ด้วยการนำเอาค่า s.count ใน db มาลบออกจาก L_k^{DB} และ $NBd(L_k^{DB})$ แล้วตรวจสอบค่า $s.count_{DB-db}$ กับค่า min_sup * (DB – db) เช่นเดียวกับขั้นตอนที่ 1-3 ในส่วนของการเพิ่มข้อมูลใหม่ (Addition of new transactions)

จุดเด่นของ Negative border

1. มีการใช้ itemset ที่เป็น negative border เป็นตัวตัดสินใจในการ scan ฐานข้อมูลเดิม
2. มีการ scan ฐานข้อมูลเดิมเพียงครั้งเดียว ในกรณีที่ $L_k^{DB} \cup NBd(L_k^{DB}) \neq L_k^{DB+} \cup NBd(L_k^{DB+})$

ข้อเสียของ negative border

1. ใช้ได้เฉพาะในกรณีที่ไม่มี new item เกิดขึ้น
2. การหา negative border closer ใช้เวลานานในการหา L_k เพราะต้องมีการวนซ้ำหลายรอบ เพื่อให้ได้ $L = L \cup NBd(L)$ ทั้งหมด
3. ต้องมีใช้พื้นที่เก็บข้อมูลเพิ่มขึ้นเพื่อใช้เก็บค่า negative border

บทที่ 3

อัลกอริทึมการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย (Batch Fast Update Algorithm)

การค้นหากฎความสัมพันธ์ เป็นเทคนิคที่สำคัญของกระบวนการทำเหมืองข้อมูล (Data Mining) เพื่อค้นหาความสัมพันธ์ระหว่างข้อมูลในฐานข้อมูล และสกัดรูปแบบข้อมูลที่น่าสนใจให้ออกมาอยู่ในรูปแบบของกฎความสัมพันธ์ if X then Y โดยมีหลักการทำงานหลัก 2 ขั้นตอนได้แก่ 1) การค้นหาไอเท็มเซ็ตที่เรียกว่าฟรีเคว้นท์ไอเท็มเซ็ต ซึ่งเป็นไอเท็มเซตที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ (minimum support) ที่ผู้ใช้กำหนด และ 2) การนำฟรีเคว้นท์ไอเท็มเซ็ตที่หาได้จากข้อแรกมาสร้างกฎความสัมพันธ์ ซึ่งกฎความสัมพันธ์ที่น่าสนใจ จะเป็นกฎความสัมพันธ์ที่มีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ (minimum confidence) ที่ผู้ใช้กำหนด

โดยทั่วไปแล้ว อัลกอริทึมที่ได้รับการยอมรับและเป็นที่นิยมในการนำมาค้นหากฎความสัมพันธ์คืออัลกอริทึม Apriori [3] ที่ถูกเสนอโดย Agrawal โดยในการค้นหาฟรีเคว้นไอเท็มเซ็ตของ Apriori จะประกอบด้วยขั้นตอนที่สำคัญ 2 ขั้นตอนหลัก คือ การเชื่อมไอเท็มเซ็ต (join) และการตัด (prune) ไอเท็มเซ็ตที่ไม่มีโอกาสเป็นฟรีเคว้นท์ไอเท็มเซ็ตทิ้งไป อย่างไรก็ดี เมื่อมีการเพิ่มชุดข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม การค้นหากฎความสัมพันธ์ด้วยวิธีการของ Apriori จะต้องทำประมวลผลข้อมูลทั้งหมดที่อยู่ฐานข้อมูล นั่นคือ การหาแคนดิเดตไอเท็มเซตใหม่ และสแกนหาสนับสนุนของแคนดิเดตไอเท็มเซ็ตใหม่ในทุกๆ รอบ ซึ่งทำให้เวลาที่ใช้ในการประมวลผลถูกใช้มากเกินความจำเป็น และไม่เกิดประสิทธิภาพในการทำงาน ยกตัวอย่างเช่น สมมติมีการชุดข้อมูลใหม่จำนวน 3 ชุดที่จะถูกเพิ่มเข้าสู่ฐานข้อมูลเดิม นั่นหมายถึง หากต้องการปรับปรุงกฎความสัมพันธ์ในทุกๆ รอบที่มีการเพิ่มข้อมูลชุดใหม่เข้ามา Apriori จะต้องประมวลผลฐานข้อมูลทั้งเก่าและใหม่ตั้งแต่ต้นรวมเป็น 3 รอบ

ด้วยข้อด้อยดังกล่าวของ Apriori จึงได้มีผู้นำเสนอการปรับปรุงกฎความสัมพันธ์เมื่อมีการเพิ่มข้อมูลชุดใหม่เข้ามาในฐานข้อมูลเดิม Cheung และคณะ [4] ได้ เสนออัลกอริทึม FUP (Fast UPdate algorithm) ซึ่งเป็นอัลกอริทึมสำหรับการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย เมื่อมีการเพิ่มข้อมูลใหม่เข้ามา โดยอาศัยองค์ความรู้เดิมจากการไมนิ่งในฐานข้อมูลเดิมมาช่วยเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ นั่นคือ FUP จะใช้ฟรีเคว้นไอเท็มเซ็ตที่ได้จากการประมวลผลในฐานข้อมูลเดิม มาช่วยลดจำนวนแคนดิเดตไอเท็มเซ็ตที่จะต้องถูกนำไปสแกนในฐานข้อมูลเดิม ด้วยวิธีการประมวลผลดังกล่าวจึงทำให้ FUP ใช้เวลาในการประมวลผลน้อยกว่า Apriori

อย่างไรก็ตาม ในแต่ละรอบ k เมื่อมีการค้นพบแคนดิเดตไอเท็มเซตที่ไม่ได้เป็นสมาชิกของฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลเดิม แคนดิเดตไอเท็มเซ็ตนั้นๆ จะถูกนำไปสแกนในฐานข้อมูลเดิมทุกรอบที่ k นั่นแปลว่า หากขนาดของฟรีเคว้นไอเท็มเซตในฐานข้อมูลปรับปรุงมีค่าเท่ากับ 5 (k=5) ย่อมหมายความว่า FUP จะต้องนำแคนดิเดตไอเท็มเซ็ตไปสแกนในฐานข้อมูลเดิมรวมจำนวนทั้งสิ้น 5 รอบ เช่นเดียวกับ Apriori ต่างกันตรงที่ จำนวนแคนดิเดตไอเท็มเซ็ตที่ถูกนำไปสแกนของ FUP จะมีจำนวนน้อยกว่า Apriori เท่านั้น

เพื่อลดจำนวนการสแกนฐานข้อมูลเดิมให้เหลือจำนวนรอบการสแกนที่น้อยที่สุด ผู้วิจัยจึงได้ศึกษาเพื่อปรับปรุงอัลกอริทึม FUP ให้ทำงานโดยเหลือจำนวนรอบของการประมวลในฐานข้อมูลเดิมเพียง

รอบเดียว โดยอัลกอริทึมที่ปรับปรุงประสิทธิภาพการทำงานของ FUP นี้ ผู้วิจัยจะเรียกว่าอัลกอริทึม BFUP (Batch Fast UPdate Algorithm)

3.1 ข้อกำหนดเบื้องต้นของอัลกอริทึม

ด้วยอัลกอริทึม BFUP เป็นอัลกอริทึมที่อยู่ในงานวิจัยด้านการค้นหากฎความสัมพันธ์ ดังนั้นจึงต้องกำหนดสมมติฐานเบื้องต้นตามหลักการพื้นฐานทั่วไปของการค้นกฎความสัมพันธ์แบบเพิ่มขยายดังนี้

1. การกำหนดค่าทดสอบเบื้องต้นให้คงที่

ในการค้นหากฎความสัมพันธ์โดยทั่วไป มีค่าทดสอบเบื้องต้นพื้นฐาน 2 ค่า ซึ่งกำหนดโดยผู้ใช้ ได้แก่ ค่าสนับสนุนขั้นต่ำ (minimum support) และค่าความเชื่อมั่นขั้นต่ำ (minimum confidence) ในงานวิจัยฉบับนี้ อัลกอริทึม BFUP จะประมวลผลด้วยจะกำหนดค่าทดสอบเบื้องต้นดังกล่าวให้เป็นค่าคงที่ไม่มีการเปลี่ยนแปลง

2. ข้อกำหนดเกี่ยวกับการปรับปรุงฐานข้อมูล

การค้นหากฎความสัมพันธ์แบบเพิ่มขยาย จะเกี่ยวข้องกับการเปลี่ยนแปลงของฐานข้อมูลเดิม ซึ่งโดยทั่วไปแล้วการเปลี่ยนแปลงหรือการปรับปรุงฐานข้อมูลเดิม จะมีการปรับปรุงเปลี่ยนแปลงอยู่ 3 ลักษณะ ได้แก่

- การเพิ่มชุดข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม(Insert)
- การลบข้อมูลเดิมออกจากฐานข้อมูลเดิม (Delete)
- การปรับปรุงแก้ไขข้อมูลในฐานข้อมูลเดิม (Update)

ในงานวิจัยฉบับนี้ จะเป็นการศึกษาการปรับปรุงประสิทธิภาพของอัลกอริทึมการค้นหากฎความสัมพันธ์แบบเพิ่มขยายเฉพาะในกรณีแรก คือกรณีที่มีการเพิ่มชุดข้อมูลใหม่เข้าไปในฐานข้อมูลเดิมเท่านั้น

3.2 สัญลักษณ์ต่างๆ ที่ใช้ในงานวิจัย

เพื่อให้เป็นที่เข้าใจตรงกันสำหรับผู้วิจัยและผู้ที่สนใจศึกษางานวิจัยฉบับนี้ ผู้วิจัยได้กำหนดสัญลักษณ์ต่างๆ พร้อมความหมายที่ใช้ในงานวิจัย เพื่อให้ผู้ศึกษาเข้าใจตรงกัน ดังนี้

สัญลักษณ์	**ความหมาย**
DB	ฐานข้อมูลเดิม (Original Database)
db	ฐานข้อมูลใหม่หรือชุดข้อมูลใหม่ (Increment Database) ที่ถูกเพิ่มเข้าไปในฐานข้อมูลเดิม
UD	ฐานข้อมูลปรับปรุง (Updated Database) ซึ่งเป็นฐานข้อมูลที่ฐานข้อมูลเดิมและฐานข้อมูลใหม่รวมกันเรียบร้อยแล้ว UD = DB$\cup$db
|DB|	จำนวนทรานแซคชั่นในฐานข้อมูลเดิม
|db|	จำนวนทรานแซคชั่นในฐานข้อมูลใหม่

|UD|	จำนวนทรานแซคชั่นในฐานข้อมูลปรับปรุง |UD| = |DB|+|db|
s	ค่าสนับสนุนขั้นต่ำ (minimum support)
L_k^{DB}	ฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลเดิม
L_k^{UD}	ฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลปรับปรุง
C_k^{DB}	แคนดิเดตไอเท็มเซ็ตของฐานข้อมูลเดิม
C_k^{db}	แคนดิเดตไอเท็มเซ็ตของฐานข้อมูลใหม่
C_k^{UD}	แคนดิเดตไอเท็มเซ็ตของฐานข้อมูลปรับปรุง
X.count	ค่าสนับสนุนของไอเท็มเซ็ต เมื่อ X คือไอเท็มเซ็ตใดๆ
Temp_scanDB	แคนดิเดตไอเท็มเซ็ตที่ถูกเก็บไว้เพื่อนำไปสแกนฐานข้อมูลเดิม

3.3 ขั้นตอนการทำงานของ BFUP

อัลกอริทึม BFUP ประกอบด้วยกระบวนการทำงานหลัก 2 กระบวนการ ได้แก่ การประมวลผลฐานข้อมูลใหม่ (Incremental updating phase) และการสแกนฐานข้อมูลเดิมอีกครั้ง (Re-scanning Original Database Phase)

3.3.1 ขั้นตอนการประมวลผลฐานข้อมูลใหม่ (Incremental Updating Phase)

ขั้นตอนนี้ จะเป็นการประมวลผลฐานข้อมูลใหม่ เพื่อให้ได้ ฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลปรับปรุง (L_k^{UD}) แคนดิเดต 1-ไอเท็มเซ็ตของฐานข้อมูลปรับปรุง (C_1^{UD}) และแคนดิเดตไอเท็มเซตที่จะถูกนำไปสแกนในฐานข้อมูลเดิม (Temp_scanDB) โดยขั้นตอนนี้จะประกอบด้วยการทำงาน 2 กระบวนการ ได้แก่ การประมวลผลรอบที่ k=1 และการประมวผลรอบที่ k มีค่าตั้งแต่ 2 ขึ้นไป

ในเบื้องต้น ก่อนที่อัลกอริทึม BFUP จะประมวลผลในส่วนของการประมวลผลฐานข้อมูลใหม่ จะต้องมีการประมวลฐานข้อมูลเดิมเพื่อให้ได้ซึ่งผลลัพธ์ที่จำเป็นต้องใช้ ได้แก่ ฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลเดิม (L_k^{DB}) แคนดิเดต 1-ไอเท็มเซ็ตของฐานข้อมูลเดิม (C_1^{DB}) จากนั้นจึงจะเริ่มเข้าสู่การทำงานของอัลกอริทึม BFUP ดังนี้

การทำงานรอบที่ k=1

- สแกนฐานข้อมูลใหม่ (db) เพื่อหาค่าสนับสนุนของ 1-ไอเท็มเซ็ตทุกตัว (C_1^{db})ในฐานข้อมูลใหม่
- สำหรับ C_1^{db}ที่เป็นสมาชิกของ C_1^{DB}หรือเป็นสมาชิกของ L_1^{DB} ให้ทำการปรับปรุงค่าสนับสนุนของไอเท็มเซ็ต
- นำค่าสนับสนุนที่ปรับปรุงแล้วมาตรวจสอบกับค่าสนับสนุนขั้นต่อ (minimum support) หรือ s โดย ถ้าไอเท็มเซ็ตใดๆ มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับบสนุนขั้นต่ำ ไอเท็มเซ็ตนั้นๆ จะถูกนำไปเก็บในตัวแปร L_k^{UD} แต่หากไอเท็มเซ็ตใดๆ ที่มีค่าสนับสนุนน้อยกว่าค่าสนับสนุนขั้นต่ำ ไอเท็มเซ็ตนั้นจะถูกนำปเก็บในตัวแปร C_1^{UD} เพื่อนำไปใช้ในการประมวลผลในครั้งต่อไปเมื่อมีการเพิ่มข้อมูลชุดใหม่เข้ามา
- สำหรับ C_1^{db}ที่ไม่ได้เป็นสมาชิกของ C_1^{DB} และไม่ได้เป็นสมาชิกของ L_1^{DB} ไอเท็มเซ็ตดังกล่าวจะถูกคำนวณ ด้วยการนำเอาค่าสนับสนุนของไอเท็มเซ็ตนั้นๆ บวกด้วยจำนวนทรานแซคชั่นในฐานข้อมูลเดิม แล้วลบออกด้วย 1 นั่นคือ X.count+|DB|-1 จากนั้น นำผลลัพธ์ที่ได้ไปเปรียบเทียบกับค่า

สนับสนุนขั้นต่ำของฐานข้อมูลปรับปรุง (s*|UD|) ซึ่งหากผลลัพธ์ที่ได้มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนของฐานข้อมูลปรับปรุง ไอเท็มเซ็ตนั้นๆ จะถูกนำไปเก็บไว้ในตัวแปร Temp_scanDB เพื่อรอการนำไปสแกนในฐานข้อมูลเดิมในรอบสุดท้าย

การทำงานรอบที่ k≥ 2

การประมวลผลในรอบนี้ จะเป็นการประมวลซ้ำกันไปจำนวน k รอบ จนกว่าจะไม่สามารถเชื่อมไอเท็มเซ็ต (join) ได้อีก โดยขั้นตอนการทำงานในแต่ละรอบเป็นดังนี้

- สร้างแคนดิเดตไอเท็มเซ็ตของฐานข้อมูลใหม่ (C_k^{db}) ด้วยการเชื่อม (join) ไอเท็มเซ็ตโดยใช้หลักการ Apriori_gen() แต่จะแตกต่างจากการเชื่อมไอเท็มเซ็ตของอัลกอริทึม Apriori ตรงที่ BFUP จะทำการเชื่อมไอเท็มระหว่าง (L_{k-1}^{UD} ∪Temp_scanDB$_{k-1}$)* (L_{k-1}^{UD} ∪Temp_scanDB$_{k-1}$)
- สแกนฐานข้อมูลใหม่ (db) เพื่อหาค่าสนับสนุนของ C_k
- สำหรับ C_k^{db}ที่เป็นสมาชิกของ L_k^{DB} ให้ทำการปรับปรุงค่าสนับสนุนของไอเท็มเซ็ต
- นำค่าสนับสนุนที่ปรับปรุงแล้วมาตรวจสอบกับค่าสนับสนุนขั้นต่อ (minimum support) หรือ s โดย ถ้าไอเท็มเซ็ตใดๆ มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับบสนุนขั้นต่ำ ไอเท็มเซ็ตนั้นๆ จะถูกนำไปเก็บในตัวแปร L_k^{UD}
- สำหรับ C_k^{db}ที่ไม่ได้เป็นสมาชิกของ L_k^{DB} ไอเท็มเซ็ตดังกล่าวจะถูกคำนวณ ด้วยการนำเอาค่าสนับสนุนของไอเท็มเซ็ตนั้นๆ บวกด้วยจำนวนทรานแซคชั่นในฐานข้อมูลเดิม แล้วลบออกด้วย 1 นั่นคือ X.count+|DB|-1 จากนั้น นำผลลัพธ์ที่ได้ไปเปรียบเทียบกับค่าสนับสนุนขั้นต่ำของฐานข้อมูลปรับปรุง (s*|UD|) ซึ่งหากผลลัพธ์ที่ได้มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนของฐานข้อมูลปรับปรุง ไอเท็มเซ็ตนั้นๆ จะถูกนำไปเก็บไว้ในตัวแปร Temp_scanDB เพื่อรอการนำไปสแกนในฐานข้อมูลเดิมในรอบสุดท้าย

ผลลัพธ์ที่ได้จากการประมวลผลฐานข้อมูลใหม่นี้ แสดงดังภาพที่ 3.1 จะประกอบด้วยฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลปรับปรุง (L_k^{UD}) แคนดิเดต 1-ไอเท็มเซ็ตของฐานข้อมูลปรับปรุง C_1^{UD} และแคนดิเดตไอเท็มเซ็ตที่ถูกเก็บไว้ในตัวแปร Temp_scanDB ซึ่งผลลัพธ์ของตัวแปร Temp_scanDB จะถูกนำไปใช้ในการประมวลในขั้นตอนที่ 3.3.2

3.3.2 ขั้นตอนการสแกนฐานข้อมูลเดิมซ้ำอีกครั้ง (Re-scanning Original Database Phase)

จากผลลัพธ์ที่ได้จากการประมวลผลในขั้นตอนที่ 3.3.2 คือตัวแปร Temp_scanDB แคนดิเดตไอเท็มเซ็ตทุกตัวที่อยู่ในตัวแปร Temp_scanDB จะถูกนำมาสแกนในฐานข้อมูลเดิมอีกเพียง 1 รอบ เพื่อปรับปรุงค่าสนับสนุนของแคนดิเดตไอเท็มเซ็ตนั้นๆ จากนั้น จึงนำค่าสนับสนุนที่ปรับปรุงแล้วไปตรวจสอบกับค่าสนับสนุนต่ำสุดของฐานข้อมูลปรับปรุง (s*|UD|) ถ้าไอเท็มเซ็ตใดๆ มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับบสนุนขั้นต่ำ ไอเท็มเซ็ตนั้นๆ จะถูกนำไปเก็บในตัวแปร L_k^{UD} แต่ถ้ามีค่าสนับสนุนน้อยกว่าค่าสนับสนุนขั้นต่ำ ไอเท็มเซ็ตนั้นๆ จะถูกตัดทิ้งไป

ขั้นตอนการ ขั้นตอนการสแกนฐานข้อมูลเดิมซ้ำอีกครั้ง (Re-scanning Original Database Phase) จะแสดงดังภาพที่ 3.2

```
Algorithm 1 incremental process()
Input : db,s,L_k^DB,C_1^DB,|DB|
Output: L_k^UD
1   |UD|=|DB|+|db|
2   k = 1
3   scan db for all X
4   for all X∈L_1^DB or X∈C_1^DB
5     update X.count
6     if X.count ≥ s*|UD|
7        X → L_k^UD
8   for all X ∉ L_1^DB or X∉C_1^DB
9     if X.count+|DB|-1 ≥ s*|UD|
10       X → temp_scanDB
11  k=2
12  while (L_k-1^UD ∪ temp_scanDB_k) > 1
13    C_k = L_k-1^UD * temp_scanDB_k
14    // using Apriori_gen()
15    scan db for all C_k
16    for all X∈ C_k do
17       for all X ∈ L_k^DB
18          update X.count
19          if X.count ≥  s*|UD|
20             X → L_k^UD
21       for all X ∉ L_k^DB
22          if X.count+|DB|-1 ≥ s*|UD|
23             X → temp_scanDB
24    k++
25  end loop
26  rescan_original()
27  L_k^UD = L_k^UD ∪ tempL
28  return L_k^UD
```

ภาพที่ 3.1 ขั้นตอนการประมวลผลฐานข้อมูลใหม่ (Incremental Updating Phase)

```
Algorithm 2 rescan_original()
Input DB, temp_scanDB,s,|UD|
Output tempL
1   if temp_scanDB ≠ ∅
2      scan DB for all X ∈ temp_scanDB
3      update X.count
4      if X.count ≥ s*|UD|
5         X → tempL
6   endif
7   return tempL
```

ภาพที่ 3.2 ขั้นตอนการสแกนฐานข้อมูลเดิมซ้ำอีกครั้ง (Re-scanning Original Database Phase)

บทที่ 4

การทดลองและผลการทดลอง
(Experimental Result)

เพื่อแสดงให้เห็นถึงประสิทธิภาพการทำงานของอัลกอริทึม BFUP ซึ่งเป็นอัลกอริทึมที่ปรับปรุงประสิทธิภาพการทำงานของอัลกอริทึม FUP ที่ช่วยในการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย ในบทนี้จะกล่าวถึงวัตถุประสงค์ของการทดลอง วิธีการทดลอง และผลการทดลอง ของอัลกอริทึม BFUP

4.1 วัตถุประสงค์ของการทดลอง

เพื่อแสดงให้เห็นถึงประสิทธิภาพการทำงานของอัลกอริทึม BFUP สำหรับการเพิ่มขยายการค้นหากฎความสัมพันธ์ โดยเป็นการปรับปรุงประสิทธิภาพอัลกอริทึม FUP ในหัวข้อนี้จะกล่าวถึงวัตถุประสงค์ของการทดลอง ซึ่งประกอบด้วย 2 วัตถุประสงค์หลัก ดังนี้

1. เพื่อทดสอบความถูกต้องของผลลัพธ์ที่ได้จากการเพิ่มฐานข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม อัลกอริทึม BFUP แม้ว่าจะเป็นอัลกอริทึมที่พัฒนามาเพื่อปรับปรุงประสิทธิภาพของอัลกอริทึม FUP อย่างไรก็ดี การทำงานของทั้งอัลกอริทึม BFUP และ FUP ล้วนมีฐานการทำงานมาจากอัลกอริทึม Apriori ดังนั้นผู้วิจัยจะออกแบบการทดลองเพื่อทดสอบความถูกต้องของผลลัพธ์โดยเปรียบเทียบกับอัลกอริทึม Apriori เป็นหลัก

2. เพื่อทดสอบประสิทธิภาพในการทำงานของอัลกอริทึมในการเพิ่มฐานข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม การทดสอบประสิทธิภาพของอัลกอริทึมในงานวิจัยนี้ จะเป็นการทดสอบเพื่อวัดประสิทธิภาพการเพิ่มขยายการค้นหากฎความสัมพันธ์โดยวัดจากเวลาที่ใช้ในการประมวลผล (Execution Time) โดยเปรียบเทียบเวลาที่ใช้ในการประมวลผลระหว่างอัลกอริทึม BFUP อัลกอริทึม FUP และอัลกอริทึม Apriori

4.2 วิธีการทดลอง

การทดลองอัลกอริทึมเพื่อค้นหากฎความสัมพันธ์แบบเพิ่มขยาย เมื่อมีการเพิ่มฐานข้อมูลหรือชุดข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม ซึ่งอาจทำให้ฟรีเคว้นท์ไอเท็มเซ็ตที่ได้จากการค้นหาจากฐานข้อมูลเดิมมีการเปลี่ยนแปลงไป ในการทดลองสำหรับงานวิจัยนี้ จะเป็นการทดลองโดยการเพิ่มข้อมูลเข้าไป 1 ชุดข้อมูล จำนวน 1,000 ทรานแซคชั่น โดยเพิ่มเข้าไปในฐานข้อมูลเดิมจำนวน 2 ฐานข้อมูล ที่จำนวนทรานแซคชั่น 50,000 และ 100,000 ทรานแซคชั่น ตามลำดับ ด้วยค่าสนับสนุนขั้นต่ำ (minimum support) ในระดับที่ต่างกัน โดยฐานข้อมูล 50,000 ทรานแซคชั่น จะทดสอบกับค่าสนับสนุนขั้นต่ำ 0.2% 0.3% และ 0.4% ส่วนฐานข้อมูล 100,000 ทรานแซคชั่น จะทดสอบกับค่าสนับสนุนขั้นต่ำ 0.1% 0.3% และ 0.5%

สำหรับชุดข้อมูลที่นำมาทดลอง เป็นชุดข้อมูลสังเคราะห์ (Synthesis Dataset) ซึ่งเป็นชุดข้อมูลที่นำเสนอโดย Agrawal และคณะ [1] ซึ่งได้เสนอวิธีการสร้างชุดข้อมูลสังเคราะห์เพื่อใช้ในการประเมินประสิทธิภาพของอัลกอริทึม โดยอาศัยหลักการทางสถิติมาใช้ในการสร้างชุดข้อมูล สำหรับการทดลองในงานวิจัยนี้ ได้เลือกใช้ชุดข้อมูล 2 ชุดข้อมูลคือ ชุดข้อมูล I10T4D50K สำหรับฐานข้อมูลเดิม 50,000 ทรานแซคชั่น และชุดข้อมูล ชุดข้อมูล I10T4D100K สำหรับฐานข้อมูลเดิม 100,000 ทรานแซคชั่น

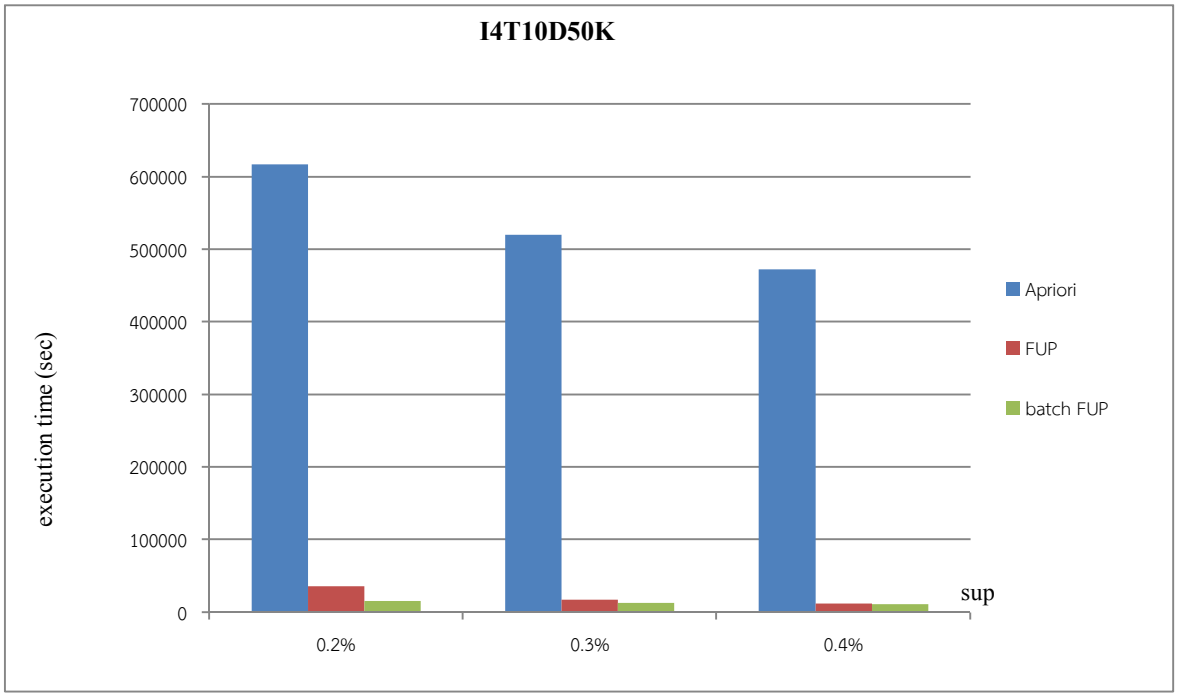
4.3 ผลการทดลอง

เมื่อนำข้อมูลทั้ง 2 ชุดดังกล่าวข้างต้นมาทำการทดลองด้วยโปรแกรม MATLAB 7.6 ผลการทดลองเป็นดังนี้

ผลการทดลองชุดข้อมูลที่ 1 คือ I10T4D50K สำหรับฐานข้อมูลเดิม 50,000 ทรานแซคชั่น และมีการเพิ่มข้อมูลเข้ามาจำนวน 1,000 ทรานแซคชั่น พร้อมด้วยการทดสอบกับค่าสนับสนุนต่ำสุดจำนวน 3 ค่า คือ 0.2% 0.3% และ 0.4% และทดสอบประสิทธิภาพของอัลกอริทึมโดยเปรียบเทียบกับอัลกอริทึม FUP และ อัลกอริทึม Apriori ผลการทดลองแสดงตามตารางที่ 4.1 และภาพที่ 4.1

ตารางที่ 4.1 ผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผลข้อมูลชุด I10T4D50K ระหว่างอัลกอริทึม Apriori FUP และ BFUP

อัลกอริทึม	**เวลาที่ใช้ในการประมวลผลเฉลี่ย (วินาที)**		
	ค่าสนับสนุนขั้นต่ำ (min_sup)		
	0.2%	0.3%	0.4%
Apriori	617,259.2492	520,470.1767	472,622.0955
FUP	36,012.0041	17,695.8894	12,363.9417
BFUP	15,402.0908	12,906.0013	10,866.7162
จำนวนฟรีเคว้นท์ไอเท็มเซ็ต	5,307	1,995	1,051
ขนาดสูงสุดของ k-itemset	L_{10}	L_7	L_3



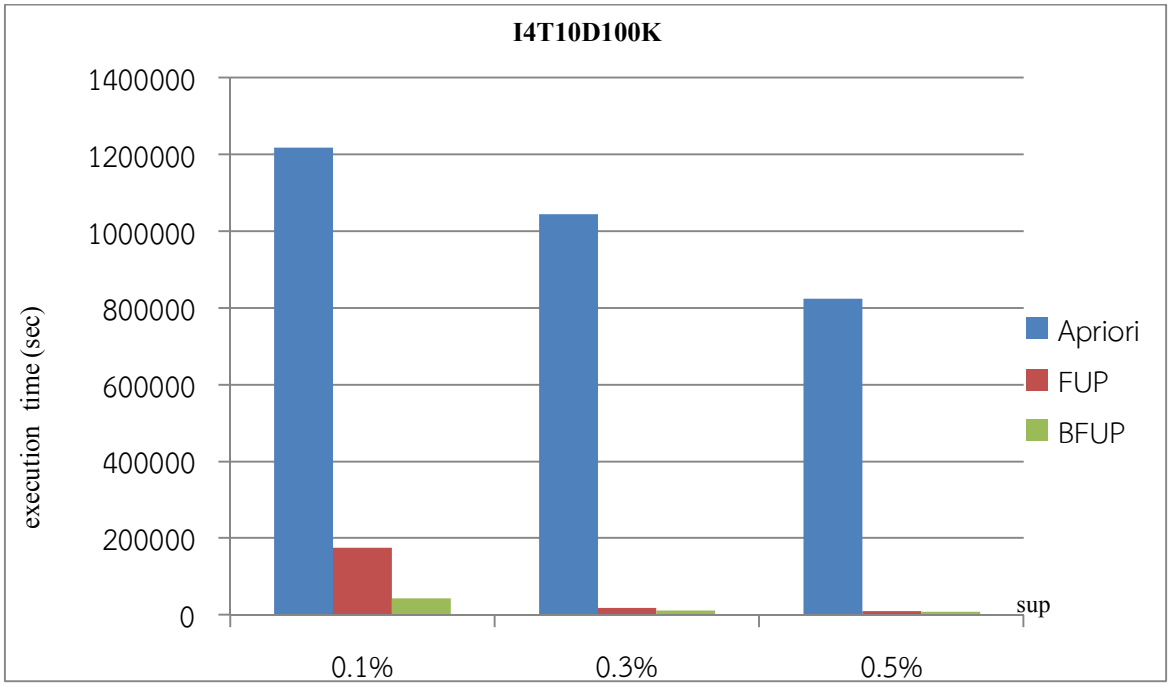
ภาพที่ 4.1 แผนภูมิแท่งเปรียบเทียบเวลาที่ใช้ในการประมวลผลของชุดข้อมูล I10T4D50K

จากตารางที่ 4.1 และภาพที่ 4.1 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผลระหว่างอัลกอริทึม Apriori FUP และ BFUP จะเห็นได้ว่า เมื่อมีการเพิ่มข้อมูลจำนวน 1,000 ทรานแซคชั่น เข้าไป

ในฐานข้อมูลเดิมจำนวน 50,000 ทรานแซคชั่น เมื่อพิจารณาถึงความถูกต้องของความถูกต้องในการประมวลผลโดยเปรียบเทียบกับอัลกอริทึม Apriori ผลการทดสอบพบว่า BFUP ให้ผลลัพธ์ที่ถูกต้อง คือมีจำนวน ฟรีเคว้นท์ไอเท็มเซ็ตเท่ากับจำนวนฟรีเคว้นท์ไอเท็มเซตที่ประมวลผลได้จากอัลกอริทึม Apriori และ FUP นอกจากนี้ เมื่อพิจารณาถึงเวลาที่ใช้ในการประมวลผล จะพบว่า ใช้ระยะเวลาในการประมวลผลน้อยกว่าอัลกอริทึม Apriori อย่างเห็นได้ชัด และใช้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม FUP ซึ่งถือว่าอัลกอริทึม BFUP สามารถปรับปรุง FUP ให้สามารถใช้เวลาในการประมวลน้อยลงอย่างมีประสิทธิภาพโดยเฉพาะอย่างยิ่ง เมื่อทดสอบกับค่าสนับสนุนขั้นต่ำจำนวน 3 ค่า คือ 0.2% 0.3% และ 0.4% จะเห็นได้ว่า ยิ่งกำหนดค่าสนับสนุนขั้นต่อน้อยลงเท่าใด จะยิ่งเห็นความแตกต่างของเวลาที่ใช้ในการประมวลของทั้ง 3 อัลกอริทึมมากยิ่งขึ้น นั่นคือ ยิ่งกำหนดค่าสนับสนุนขั้นต่ำน้อยมากเท่าใด อัลกอริทึม BFUP จะยิ่งใช้เวลาน้อยกว่าอัลกอริทึม Apriori และ FUP มากขึ้นเท่านั้น

ตารางที่ 4.2 ผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผลข้อมูลชุด I10T4D100K ระหว่างอัลกอริทึม Apriori FUP และ BFUP

อัลกอริทึม	**เวลาที่ใช้ในการประมวลผลเฉลี่ย (วินาที)**		
	ค่าสนับสนุนขั้นต่ำ (min_sup)		
	0.1%	**0.3%**	**0.5%**
Apriori	1,218,728.9102	1,044,195.4851	823,997.0138
FUP	175,890.2901	19,645.1002	9,771.1612
BFUP	43,580.3014	11,678.5801	9,105.8170
จำนวนฟรีเคว้นท์ไอเท็มเซ็ต	17,127	1,991	862
ขนาดสูงสุดของ k-itemset	L_{10}	L_7	L_2



ภาพที่ 4.2 แผนภูมิแท่งเปรียบเทียบเวลาที่ใช้ในการประมวลผลของชุดข้อมูล I10T4D100K

จากตารางที่ 4.2 และภาพที่ 4.2 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผลระหว่างอัลกอริทึม Apriori FUP และ BFUP จะเห็นได้ว่า เมื่อมีการเพิ่มข้อมูลจำนวน 1,000 ทรานแซคชั่น เข้าไปในฐานข้อมูลเดิมจำนวน 100,000 ทรานแซคชั่น เมื่อพิจารณาถึงความถูกต้องของความถูกต้องในการประมวลผลโดยเปรียบเทียบกับอัลกอริทึม Apriori ผลการทดสอบพบว่า BFUP ให้ผลลัพธ์ที่ถูกต้อง คือมีจำนวน ฟรีเคว้นท์ไอเท็มเซ็ตเท่ากับจำนวนฟรีเคว้นท์ไอเท็มเซตที่ประมวลผลได้จากอัลกอริทึม Apriori และ FUP นอกจากนี้ เมื่อพิจารณาถึงเวลาที่ใช้ในการประมวลผล จะพบว่า ใช้ระยะเวลาในการประมวลผลน้อยกว่าอัลกอริทึม Apriori และ FUP อย่างเห็นได้ชัด เมื่อพิจารณาถึงผลการทดสอบกับค่าสนับสนุนขั้นต่ำจำนวน 3 ค่า คือ 0.1% 0.3% และ 0.5% จะเห็นได้ว่า ยิ่งกำหนดค่าสนับสนุนขั้นต่อน้อยลงเท่าใด จะยิ่งเห็นความแตกต่างของเวลาที่ใช้ในการประมวลของทั้ง 3 อัลกอริทึมมากยิ่งขึ้น นั่นคือ ยิ่งกำหนดค่าสนับสนุนขั้นต่ำน้อยมากเท่าใด อัลกอริทึม BFUP จะยิ่งใช้เวลาน้อยกว่าอัลกอริทึม Apriori และ FUP มากขึ้นเท่านั้น ซึ่งผลลัพธ์ที่ได้ให้ผลเช่นเดียวกับการทดสอบกับข้อมูลชุดแรก (I10T4D50K)

4.4 สรุปผลการทดลอง

จากการทดลองข้อมูล 2 ชุด คือ ชุดข้อมูล I10T4D50K สำหรับฐานข้อมูลเดิม 50,000 ทรานแซคชั่น และชุดข้อมูล ชุดข้อมูล I10T4D100K สำหรับฐานข้อมูลเดิม 100,000 ทรานแซคชั่น และมีการเพิ่มจำนวนชุดข้อมูลใหม่เข้าไปจำนวน 1,000 ทรานแซคชั่นเข้าไปในฐานข้อมูลเดิมทั้งสองชุด ในการทดลองจะแบ่งการวัดผลการทดลองออกเป็น 2 ประเด็น คือประเด็นของความถูกต้องในการประมวลผลและการวัดประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล สามารถสรุปผลการทดลองทั้ง 2 ประเด็นได้ดังนี้

1. ด้านความถูกต้อง ในการทดสอบความถูกต้องของอัลกอริทึม BFUP ผู้วิจัยได้ออกแบบการทดสอบความถูกต้องโดยการเปรียบเทียบกับผลลัพธ์ที่ได้จากการประมวลผลอัลกอริทึม Apriori และ FUP ซึ่งผลการทดสอบ พบว่า อัลกอริทึม BFUP ให้ผลลัพธ์ในการประมวลผลที่ถูกต้อง นั่นคือ มีจำนวนฟรีเคว้นท์ไอเท็มเซ็ตที่ได้จากการประมาลผลและขนาดสูงสุดของ k-itemset เท่ากับอัลกอริทึม Apriori และ FUP

2. ด้านประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล ในการทดสอบ ผู้วิจัยได้ทดสอบประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล โดยการเปรียบเทียบกับ 2 อัลกอริทึมคือ Apriori และ FUP ด้วยค่าสนับสนุนต่ำสุดที่แตกต่างกันจำนวน 3 ค่า คือ 0.2% 0.3% และ 0.4% สำหรับข้อมูลชุด 50.000 ทรานแซคชั่น และ 0.1% 0.3% และ 0.5% สำหรับข้อมูลชุด 100,000 ทรานแซคชั่น ผลการทดลองพบว่า อัลกอริทึม BFUP ใช้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม Apriori และ FUP โดยเฉพาะอย่างยิ่ง เมื่อมีการกำหนดค่าสนับสนุนขั้นต่อน้อยลงเท่าใด จะยิ่งเห็นความแตกต่างของเวลาที่ใช้ในการประมวลของทั้ง 3 อัลกอริทึมมากยิ่งขึ้น นั่นคือ ยิ่งกำหนดค่าสนับสนุนขั้นต่ำน้อยมากเท่าใด อัลกอริทึม BFUP จะยิ่งใช้เวลาน้อยกว่าอัลกอริทึม Apriori และ FUP มากขึ้นเท่านั้น นั่นเป็นเพราะว่า อัลกอริทึม Apriori และอัลกอริทึม FUP จำเป็นต้องมีการสแกนแคนดิเดตไอเท็มเซ็ตในฐานข้อมูลเดิมทุกๆ รอบ จำนวน k รอบ ในขณะที่อัลกอริทึม BFUP จะมีการสแกนฐานข้อมูลเดิมเพียงครั้งเดียว จึงทำให้ผลลัพธ์ทางด้านเวลาเป็นไปดังผลการทดลองที่แสดงในตารางที่ 4.1 และ 4.2 ดังนั้น จากผลการทดลองจะเห็นได้ว่า อัลกอริทึม BFUP สามารถช่วยปรับปรุงประสิทธิภาพของอัลกอริทึม FUP ได้อย่างมีประสิทธิผล

บทที่ 5

สรุปและข้อเสนอแนะ
(Conclusion)

5.1 สรุปผลการวิจัย

การทำเหมืองข้อมูล (Data Mining) ซึ่งบางครั้งจะถูกกล่าวถึงว่า การค้นหาองค์ความรู้ในฐานข้อมูล (Knowledge Discovery in Database: KDD) เป็นกระบวนการที่ถูกนำมาใช้เพื่อค้นหารูปแบบ (pattern) หรือความสัมพันธ์ที่ไม่เคยค้นพบมาก่อนที่ถูกซ่อนอยู่ในข้อมูลจำนวนมหาศาลโดยอัตโนมัติ รูปแบบข้อมูลที่ค้นพบดังกล่าวจะช่วยให้องค์กรสามารถสร้างความได้เปรียบในการแข่งขันได้ อาทิ การนำเทคนิคการทำเหมืองข้อมูลเข้ามาช่วยในการวิเคราะห์กลุ่มลูกค้าเงินกู้ชั้นดีของธนาคาร การจำแนกกลุ่มลูกค้าที่มีความสามารถในการซื้อสินค้าราคาสูงจำพวกบ้านหรือรถยนต์ และการวิเคราะห์รูปแบบการซื้อสินค้าของลูกค้าเป็นต้น

การค้นหากฎความสัมพันธ์ เป็นเทคนิคที่สำคัญของกระบวนการทำเหมืองข้อมูล (Data Mining) เพื่อค้นหาความสัมพันธ์ระหว่างข้อมูลในฐานข้อมูล และสกัดรูปแบบข้อมูลที่น่าสนใจให้ออกมาอยู่ในรูปแบบของกฎความสัมพันธ์ if X then Y โดยมีหลักการทำงานหลัก 2 ขั้นตอนได้แก่ 1) การค้นหาไอเท็มเซ็ตที่เรียกว่าฟรีเคว้นท์ไอเท็มเซ็ต ซึ่งเป็นไอเท็มเซตที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ (minimum support) ที่ผู้ใช้กำหนด และ 2) การนำฟรีเคว้นท์ไอเท็มเซ็ตที่หาได้จากข้อแรกมาสร้างกฎความสัมพันธ์ ซึ่งกฎความสัมพันธ์ที่น่าสนใจ จะเป็นกฎความสัมพันธ์ที่มีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ (minimum confidence) ที่ผู้ใช้กำหนด

โดยทั่วไปแล้ว อัลกอริทึมที่ได้รับการยอมรับและเป็นที่นิยมในการนำมาค้นหากฎความสัมพันธ์คืออัลกอริทึม Apriori [3] ที่ถูกเสนอโดย Agrawal อย่างไรก็ดี เมื่อมีการเพิ่มชุดข้อมูลใหม่เข้าไปในฐานข้อมูลเดิม การค้นหากฎความสัมพันธ์ด้วยวิธีการของ Apriori จะต้องทำประมวลผลข้อมูลทั้งหมดที่อยู่ฐานข้อมูล นั่นคือ การหาแคนดิเดตไอเท็มเซตใหม่ และสแกนหาสนับสนุนของแคนดิเดตไอเท็มเซ็ตใหม่ในทุกๆ รอบ ซึ่งทำให้เวลาที่ใช้ในการประมวลผลถูกใช้มากเกินความจำเป็น และไม่เกิดประสิทธิภาพในการทำงาน

ด้วยข้อด้อยดังกล่าวของ Apriori จึงได้มีผู้นำเสนอการปรับปรุงกฎความสัมพันธ์เมื่อมีการเพิ่มข้อมูลชุดใหม่เข้ามาในฐานข้อมูลเดิม Cheung และคณะ [4] ได้ เสนออัลกอริทึม FUP (Fast UPdate algorithm) ซึ่งเป็นอัลกอริทึมสำหรับการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย เมื่อมีการเพิ่มข้อมูลใหม่เข้ามา โดยอาศัยองค์ความรู้เดิมจากการไมนิ่งในฐานข้อมูลเดิมมาช่วยเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ นั่นคือ FUP จะใช้ฟรีเคว้นไอเท็มเซ็ตที่ได้จากการประมวลผลในฐานข้อมูลเดิม มาช่วยลดจำนวนแคนดิเดตไอเท็มเซ็ตที่จะต้องถูกนำไปสแกนในฐานข้อมูลเดิม ด้วยวิธีการประมวลผลดังกล่าวจึงทำให้ FUP ใช้เวลาในการประมวลผลน้อยกว่า Apriori

อย่างไรก็ตาม ในแต่ละรอบ k เมื่อมีการค้นพบแคนดิเดตไอเท็มเซตที่ไม่ได้เป็นสมาชิกของฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลเดิม แคนดิเดตไอเท็มเซ็ตนั้นๆ จะถูกนำไปสแกนในฐานข้อมูลเดิมทุกรอบที่ k นั่นแปลว่า หากขนาดของฟรีเคว้นไอเท็มเซตในฐานข้อมูลปรับปรุงมีค่าเท่ากับ 5 (k=5) ย่อมหมายความว่า FUP จะต้องนำแคนดิเดตไอเท็มเซ็ตไปสแกนในฐานข้อมูลเดิมรวมจำนวนทั้งสิ้น 5 รอบ เช่นเดียวกับ

Apriori ต่างกันตรงที่ จำนวนแคนดิเดตไอเท็มเซ็ตที่ถูกนำไปสแกนของ FUP จะมีจำนวนน้อยกว่า Apriori เท่านั้น

เพื่อลดจำนวนการสแกนฐานข้อมูลเดิมให้เหลือจำนวนรอบการสแกนที่น้อยที่สุด ผู้วิจัยจึงได้ศึกษาเพื่อปรับปรุงอัลกอริทึม FUP ให้ทำงานโดยเหลือจำนวนรอบของการประมวลในฐานข้อมูลเดิมเพียงรอบเดียว ซึ่งผู้วิจัยได้อาศัยแนวคิดจากอัลกอริทึม Negative Border ที่มีการจัดเก็บไอเท็มเซ็ตที่ไม่ใช่ฟรีเคว้นท์ไอเท็มเซ็ต เพื่อช่วยลดการสแกนฐานข้อมูลเดิม ทั้งนี้อัลกอริทึมที่ปรับปรุงประสิทธิภาพการทำงานของ FUP นี้ ผู้วิจัยจะเรียกว่าอัลกอริทึม BFUP (Batch Fast UPdate Algorithm)

ในการทดลองผล ผู้วิจัยทดลองผลกับข้อมูลสังเคราะห์จำนวน 2 ชุด คือ ชุดข้อมูล I10T4D50K สำหรับฐานข้อมูลเดิม 50,000 ทรานแซคชั่น และชุดข้อมูล ชุดข้อมูล I10T4D100K สำหรับฐานข้อมูลเดิม 100,000 ทรานแซคชั่น และมีการเพิ่มจำนวนชุดข้อมูลใหม่เข้าไปจำนวน 1,000 ทรานแซคชั่นเข้าไปในฐานข้อมูลเดิมทั้งสองชุด ด้วยค่าสนับสนุนต่ำสุดที่แตกต่างกัน 3 ค่า คือ คือ 0.2% 0.3% และ 0.4% สำหรับข้อมูลชุด 50.000 ทรานแซคชั่น และ 0.1% 0.3% และ 0.5% สำหรับข้อมูลชุด 100,000 ทรานแซคชั่น

ผลการทดลองแบ่งออกเป็น 2 ประเด็น คือประเด็นของความถูกต้องในการประมวลผล และการวัดประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล สามารถสรุปผลการทดลองทั้ง 2 ประเด็นได้ดังนี้

1. ด้านความถูกต้อง ในการทดสอบความถูกต้องของอัลกอริทึม BFUP ผู้วิจัยได้ออกแบบการทดสอบความถูกต้องโดยการเปรียบเทียบกับผลลัพธ์ที่ได้จากการประมวลผลอัลกอริทึม Apriori และ FUP ซึ่งผลการทดสอบ พบว่า อัลกอริทึม BFUP ให้ผลลัพธ์ในการประมวลผลที่ถูกต้อง

2. ด้านประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล ในการทดสอบ ผู้วิจัยได้ทดสอบประสิทธิภาพด้านเวลาที่ใช้ในการประมวลผล โดยการเปรียบเทียบกับ 2 อัลกอริทึมคือ Apriori และ FUP ผลการทดลอง พบว่า อัลกอริทึม BFUP ใช้เวลาในการประมวลผลน้อยกว่าอัลกอริทึม Apriori และ FUP โดยเฉพาะอย่างยิ่ง เมื่อมีการกำหนดค่าสนับสนุนขั้นต่อน้อยลงเท่าใด จะยิ่งเห็นความแตกต่างของเวลาที่ใช้ในการประมวลของทั้ง 3 อัลกอริทึมมากยิ่งขึ้น นั่นคือ ยิ่งกำหนดค่าสนับสนุนขั้นต่ำน้อยมากเท่าใด อัลกอริทึม BFUP จะยิ่งใช้เวลาน้อยกว่าอัลกอริทึม Apriori และ FUP มากขึ้นเท่านั้น นั่นเป็นเพราะว่าอัลกอริทึม Apriori และ อัลกอริทึม FUP จำเป็นต้องมีการสแกนแคนดิเดตไอเท็มเซ็ตในฐานข้อมูลเดิมทุกๆ รอบ จำนวน k รอบ ในขณะที่อัลกอริทึม BFUP จะมีการสแกนฐานข้อมูลเดิมเพียงครั้งเดียว จึงทำให้ผลลัพธ์ทางด้านเวลาเป็นไปดังผลการทดลองที่แสดงในตารางที่ 4.1 และ 4.2 ดังนั้น จากผลการทดลองจะเห็นได้ว่า อัลกอริทึม BFUP สามารถช่วยปรับปรุงประสิทธิภาพของอัลกอริทึม FUP ได้อย่างมีประสิทธิผล

5.2 ข้อเสนอแนะ

1. แนวคิดในของอัลกอริทึม BFUP เป็นแนวคิดที่พยายามปรับปรุงประสิทธิภาพของอัลกอริทึมการค้นหากฎความสัมพันธ์แบบเพิ่มขยาย โดยการพยายามลดจำนวนรอบที่จำเป็นต้องสแกนฐานข้อมูลเดิมโดยการเก็บไอเท็มเซ็ตที่มีโอกาสเป็นฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลปรับปรุงในแต่ละรอบไว้จนถึงรอบสุดท้าย จากนั้นจึงนำไปสแกนฐานข้อมูลเดิม เพื่อปรับปรุงค่าสนับสนุนของไอเท็มเซ็ต แล้วทดสอบการเป็นฟรีเคว้นท์ไอเท็มเซ็ตของฐานข้อมูลปรับปรุงอีกครั้ง อย่างไรก็ดี อัลกอริทึม BFUP นี้ เป็นการดำเนินการกับข้อมูลที่มีการเพิ่มเข้ามาในฐานข้อมูลเท่านั้น ดังนั้นงานวิจัยที่สามารถพัฒนาต่อยอดได้ควรเป็นงานวิจัยที่ปรับปรุงประสิทธิภาพของอัลกอริทึม BFUP โดยให้สามารถประมวลผลได้กับชุดข้อมูลเดิมที่มีการถูกลบทิ้งไป หรือมีการปรับปรุงแก้ไขข้อมูล

2. งานวิจัยฉบับนี้ เป็นการวิจัยภายใต้สมมติฐานที่ว่าค่าสนับสนุนขั้นต่ำที่ถูกกำหนดโดยผู้ใช้จะไม่มีการเปลี่ยนแปลง ดังนั้น งานวิจัยในครั้งต่อไป ควรจะวิจัยเพื่อศึกษาและพัฒนาอัลกอริทึม BFUP ให้สามารถประมวลผลกับค่าสนับสนุนขั้นต่ำที่มีการเปลี่ยนแปลงได้

บรรณานุกรม

[1] Agrawal R, Imielinski T, Swami A (1993), A mining association rules between sets of items in large database, In Proceeding of the ACM SIGMOD Int'l Conf. on Management of Data (ACM SIGMOD'93), Washington, USA, May 1993, pp.207-216.

[2] Agrawal R, Srikant R (1994), Fast algorithm for mining association rules, In Proc. 20th Int. Conf. Very Large DataBases (VLDB'94), Santiago, Chile, September 12-15, 1994, pp.487-499.

[3] Tsai Paulry S.M., Lee Chih-Chong, Chen Abree L.P. (2005), An Efficient Approach for Incremental Association Rule Mining, Proceedings of the third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining, Lecture Notes In Computer Science, Vol. 1574 archieve, 1999.

[4] Cheung D.W., Han J, Ng V.T., Wong C.Y., Maintenance of Discovered Association rules in Large Databases: An incremental updating technique, In 12th IEEE International Conference on Data Engineering, pp 106-114, 1996.

[5] Thomas, S., Bodagala, S., Alsabti, K., and Ranka, S., "An efficient algorithm for th incremental updation of association rules in large databases." In Proceedings of the 3 rd International Conference on Knowledge Discovery and Data Mining." New Port Beach, California, 1997.

ภาคผนวก

PROCEEDINGS OF THE SEVENTEENTH INTERNATIONAL SYMPOSIUM ON ARTIFICIAL LIFE AND ROBOTICS

(AROB 17th '12)

Jan. 19-21, 2012

B-Con Plaza, Beppu, Oita, JAPAN

Editors: Masanori Sugisaka and Hiroshi Tanaka

Publisher: ALife Robotics Co., Ltd.
Publication Date: Jan. 10, 2012
ISBN 978-4-9902880-6-8

Proceedings of the Seventeenth International Symposium on

ARTIFICIAL LIFE AND ROBOTICS

(AROB 17th '12)

January 19- 21, 2012
B-Con Plaza, Beppu, Oita, Japan

Editors: Masanori Sugisaka and Hiroshi Tanaka

The Seventeenth International Symposium on Artificial Life and Robotics 2012 (AROB 17th '12),
B-Con Plaza, Beppu, Oita, Japan, January 19-21, 2012

Batch fast update algorithm for incremental association rule discovery

Araya Ariya[1] , Worapoj Kreesuradej[2]

King Mongkut's Institute of Technology Ladkrabang, Thailand

[1]araya_aa@hotmail.com , [2]worapoj@it.kmitl.ac.th

Abstract: When new transactions are inserted into an original database, the existing rules may be change. An incremental association rule mining is an approach to deal with such problem. This paper proposes an algorithm for mining incremental association rules, called batch fast update (BFUP). The proposed algorithm improves the performance of FUP algorithm by reducing a number of scanning times of an original database. The experimental results show that an execution time of BFUP is much faster than that of FUP.

Keywords: Incremental association rules mining, Association rule mining, Data mining

1. INTRODUCTION

An association rule mining, one of the important tasks in data mining, is a well-known research topic that many researchers propose a large number of algorithms for solving association rule discovering problems. This problem was first presented by Agrawal et al [1] which analyzes the behavior of customer purchasing to help business make a decision. The results show co-occurrence buying items called frequent patterns, which can generate interesting rules. From that research, the problem statement of an association rule mining is defined as follows.

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of literal items. DB is a database which contains transactions. Each transaction T is a set of items where $T \subseteq I$. Given X is an item and $X \subseteq I$. Each transaction contain X if and only if $X \subseteq T$. Let X and Y are an item where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y \neq 0$. Each set of items , itemsets, is called a frequent itemset if and only if its support is greater than or equal to support threshold s%. It calculates from a number of transactions in DB that contain $X \cup Y$. An association rule can be shown in $X \Rightarrow Y$ form. Each frequent itemset can be made the association rule if and only if it is satisfied by confidence threshold c% which calculates from a number of transactions in DB that contain X and also contain Y. Both s% and c% are specified by user.

After an association rule mining was revealed, it motivated many researchers to extend this research area in a lot of issues. An incremental association rule mining is the one of an association rule mining issue which maintains association rules when new transactions are appended to an original database.

One research issue of an incremental association rule mining is reducing running time of the algorithms by minimizing the number of times to scan an original database. This paper also works on this issue. An algorithm for mining incremental association rules, called batch fast update (BFUP), is proposed. This algorithm has only one original database scanning.

The paper is organized as follows. The literatures of an association rule mining and an incremental association rule mining are reviewed in section 2. The problem statement of an incremental mining on association rule in dynamic database and FUP algorithm are detailed in section 3. The proposed algorithm and its experiment are presented in section 4 and 5 respectively. The paper conclusion and future work are briefed in section 6.

2. RELATED WORK

Mining association rules was first proposed by Agrawal et al [1] which finds a correlation between itemsets in a transaction database. The algorithm has 2 steps: finding frequent itemsets (sometimes they are called large itemsets) and generating rules. Subsequent years, Apriori [2], the most popular algorithm, was proposed to discover frequent itemsets.

When a database, called a dynamic database, is inserted new transactions, frequent itemsets can be changed after inserting new transactions into the dynamic database. Therefore, an association rule discovery algorithm for a dynamic database has to maintain frequent itemsets when new transactions are inserted into the dynamic database.

One approach to find the new frequent itemsets is to rerun Apriori algorithm for the whole transactions of the dynamic database. This approach is not efficient because all the computation done initially at finding out the old large itemsets are wasted and all large itemsets have to be computed again from scratch [3].

Cheung et al [3] proposed fast update algorithm, FUP, to solve a rules maintenance problem by using the previous knowledge to find frequent itemsets in updated database. The concept of FUP is re-using frequent itemsets of previous mining to update with frequent itemsets of an incremental database. Although FUP can decrease a number of candidate itemsets for scanning original database, it still needs to scan an original database k times when new frequent itemsets are found. This can degrade the performance of FUP algorithm.

In our observation, the advantage of FUP are re-using frequent itemsets of a previous mining to prune itemsets which cannot be a frequent itemset in updated database and reducing candidate itemsets to scan in an original database.

However, the disadvantage of FUP is the algorithm needs to scan an original database equal to a size of k frequent itemsets, e.g., if maximum size of k frequent itemsets is k=5, FUP needs to scan an original database for 5 times.

From this problem, this paper proposes an incremental association rule mining algorithm to improve the performance of FUP algorithm by reducing a number of scanning times of an original database.

3. INCREMENTAL ASSOCIATION RULES MINING

3.1 Problem statement

In a dynamic database, new transactions are appended to a database; accordingly, the previous valid rules may be invalid. The problem statement for an incremental association rule is defined as follows.

Let DB is an original database. An increment database db is the new transactions which are inserted into DB. Updated database UD is the combining between original database and increment database, i.e, UD = DB $\cup$ db. A number of transactions of an original database, an increment database and an updated database are |DB|, |db| and |UD| = |DB|+|db| respectively.

Before updating activity, L is the frequent itemsets in DB if and only if X.support $\geq$ s×|DB|. After updating activity, L' is the frequent itemsets in updated database if and only if X.support $\geq$ s×|UD|.

According to Tsai et al [4], when news transactions are insert into an original database, an itemset, i.e. X, can be categorized into 4 cases:

Case 1: X is a frequent itemset in both DB and UD

Case 2: X is a frequent itemset in DB and an infrequent itemset in UD

Case 3: X is an infrequent itemset in DB and a frequent itemset in UD

Case 4: X is an infrequent itemset in both DB and UD

Form these cases of itemsets are mentioned above, it is easy to discovered updated frequent itemsets for the itemset of case 1 and 2 because their count in DB and UD are known, therefore, an updating activity is a trivial task. The itemset in case 4 is unimportant because it cannot change an association rule. The most serious case is the 3rd because it needs to rescan an original database for updating its count. Thus, discovering itemsets in case 3 is the most important problem in an incremental association rule mining. In section 3.2, the method to solve that problem is reviewed and section 4, batch fast update algorithm is presented how to improve a performance of FUP.

3.2 FUP algorithm

Apriori [2] is successful for finding frequent itemsets in a database. However, it is unsuitable for mining in dynamic database. Cheung et al [3] have been proposed an incremental algorithm which has a good performance for mining association rules in dynamic database. The concept of FUP is reviewed briefly in this section.

The operation of FUP has 2 phases: 1-iteration and k-iteration where k $\geq$ 2. In the first phase, an increment database is scanned for finding candidate 1-itemsets C_1 and its count. After that loser and winner itemsets are found. Finding loser and winner itemsets, C_1 are divided into 2 types: a member and not a member of previous frequent itemsets in an original database. The first type is updated its count and pruned loser itemsets if its updated count is less than s*|UD|. The second type is scanned to an original database if and only if it is a frequent itemset (winner) in an increment database, i.e., its count is greater than or equal to s*|db|. Both types which satisfy by them threshold can be frequent 1-itemsets in updated database L_1'(winner).

The second phase has 3 steps: filtering out loser itemsets, generating candidate k-itemsets $C_{k\geq 2}$ and finding new frequent itemsets. Firstly, FUP filters out losers from L_k (L_k in DB). Given $Y \in L_k$ and $X \in L_{k-1} - L_{k-1}'$, Y is a loser iff $X \in Y$. For the remaining L_k, they are scanned to an increment database and updated their count. Then they are checked for finding a winner or loser itemset similar to the first phase.

Secondly, C_k is generated by using Apriori-gen. Any C_k is pruned if and only if $Y \in C_k$ where Y is the loser from L_k. This step is a key to reduce a number of C_k before scanning an original database. Finally, new frequent itemsets L_k' are found with the same method as the first phase.

4. BATCH FAST UPDATE ALGORITHM

In this section, an algorithm for mining incremental association rules, batch fast update (BFUP) is presented. The proposed algorithm is assumed that two thresholds, minimum support s% and minimum confidence c%, are static. This algorithm needs only one original database pass and infrequent itemsets are not required. The notation used in this section is defined in Table 1.

Table 1. The notation for Batch fast update algorithm

notation	meaning
DB	original database
db	increment database
UD	updated database
s	minimum support
L_k^{DB}	frequent k-itemset in DB
L_k^{UD}	frequent k-itemset in UD
C_k	candidate k-itemset
|DB|	a number of transactions in DB
|db|	a number of transactions in db
|UD|	a number of transactions in UD
X.count	a support of an itemset
Temp_scanDB	itemsets which are scanned in DB

The algorithm has 2 phases: an increment updating phase and a re-scanning original database phase. The first phase is shown in figure 1. At each iteration, an increment

database is scanned to find candidate itemsets, i.e., C_k, and their support counts. Basically, candidate itemsets are divided into 2 types: a member and not a member of previous frequent itemsets of an original database. Candidate itemsets of the first type becomes updated frequent itemsets, i.e., L_k^{UD}, if and only if their updated support count is greater than or equal to s*|UD|. Candidate itemsets of the second type are kept in temp_scanDB for rescanning in an original database if and only if their count plus |DB|-1 is greater than s*|UD|. On the other hand, Candidate itemsets of the second type are pruned if and only if their count plus |DB|-1 is less than s*|UD|.

For generating candidate itemset C_k, Apriori-gen is applied. Apriori generates C_k with L_{k-1}*L_{k-1}, whereas BFUP generates C_k with L_k^{UD} * $temp\_scanDB_k$.

The second phase of BFUP algorithm is shown in figure 2. After an increment updating phase is ended, all itemsets in temp_scanDB are re-scanned in an original database and updated their support count. Then, all itemsets in temp_scanDB are checked to find updated frequent itemsets. Let X ∈ temp_scanDB, X can be an updated frequent itemset, i.e., L_k^{UD}, if and only if X.count ≥ s*|UD|.

Algorithm 1 An increment updating phase ()

```
Input : db,s,L_k^DB,C_1^DB,|DB|
Output: L_k^UD
1   |UD|=|DB|+|db|
2   k = 1
3   scan db for all X
4   for all X∈L_1^DB or X∈C_1^DB
5     update X.count
6     if X.count ≥ s*|UD|
7         X → L_k^UD
8   for all X ∉ L_1^DB or X∉C_1^DB
9     if X.count+|DB|-1 ≥ s*|UD|
10        X → temp_scanDB
11  k=2
12  while (L_k-1^UD ∪ temp_scanDB_k) > 1
13    C_k = L_k-1^UD * temp_scanDB_k
14    // using Apriori_gen()
15    scan db for all C_k
16    for all X∈ C_k do
17        for all X ∈ L_k^DB
18            update X.count
19            if X.count ≥  s*|UD|
20                X → L_k^UD
21        for all X ∉ L_k^DB
22            if X.count+|DB|-1 ≥ s*|UD|
23                X → temp_scanDB
24    k++
25  end loop
26  rescan_original()
27  L_k^UD = L_k^UD ∪ tempL
28  return L_k^UD
```

Fig. 1. An increment updating phase

Algorithm 2 a re-scanning original database phase ()

```
Input DB, temp_scanDB,s,|UD|
Output tempL
1   if temp_scanDB ≠ ∅
2       scan DB for all X ∈ temp_scanDB
3       update X.count
4       if X.count ≥ s*|UD|
5           X → tempL
6   endif
7   return tempL
```

Fig. 2. A re-scanning original database phase

5. EXPERIMENT

The proposed algorithm in this paper aims to improve the performance of FUP. To evaluate the performance of batch fast update (BFUP) algorithm, this algorithm is implemented and tested on a PC with a 2.93 GHz Intel Core i7 and 3 GB main memory. The experiment is tested with 2 synthetic datasets which are generated by using technique in Agrawal [1]. The first dataset is T10I4D100K which has 100,000 transactions. The second dataset is T10I4D50K which has 50,000 transactions. Both datasets are appended by an increment database that has 1,000 transactions.

For the first original database, i.e., T10I4D100K, the experiment is conducted with 0.1%, 0.3% and 0.5% minimum support thresholds. The average of an execution time is shown in table 2 and figure 3 respectively. For comparison, the results are compared with FUP.

Table 2. Average of Execution time for I10T4D100K

min sup	algorithm	execution time (sec.)	a number of frequent itemset	maximum frequent itemset (size of k)
0.1%	FUP	175890.2901	17,127	L_{10}
	BFUP	43580.3014		
0.3%	FUP	19645.1002	1,991	L_7
	BFUP	11678.5801		
0.5%	FUP	9771.1612	862	L_2
	BFUP	9105.8170		

For the second original database, i.e., T10I4D50K, the experiment is conducted with 0.2%, 0.3% and 0.4% minimum support thresholds. The average of an execution time is shown in table 3 and figure 4 respectively.

From the results of the both datasets, they are shown that an execution time of BFUP is much faster than that of FUP. Furthermore, we observe that the more size k is increasing, the execution time between FUP and BFUP is more different.

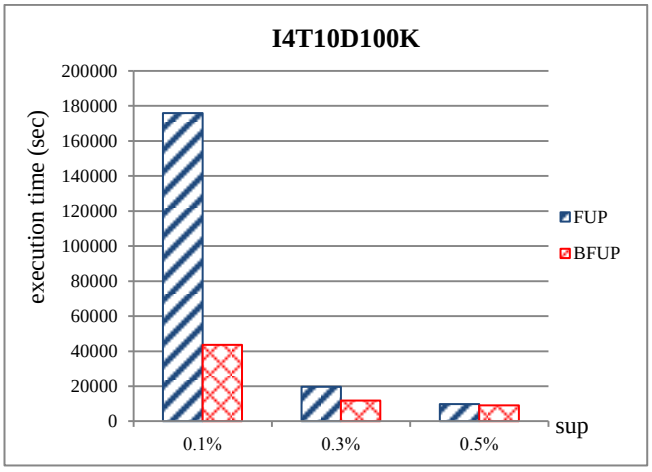


Fig. 3. Execution time comparison for I10T4D100K

Table 3. Average of Execution time for I10T4D50K

min sup	algorithm	execution time (sec.)	a number of frequent itemset	maximum frequent itemset (size of k)
0.2%	FUP	36012.0041	5,307	L_{10}
	BFUP	15402.0908		
0.3%	FUP	17695.8894	1,995	L_7
	BFUP	12906.0013		
0.4%	FUP	12363.9417	1,051	L_3
	BFUP	10866.7162		

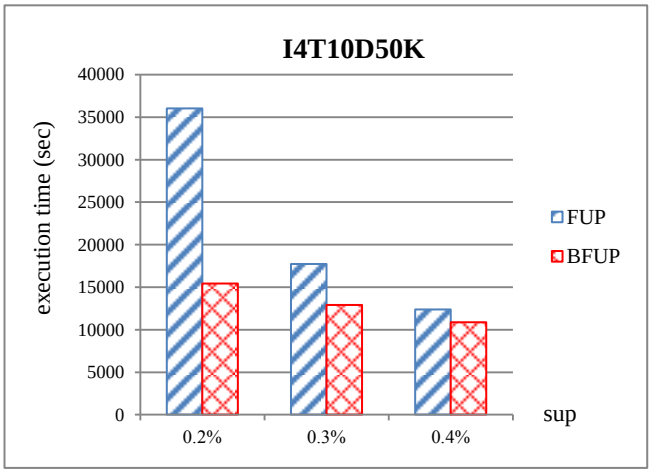


Fig. 4. Execution time comparison for I10T4D50K

6. CONCLUSION

An incremental association rules mining algorithm called batch fast update (BFUP), is proposed. The concept of this algorithm is based from Apriori and FUP algorithm. Although batch fast update algorithm has an execution time better than that of FUP, a large number of temp_scanDB are kept. In the future research, the algorithm for reducing temp_scanDB will be proposed.

7. REFERENCES

[1] Agrawal R, Imielinski T, Swami A (1993), A mining association rules between sets of items in large database, In Proceeding of the ACM SIGMOD Int'l Conf. on Management of Data (ACM SIGMOD'93), Washington, USA, May 1993, pp.207-216.

[2] Agrawal R, Srikant R (1994), Fast algorithm for mining association rules, In Proc. 20th Int. Conf. Very Large DataBases (VLDB'94), Santiago, Chile, September 12-15, 1994, pp.487-499.

[3] Cheung D.W., Han J, Ng V.T., Wong C.Y., Maintenance of Discovered Association rules in Large Databases: An incremental updating technique, In 12th IEEE International Conference on Data Engineering, pp 106-114, 1996.

[4] Tsai Paulry S.M., Lee Chih-Chong, Chen Abree L.P. (2005), An Efficient Approach for Incremental Association Rule Mining, Proceedings of the third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining, Lecture Notes In Computer Science, Vol. 1574 archieve, 1999.