การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะ ด้วยเงื่อนไขบังคับ

นายไพชยนต์ คงไชย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2554

ASSOCIATION RULE DISCOVERY WITH CONSTRAINT LOGIC PROGRAMMING

Phaichayon Kongchai

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2011

การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะ
ด้วยเงื่อนไขบังคับ

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____
(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____
(รศ. ดร.นิตยา เกิดประสพ)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____
(อ. ดร.จิติมนต์ อั่งสกุล)
กรรมการ

_____
(ศ. ดร.ชูกิจ ลิมปิจำนงค์)
รองอธิการบดีฝ่ายวิชาการ

_____
(รศ. ร.อ. ดร.กนต์ธร ชำนิประศาสน์)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ไพชยนต์ คงไชย : การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (ASSOCIATION RULE DISCOVERY WITH CONSTRAINT LOGIC PROGRAMMING) อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร.นิตยา เกิดประสพ, 98 หน้า.

การหากฎความสัมพันธ์เป็นส่วนหนึ่งของการทำเหมืองข้อมูลที่ได้รับความสนใจจากนักวิจัยและผู้ใช้จำนวนมาก เพราะความรู้ที่ได้จากการหากฎความสัมพันธ์นั้นสามารถนำไปใช้ในการจัดสินค้า การจัดแค็ตตาล็อกและการวางแผนส่งเสริมการขาย แต่ในการหากฎความสัมพันธ์นั้นใช้ระยะเวลาในการประมวลผลข้อมูลค่อนข้างมากเพราะความสัมพันธ์ในข้อมูลมีมาก และจะมีจำนวนกฎความสัมพันธ์ที่มากกว่าการทำเหมืองข้อมูลประเภทอื่น ดังนั้นซอฟต์แวร์ทั่วไปจึงสร้างกฎความสัมพันธ์ได้จำนวนมากและบางกฎที่ได้มานั้นไม่มีประโยชน์ต่อผู้ใช้งาน

งานวิจัยนี้ได้เสนอแนวทางแก้ไขปัญหาความล่าช้าของการประมวลผลข้อมูล โดยใช้วิธีการเพิ่มเงื่อนไขบังคับเข้าไปในการวิเคราะห์หากฎความสัมพันธ์ของขั้นตอนวิธีเอไพรออริเพื่อให้ผู้ใช้สามารถระบุกฎที่ปรากฏเฉพาะไอเท็มที่ต้องการได้ และยังสามารถกำหนดความยาวของกฎได้ด้วย การเพิ่มเงื่อนไขบังคับนี้ยังช่วยในการลดระยะเวลาในการวิเคราะห์และลดจำนวนกฎความสัมพันธ์ที่ไม่ได้ใช้ประโยชน์

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2554 ลายมือชื่ออาจารย์ที่ปรึกษา_____

PHAICHAYON KONGCHAI : ASSOCIATION RULE DISCOVERY WITH CONSTRAINT LOGIC PROGRAMMING. THESIS ADVISOR : ASSOC. PROF. NITTAYA KERDPRASOP, Ph.D., 98 PP.

DATA MINING/ASSOCIATION RULE/CONSTRAINT LOGIC PROGRAMMING

Association rule discovery is one of major data mining tasks that has gained much interest from researchers and general users. The knowledge from association mining can be used to recommend product, design catalogs and manage sales promotion. But data processing for association rule discovery has expensive computation time because the relationships induced from data can be tremendously many more than other data mining tasks such as classification. As a consequence, most association mining software generally create so many rules from the association mining process and some of these rules are not beneficial to any users. To solve this problem, we propose to incorporate Apriori algorithm with constraint function for users to specify subset of association rules of interesting items. Users can also identify length of the association rules. Our Apriori-with-constraint algorithm can reduce processing time and reduce a great number of useless rules.

School of Computer Engineering Student's Signature_____

Academic Year 2011 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณ บุคคล และกลุ่มบุคคลต่างๆที่ได้กรุณาให้คำปรึกษา แนะนำ ช่วยเหลืออย่างดียิ่ง ทั้งในด้านวิชาการ และด้านการดำเนินงานวิจัย ดังต่อไปนี้

รองศาสตราจารย์ ดร.นิตยา เกิดประสพ และรองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ให้คำปรึกษาในการทำงานวิจัย การจัดการรูปแบบ และช่วยตรวจทานความถูกต้องของวิทยานิพนธ์

ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาถิวัฒน์ ผู้ช่วยศาสตราจารย์ ดร.คะชา ชาญศิลป์ ผู้ช่วยศาสตราจารย์ สมพันธ์ ชาญศิลป์ และ อ. ดร.ปรเมศวร์ ห่อแก้ว อาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

คุณกัลญา พับโพธิ์ เลขานุการสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ให้ความช่วยเหลือในการประสานงานด้านเอกสารระหว่างศึกษา

คุณวิทวัส วิทยาไกรเลิศ คุณฝนทิพย์ คูณแก้ว คุณสุรสิทธิ์ ท้าวกอกและนักศึกษาบัณฑิตสาขาวิชาวิศวกรรมคอมพิวเตอร์ ทุกท่านที่ให้คำปรึกษาและช่วยเหลือด้วยดีมาโดยตลอด

คุณวราลักษณ์ วงเล็กที่คอยให้กำลังใจ และช่วยจัดการรูปแบบเอกสาร

นอกจากนี้ขอขอบคุณครู อาจารย์ทั้งในอดีตและปัจจุบันที่ให้ความรู้แก่ผู้วิจัยจนประสบความสำเร็จในชีวิต

ท้ายที่สุดที่จะลืมไม่ได้ ขอกราบขอบพระคุณ บิดา มารดา ที่ให้กำเนิด อบรม เลี้ยงดูด้วยความรัก และส่งเสริมการศึกษาเป็นอย่างดีโดยตลอด ทำให้ผู้วิจัยมีความรู้ ความสามารถ มีจิตใจที่เข็มแข็ง รวมทั้งเป็นกำลังใจที่ยิ่งใหญ่แก่ผู้วิจัย จนทำให้ผู้วิจัยประสบความสำเร็จในชีวิตเรื่อยมา

ไพชยนต์ คงไชย

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ปัจจุบันอุปกรณ์ในการเก็บข้อมูลมีขนาดความจุที่มากขึ้นและราคาที่ถูกลง ทำให้คนทั่วไปสามารถใช้งานอุปกรณ์เหล่านี้ได้ซึ่งถ้ามีผู้ใช้มากขึ้นขนาดของข้อมูลก็มากขึ้นเช่นกัน ซึ่งเป็นเหตุทำให้มีการทำเหมืองข้อมูล (Data Mining) มากขึ้นด้วย การทำเหมืองข้อมูลเป็นการค้นหารูปแบบ (Pattern) ของข้อมูลซึ่งมีอยู่ในฐานข้อมูลขนาดใหญ่ โดยอาศัยหลักการสถิติ การรู้จำ การเรียนรู้ของเครื่องและหลักการทางคณิตศาสตร์ ความรู้ที่ได้จากการทำเหมืองข้อมูลมีหลายรูปแบบไม่ว่าจะเป็นการหากฎความสัมพันธ์ (Association Rule) การจำแนกประเภทข้อมูล (Data Classification) การแบ่งกลุ่มข้อมูล (Data Clustering) และความรู้ที่ได้จากการทำเหมืองข้อมูลนั้น สามารถนำไปใช้ในการทำนายข้อมูลในอนาคตหรือช่วยในการตัดสินใจได้

การหากฎความสัมพันธ์นั้นเป็นการหาความสัมพันธ์จากข้อมูลขนาดใหญ่ เพื่อให้ได้รูปแบบความสัมพันธ์ของข้อมูล ในทางธุรกิจมีความสนใจในการหาความสัมพันธ์ของข้อมูลมาก เพราะความรู้ที่ได้จากการหากฎความสัมพันธ์สามารถนำไปใช้ในการวิเคราะห์การซื้อสินค้าของลูกค้า การวางแผนจัดวางชั้นสินค้า รวมไปถึงการวางแผนจัดรายการแนะนำสินค้าหรือแผนจัดรายการกระตุ้นยอดขายสินค้า แต่ในการหากฎความสัมพันธ์ด้วยวิธีการในปัจจุบันมีการใช้ระยะเวลาในการประมวลผลข้อมูลค่อนข้างมากเพราะกฎที่ได้จากการวิเคราะห์ความสัมพันธ์นั้นมีจำนวนมากและบางกฎที่ได้มานั้นไม่มีประโยชน์

ในงานวิจัยนี้ได้เสนอแนวทางแก้ไขปัญหาความล่าช้าของการประมวลผลข้อมูล โดยใช้วิธีการเพิ่มเงื่อนไขบังคับ (Constraint) เข้าไปในการวิเคราะห์หากฎความสัมพันธ์ เพื่อให้ผู้ใช้ (User) สามารถระบุกฎที่ปรากฏเฉพาะไอเท็ม (Item) ที่ต้องการ กำหนดความยาวของกฎ และยังสามารถกำหนดไอเท็มที่เป็นเป้าหมายได้ การเพิ่มเงื่อนไขบังคับนี้ยังช่วยในการลดระยะเวลาในการวิเคราะห์และลดการใช้ทรัพยากร (Memory) งานวิจัยนี้ได้ใช้เทคนิคการเขียนโปรแกรมเชิงตรรกะ (Logic Programming) มาใช้ในการเขียนโปรแกรมการทำงานของการหากฎความสัมพันธ์ (กิตติ ศักดิ์ เกิดประสพ และคณะ, 2551) และในงานวิจัยนี้ได้แสดงการเปรียบเทียบประสิทธิภาพในเรื่องของเวลาและความถูกต้องของกฎความสัมพันธ์ที่ได้รับ โดยอัลกอริทึมที่ใช้ในการเปรียบเทียบคือ

อัลกอริทึมเอไพรออริ (Apriori) อัลกอริทึมเอไพรออริที่มีการใช้เงื่อนไขบังคับระหว่างการหาไอเท็มเซตปรากฏบ่อย ACIF (Association Rule Discovery With Constraints In Frequent Itemset Mining) และอัลกอริทึมเอไพรออริที่มีการใช้เงื่อนไขบังคับหลังการหาไอเท็มเซตปรากฏบ่อย ACAF (Association Rule Discovery With Constraints After Frequent Itemset Mining) โดยแนวคิดและขั้นตอนการทำงานของทั้งสามอัลกอริทึมนี้จะมีรายละเอียดในบทที่ 3 และบทที่ 4 และในส่วนท้ายของงานวิจัยจะมีการสรุปข้อดีข้อเสียของแต่ละอัลกอริทึม

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อศึกษาและพัฒนาอัลกอริทึมการหากฎความสัมพันธ์ (Association Rule) ให้มีความเร็วในการประมวลผลมากขึ้น
2. เพื่อศึกษาและพัฒนาอัลกอริทึมการหากฎความสัมพันธ์ โดยให้ผู้ใช้ สามารถกำหนดเงื่อนไขบังคับเกี่ยวกับสมาชิกของกฎความสัมพันธ์ ความยาวของกฎความสัมพันธ์ และเป้าหมายของกฎความสัมพันธ์ได้
3. เพื่อศึกษาและพัฒนาการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (Constraint Logic Programming) และใช้พัฒนาอัลกอริทึมที่ปรับปรุงจากอัลกอริทึมเอไพรออริเพื่อเพิ่มประสิทธิภาพความถูกต้องของจำนวนกฎความสัมพันธ์
4. เพื่อเปรียบเทียบหาอัลกอริทึมที่เหมาะสมในการหากฎความสัมพันธ์ด้วยการใช้เงื่อนไขบังคับ

1.3 ข้อตกลงเบื้องต้น

1. ข้อมูลที่นำมาใช้จะต้องเป็นข้อมูลที่เหมาะสำหรับการหากฎความสัมพันธ์จึงจะเป็นประโยชน์
2. การที่จะได้กฎความสัมพันธ์ตามความต้องการของผู้ใช้ ผู้จะใช้ต้องกำหนดไอเท็มที่ต้องการไว้ด้วย
3. กฎความสัมพันธ์ ที่ได้มานั้นจะอยู่ในรูปแบบ ถ้า...แล้ว... (If… Then…) และจะมีการระบุค่าสนับสนุน (Support) และค่าความเชื่อมั่น (Confidence) ด้วย
4. รูปแบบของข้อมูลที่ใช้ในการประมวลจะต้องอยู่ในรูปแบบของภาษาโปรล็อก
5. ในการสอบถามเพื่อหากฎความสัมพันธ์จากโปรแกรมจะต้องใส่เงื่อนไขในกล่องข้อความให้ครบทั้งสามกล่อง

1.4 ขอบเขตของการวิจัย

1. ใช้หลักการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับพัฒนาอัลกอริทึมหากฎความสัมพันธ์เพื่อเพิ่มประสิทธิภาพการประมวลผล
2. พัฒนาอัลกอริทึมการหากฎความสัมพันธ์ที่ใช้อัลกอริทึม Apriori (Agrawal et al., 1994) เป็นพื้นฐาน โดยให้ผู้ใช้สามารถกำหนดสมาชิกของกฎได้
3. ข้อมูลที่นำมาใช้จะต้องเป็นข้อมูลที่เอื้อต่อการหากฎความสัมพันธ์ (Association Rule) และมีขนาดไม่ใหญ่เกินความจุหน่วยความจำหลักของเครื่องคอมพิวเตอร์
4. งานวิจัยนี้พัฒนาอัลกอริทึมโดยใช้โปรแกรมอีคลิปส์ (ECLiPSe) ซึ่งถ้าหากนำซอร์ซโค้ด (Source Code) ที่พัฒนาด้วยโปรแกรมอีคลิปส์นี้ ไปประมวลผลกับโปรแกรมเชิงตรรกะอื่นจะไม่สามารถประมวลผลได้เพราะซอร์ซโค้ดมีเพรดิเคตพิเศษ (Special Predicate) ซึ่งใช้ได้เฉพาะโปรแกรมอีคลิปส์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

จากการศึกษาและพัฒนางานวิจัยนี้ผู้วิจัยคาดว่าจะเกิดประโยชน์ต่อผู้ใช้ในการใช้งานโปรแกรมเพื่อค้นหากฎความสัมพันธ์ดังนี้

1. ผู้ใช้มีความสะดวกในการระบุไอเท็มที่ต้องการหรือไม่ต้องการในการหากฎความสัมพันธ์ได้
2. ผู้ใช้สามารถกำหนดขนาดของกฎความสัมพันธ์ที่ต้องการได้
3. เวลาที่ใช้ในการวิเคราะห์หากฎความสัมพันธ์ ควรจะต้องลดลง
4. กฎความสัมพันธ์ที่ได้จะสามารถวิเคราะห์และทำความเข้าใจได้ง่าย
5. ผู้ใช้สามารถกำหนดเป้าหมายของกฎได้ (ไอเท็มที่ปรากฏหลังข้อความ then)
6. ผู้ใช้สามารถเลือกใช้อัลกอริทึมทั้งสาม (Apriori, ACIF, ACAF) นี้ได้เหมาะสมกับขนาดของข้อมูลหรือประเภทของงาน

บทที่ 2
ปริทัศน์วรรณกรรม

เนื้อหาในบทนี้ประกอบด้วยการทบทวนวรรณกรรมและงานวิจัยที่เกี่ยวข้อง โดยมีรายละเอียดของการทำเหมืองข้อมูล (Data Mining) การหากฎความสัมพันธ์ (Association Rule) ภาษาสำหรับการเขียนโปรแกรมเชิงตรรกะ (Prolog) การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (Constraint Logic Programming) โปรแกรมอีคลิปส์ (ECLiPSe) และงานวิจัยที่เกี่ยวข้อง

2.1 การทำเหมืองข้อมูล (Data mining)

การทำเหมืองข้อมูล คือการค้นหารูปแบบ (Pattern) หรือกฎที่ซ่อนอยู่ในข้อมูลเพื่อกลั่นกรองแยกแยะสาระที่เป็นประโยชน์ที่ได้จากฐานข้อมูลออกมา โดยอาศัยหลักสถิติ คณิตศาสตร์ การเรียนรู้ของเครื่อง (Machine Learning) (Seifert, 2004) และข้อมูลที่นำมาใช้เพื่อทำเหมืองข้อมูลจะมีขนาดใหญ่วัตถุประสงค์ของการทำเหมืองข้อมูลมีได้หลายแบบ เช่น การจำแนกประเภทข้อมูล (Classification) การแบ่งกลุ่มข้อมูล (Clustering) และการหากฎความสัมพันธ์ (Association Rule) ความรู้ที่ได้จากการทำเหมืองข้อมูลนั้น สามารถนำไปใช้ในการทำนายข้อมูลในอนาคตหรือช่วยในการตัดสินใจ ในงานวิจัยนี้เน้นเฉพาะวิธีการหากฎความสัมพันธ์จากข้อมูลขนาดใหญ่

2.1.1 วิวัฒนาการของการทำเหมืองข้อมูล

การรวบรวมและใช้ประโยชน์จากข้อมูลนั้นได้เริ่มต้นตั้งแต่การรวบรวมข้อมูลแบบแฟ้ม จนกระทั่งถึงการหาความรู้จากข้อมูลซึ่งมีพัฒนาการดังต่อไปนี้ (ณัฏภัทรศญา ทับทิมเทศ, 2550)

- ปี 1960 การเก็บรวบรวมข้อมูล และการสร้างฐานข้อมูลได้เริ่มพัฒนามาจากการประมวลผลแฟ้ม
- ปี 1970 เกิดการพัฒนาระบบการเก็บข้อมูลในรูปแบบตาราง (Relational Database System) มีการจัดเก็บอยู่ในรูปแบบตารางมีลักษณะเป็น 2 มิติ คือ แถว (Row) และคอลัมน์(Column) และเหมาะสำหรับงานเลือกดูข้อมูลแบบหลายฟิลด์ มีการป้องกันข้อมูลถูกทำลายหรือแก้ไขได้ดี การเลือกดูข้อมูลทำได้ง่าย□

- ปี 1980 มีการพัฒนาเทคนิค Data Access คือ การนำข้อมูลที่จัดเก็บมาสร้างความสัมพันธ์ต่อกันในข้อมูล เพื่อประโยชน์ในการนำไปวิเคราะห์ และการตัดสินใจอย่างมีคุณภาพ
- ปี 1990 เกิดระบบ Data Warehouse และ Decision Support System อยู่ในลักษณะการเก็บข้อมูลลงในฐานข้อมูลที่มีขนาดใหญ่ โดยครอบคลุมส่วนประกอบทั้งหมดขององค์กร เพื่อช่วยสนับสนุนการตัดสินใจ
- ปี 2000 มีการพัฒนา Data Mining เพื่อนำข้อมูลจากฐานข้อมูลมาวิเคราะห์และประมวลผล โดยการสร้างแบบจำลองและความสัมพันธ์ โดยใช้วิธีการทางสถิติเป็นพื้นฐานสำคัญ

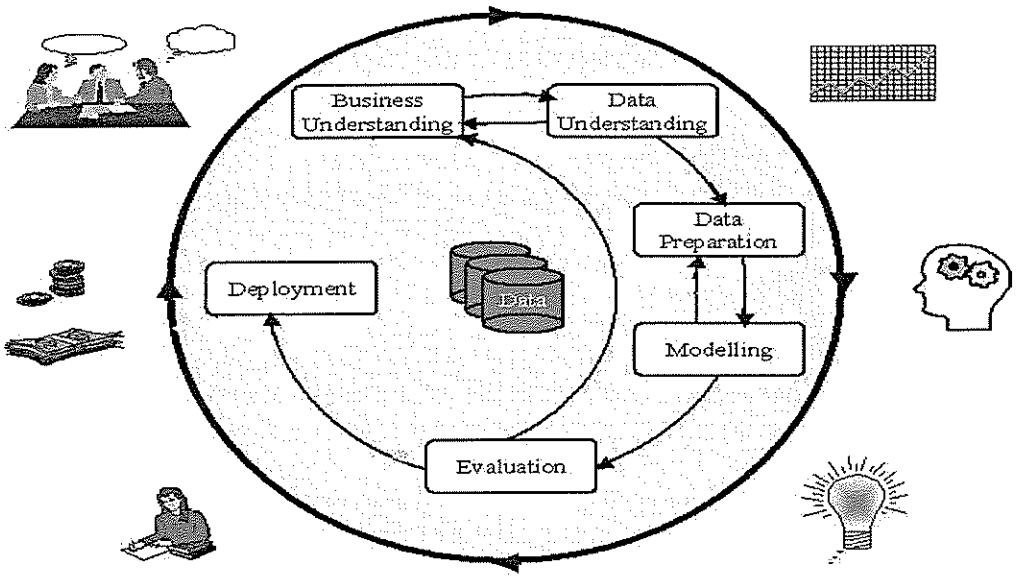
2.1.2 วัตถุประสงค์ในการทำเหมืองข้อมูล

การทำเหมืองข้อมูลคือการหารูปแบบของข้อมูลและมีจุดประสงค์ ดังนี้ (ณัฏภัทรศญา ทับทิมเทศ, 2550)

1) เพื่อค้นหาความรู้ใหม่ในฐานข้อมูล (Knowledge Discovery in Databases)
2) เพื่อช่วยในการวิเคราะห์และคัดแยกหาความรู้ที่ซ่อนเร้นอยู่ภายในฐานข้อมูล (Knowledge Extraction)
3) เพื่อจัดการกับข้อมูลในอดีตที่มีความสัมพันธ์กับข้อมูลปัจจุบัน มาสร้างโมเดลหรือรูปแบบที่มีประโยชน์ต่อการทำเหมืองข้อมูล (Data Archeology)
4) เพื่อสำรวจข้อมูลเพื่อใช้ในการวางแผนทำเหมืองข้อมูล (Data Exploration)
5) เพื่อค้นหารูปแบบ (Pattern) ของข้อมูลที่ซ่อนอยู่ในฐานข้อมูล ซึ่งจะเป็นประโยชน์ต่อการทำนายผลของฐานข้อมูลนั้นๆ ว่ามีทิศทางไปทางไหน (Data Pattern Processing)
6) เพื่อใช้ขุดเจาะข้อมูลในเชิงลึก และอาจจะแสดงผลออกมาทางด้านตัวเลขสถิติ (Data Dredging)

2.1.3 กระบวนการทำเหมืองข้อมูล

มาตรฐานในกระบวนการทำเหมืองข้อมูลเรียกว่า CRISP-DM (Cross Industry Standard Process for Data Mining) กำหนดว่ากระบวนการทำเหมืองข้อมูลมี 6 ขั้นตอน (รูปที่ 2.1)



รูปที่ 2.1 กระบวนการทำเหมืองข้อมูล (Squier, 2001)

1) การทำความเข้าใจในเรื่องธุรกิจที่จะทำ (Business Understanding) เป็นขั้นตอนที่สำคัญที่สุดเพราะเราต้องเข้าใจวัตถุประสงค์ของการทำเหมืองข้อมูลของเราก่อน ว่าเราต้องการหาความรู้อะไรจากข้อมูล จากนั้นจึงเริ่มวางแผนขั้นตอนวิเคราะห์ ขั้นตอนออกแบบ และหาวิธีการแก้ปัญหา
2) การทำความเข้าใจกับข้อมูลที่จัดเก็บ (Data Understanding) ขั้นตอนนี้เป็นการรวบรวมข้อมูลที่เกี่ยวข้อง เพื่อใช้ในการทำเหมืองข้อมูลและในการรวบรวมข้อมูลนั้นควรพิจารณาแหล่งที่มาของข้อมูล ว่าข้อมูลที่ได้มาถูกต้องน่าเชื่อถือหรือไม่ข้อมูลที่ได้มีปริมาณเพียงพอที่จะทำการหาความรู้จากการทำเหมืองข้อมูลไหม
3) การเตรียมข้อมูล (Data Preparation) ในขั้นตอนนี้คือการเตรียมข้อมูลเพื่อนำไปใช้ในการทำเหมืองข้อมูล โดยจะมีการคัดเลือกข้อมูลเป็นการเลือกข้อมูลให้ตรงกับงานที่เราจะทำเหมืองข้อมูล การกรองข้อมูลก็คือการทำความสะอาดข้อมูล บางข้อมูลอาจจะมีการกรอกข้อมูลที่ไม่ครบอาจจะทำให้ความรู้ที่ได้มาไม่ถูกต้อง งานในขั้นนี้จะรวมถึงการแปลงข้อมูลให้อยู่ในรูปแบบที่เหมาะสมกับการทำเหมืองข้อมูลในแต่ละอัลกอริทึม
4) การสร้างโมเดล (Modeling) เป็นขั้นตอนการสร้างรูปแบบข้อมูลจากข้อมูลที่เราได้เตรียมมาจากขั้นตอนที่ 3 โดยรูปแบบข้อมูลที่ได้มานั้นเรียกว่า ความรู้

5) การประเมินผล (Evaluation) เป็นการประเมินประสิทธิภาพของความรู้ที่ได้ว่ามีความถูกต้องมากเพียงใด
6) การนำไปใช้งาน (Deployment) ความรู้ที่ได้จากการทำเหมืองข้อมูลนั้นสามารถนำไปใช้งานได้ เช่น การวินิจฉัยโรค การพยากรณ์อากาศ การจัดโปรโมชั่นสินค้า

2.2 การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ (Association Mining)

การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ เป็นการหาความสัมพันธ์ของข้อมูลจากข้อมูลขนาดใหญ่เพื่อนำไปใช้ในการวิเคราะห์รูปแบบของข้อมูล เช่น ลูกค้าที่เข้ามาซื้อของในห้างสรรพสินค้าจะมีรูปแบบการซื้อของเป็นอย่างไร โดยวิธีการจะเริ่มจากการบันทึกรายการซื้อของของลูกค้าแต่ละคนไว้ เมื่อได้ข้อมูลมากพอก็จะนำข้อมูลนั้นไปหากฎความสัมพันธ์ จากการทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ ผลลัพธ์ที่ได้การทำเหมืองข้อมูลก็คือรูปแบบการซื้อของของลูกค้าที่มาซื้อของที่ห้างสรรพสินค้าซึ่งจะอยู่ในลักษณะ ถ้า => แล้ว (If-Then) เช่น Sandwich => Coke แปลว่าถ้าลูกค้าซื้อแซนด์วิชแล้วลูกค้าจะซื้อโค้กด้วย รูปแบบกฎความสัมพันธ์ที่ได้มานี้จะเป็นประโยชน์ต่อผู้ประกอบการในการวางแผนจัดวางชั้นสินค้า รวมไปถึงการวางแผนจัดรายการแนะนำสินค้าหรือแผนจัดรายการกระตุ้นยอดขายสินค้า การวิเคราะห์การซื้อสินค้าของลูกค้าแบบนี้เรียกว่า "Market Basket Analysis" เนื่องจากกฎความสัมพันธ์ลักษณะนี้มักจะถูกนำไปใช้ในเชิงธุรกิจ (Han and Kamber, 2000)

การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ ขั้นตอนที่สำคัญคือ การค้นหารูปแบบที่ปรากฏบ่อย (Frequent Pattern Mining) ซึ่งผลลัพธ์ที่ได้จะอยู่ในรูปแบบเซตของกลุ่มข้อมูลที่มักจะปรากฏร่วมกันจากนั้นข้อมูลในแต่ละเซตจะถูกนำไปสร้างเป็นกฎความสัมพันธ์ การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์มีการเสนอความคิดและเทคนิคไว้เป็นจำนวนมาก และอัลกอริทึม AIS เป็นเทคนิคหนึ่งในการสืบค้นกลุ่มข้อมูลที่เกิดขึ้นบ่อยในฐานข้อมูลคิดค้นโดย Agrawal และคณะเมื่อปี 1993

2.2.1 นิยามการวิเคราะห์กฎความสัมพันธ์

กำหนดให้ $I = \{i_1, i_2, \ldots, i_n\}$ เป็นเซตของข้อมูลและเรียกแต่ละข้อมูลว่าไอเท็ม (Item) กำหนดให้ $D = \{ t_1, t_2, \ldots, t_n \}$ เป็นฐานข้อมูลซึ่งประกอบไปด้วยชุดข้อมูล $T_i = \{ i_{i1}, i_{i2}, \ldots, i_{ik} \}$ เป็นชุดข้อมูล (Transaction) ที่ประกอบขึ้นจากกลุ่มข้อมูลหรือเรียกว่าไอเท็มเซต (Itemset) โดยที่ $T \subseteq I$ การวิเคราะห์หารูปแบบที่ปรากฏบ่อยจะต้องระบุค่าสนับสนุน (Support) สามารถหาค่าสนับสนุนได้จากการคำนวณไอเท็มเซตจากทรานแซคชันทั้งหมด ถ้าค่าสนับสนุนของทุกไอเท็ม

ในไอเท็มเซตใดมีค่ามากกว่าค่าสนับสนุนที่น้อยที่สุด (Minimum Support) เรียกเซตนี้ว่าไอเท็มเซตปรากฏบ่อย (Frequent Itemset) ไอเท็มจากเซตที่ปรากฏบ่อยนี้จะถูกนำมาสร้างให้เป็นกฎความสัมพันธ์ โดยจะวัดความน่าเชื่อถือของกฎที่ได้ จากค่าความเชื่อมั่น (Confidence) ซึ่งเขียนในรูปแบบ conf(X => Y) หรือ X => Y (c%) โดยที่ X และ Y เป็น Itemset ค่าความน่าเชื่อถือของกฎหรือค่าความเชื่อมั่น คืออัตราส่วนของจำนวนทรานแซคชันที่มีทั้ง Itemsets X และ Y ต่อจำนวนทรานแซคชันที่มี Itemset X โดยที่ $X \subset I$, $Y \subset I$ และ $X \cap Y = \varnothing$

การหาค่าสนับสนุนของไอเท็ม A หาได้จาก

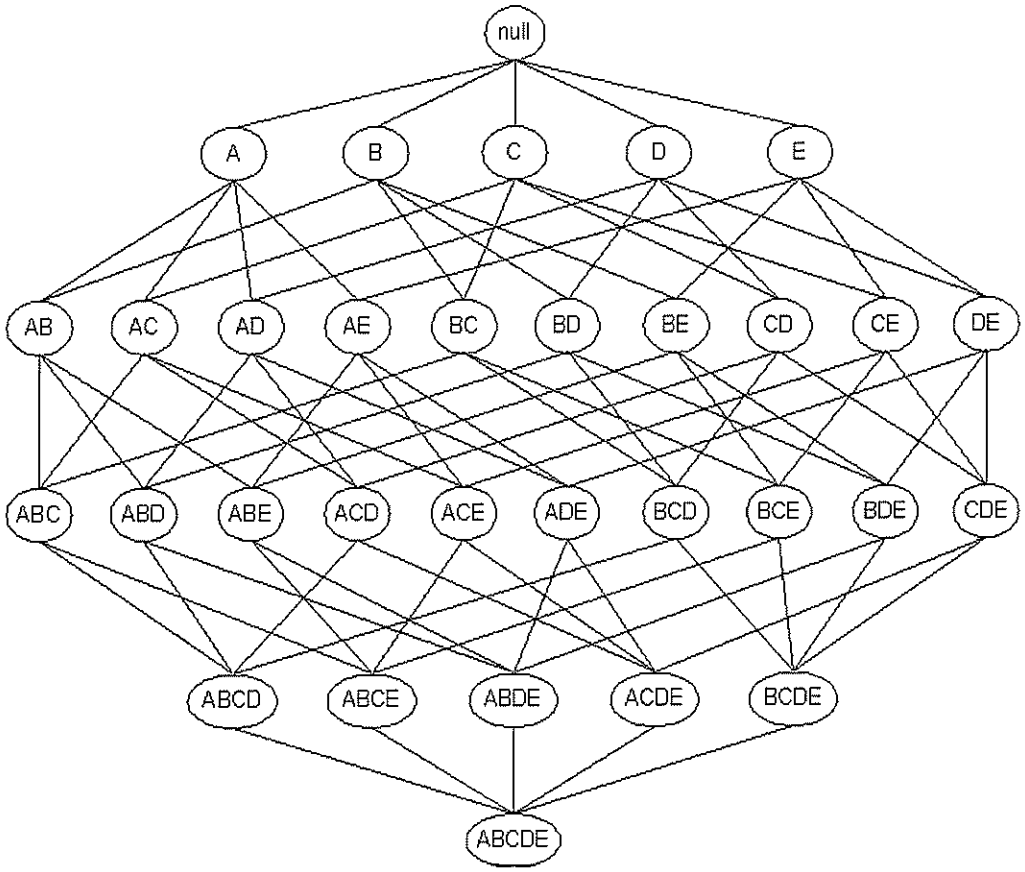
$$support(A) = \frac{number\ of\ transactions\ that\ contain\ A}{number\ of\ all\ transactions}$$

การหาค่าความเชื่อมั่นของกฎ $A => B$ หาได้จาก

$$\text{confidence}(A => B) = \frac{support\ (A\ and\ B)}{support(A)}$$

2.2.2 ไอเท็มเซตปรากฏบ่อย (Frequent Itemset)

ข้อมูลหรือไอเท็มที่ปรากฏในทรานแซคชันด้วยความถี่ไม่ต่ำกว่าเกณฑ์ Minimum Support จะเรียกว่า ไอเท็มเซตปรากฏบ่อย ขอบเขตการค้นหาไอเท็มเซตปรากฏบ่อย แสดงได้ด้วยโครงสร้างแลตทิซ (Lattice Structure) และจำนวนไอเท็มเซตที่เป็นไปได้ทั้งหมดจากเซตของ I = {A, B, C, D, E} แสดงได้ดัง รูปที่ 2.2 ซึ่งมีขนาดของ Itemset จากระดับชั้นที่ 1 (1-itemset) ถึงระดับชั้นที่ 5 (5-itemset)

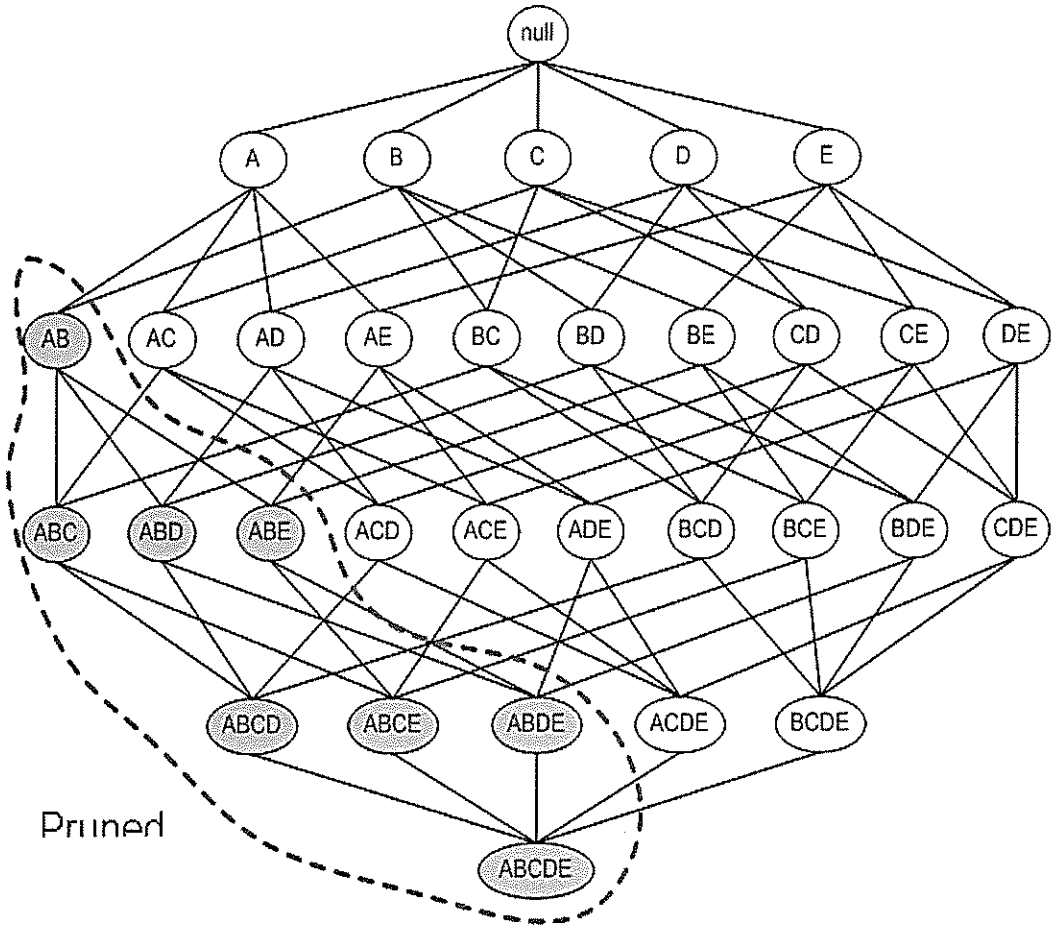


รูปที่ 2.2 โครงสร้างแลตทิซ (Lattice Structure) ของการค้นหาไอเท็มเซตปรากฏบ่อย จากเซตของไอเท็ม I = {A, B, C, D, E}

จำนวน Itemsets ที่มีโอกาสเป็น Frequent Itemsets ทั้งหมดคำนวณได้จาก $2^k - 1$ โดย k คือ จำนวนไอเท็มทั้งหมด จากรูปจะเห็นได้ว่าค่า k มีค่าเท่ากับ 5 ซึ่งแทนค่า k เข้าไปใน $2^k - 1$ จึงได้จำนวนไอเท็มเซตปรากฏบ่อยทั้งหมด 31 เซต และถ้าจำนวน k สูงขึ้นโครงสร้างแลตทิซจะใหญ่ขึ้นและซับซ้อนมากขึ้นทำให้การคำนวณและการค้นหามีปริมาณมากขึ้นด้วย ซึ่งจะทำให้คอมพิวเตอร์ใช้ทรัพยากรมากขึ้นและการทำงานก็จะช้าลง

2.2.3 อัลกอริทึมเอไพรออริ (Apriori Algorithm)

ในปี 1994 Rakesh Agrawal และ Ramakrishnan Srikant ได้ปรับปรุงอัลกอริทึม AIS ให้ทำงานได้เร็วขึ้นและได้เรียกอัลกอริทึมที่พัฒนาขึ้นใหม่นี้ว่า อัลกอริทึม Apriori โดยใช้เทคนิค Support-based Pruning เพื่อช่วยตัดหรือลดจำนวน Candidate Itemsets แทนที่จะแจกแจงทั้งหมด ดังรูปที่ 2.3



รูปที่ 2.3 โครงสร้างแลตทิซ (Lattice Structure) ที่มีการใช้เทคนิคการพรุน (Pruning)

จากรูปจะเห็นได้ว่ามีการทำเส้นประตั้งแต่ไอเท็มเซต {A, B} ที่มีสองไอเท็ม จนมาถึงไอเท็มเซต {A, B, C, D, E} ที่มีห้าไอเท็มซึ่งมีความหมายว่า ไม่สนใจไอเท็มที่มี {A, B} เป็นสับเซต เนื่องจากค่า Support ของ A และ B ไม่ถึงเกณฑ์ Minimum Support เทคนิคนี้จึงได้ตัดเซตที่มี A และ B เป็นสมาชิกออกเพื่อช่วยลดเวลาในการสร้าง Candidate Itemsets และการทำงานของอัลกอริทึมเอไพรออริมีการทำงานเป็นแบบลูป (Loop) หรือเรียกว่า Level-wise คือจะทำงานทีละระดับ จากโครงสร้างแลตทิซ ระดับที่ 1 จนถึงระดับที่ k ในการสร้าง Frequent Itemsets

ตัวอย่างการหากฎความสัมพันธ์ เช่น สมมติว่ามีลูกค้ามาซื้อของที่ร้านจำนวน 4 คน (ตารางที่ 2.1) เราต้องการทราบว่าลูกค้าที่เข้ามาซื้อของในร้านมีรูปแบบการซื้อเป็นอย่างไร โดยกำหนดให้ค่าสนับสนุนต่ำสุดเท่ากับ 2 ทรานแซคชันจากทั้งหมด 4 ทรานแซคชัน (หรือคิดเป็นค่า Support เท่ากับ 50%) และค่าความเชื่อมั่นเท่ากับ 100%

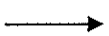
ตารางที่ 2.1 รายการซื้อของของลูกค้า 4 ราย

TID	Items
001	Eggs, Coke, Milk
002	Bread, Coke, Diaper
003	Eggs, Bread, Coke, Diaper
004	Bread, Diaper

จากรายการซื้อของของลูกค้า 4 รายหรือเรียกว่า 4 ทรานแซคชัน มีการซื้อสินค้าอยู่ 5 ชนิด คือ Eggs, Bread, Coke, Milk, Diaper โดยสินค้าเหล่านี้เรียกว่าไอเท็ม

ขั้นตอนแรกของการหากฎความสัมพันธ์คือ นับไอเท็มว่าแต่ละไอเท็มปรากฏในทรานแซคชันด้วยค่าความถี่มากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุดหรือไม่ (รูปที่ 2.4) จากรูปจะเห็นได้ว่าตารางด้านซ้ายมือมีไอเท็ม Milk ที่มีค่าสนับสนุนเป็น 1 ซึ่งต่ำกว่าค่าสนับสนุนต่ำสุดซึ่งกำหนดไว้เป็น 2 ไอเท็ม สินค้า Milk จึงไม่นำมาพิจารณาในการหากฎความสัมพันธ์ และตารางทางด้านขวาคือไอเท็มที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุด ไอเท็มเหล่านี้เรียกว่า Frequent 1-itemset และจะถูกนำไปพิจารณาหา Frequent 2-itemset ต่อไป

Item	Support
Eggs	2
Bread	3
Coke	3
Milk	1
Diaper	3



Item	Support
Eggs	2
Bread	3
Coke	3
Diaper	3

รูปที่ 2.4 ไอเท็มที่มีค่าสนับสนุนที่มากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุด Frequent 1-itemset

หลังจากได้ Frequent 1-itemset แล้วไอเท็มเซตเหล่านี้จะถูกนำไปจับคู่เพื่อสร้างเซตของสองไอเท็มและหาค่าสนับสนุนเพื่อหา Frequent 2-itemset (รูปที่ 2.5) จากรูปตารางด้านซ้ายมือมีไอเท็มเซต {Eggs, Bread} และ {Eggs, Diaper} ที่มีค่าสนับสนุนเป็น 1 ซึ่งต่ำกว่าค่าสนับสนุนต่ำสุด ไอเท็มเซตทั้งสองจึงไม่เรียกว่า Frequent 2-itemset และจะถูกตัดออกจากการพิจารณา และตารางทางด้านขวาคือไอเท็มที่มีค่าสนับสนุนมากกว่าค่าสนับสนุนต่ำสุด ไอเท็มเหล่านี้เรียกว่า Frequent 2-itemset

Item	Support
Eggs, Bread	1
Eggs, Coke	2
Eggs, Diaper	1
Bread, Coke	2
Bread, Diaper	3
Coke, Diaper	2

→

Item	Support
Eggs, Coke	2
Bread, Coke	2
Bread, Diaper	3
Coke, Diaper	2

รูปที่ 2.5 เซตของสองไอเท็มที่มีค่าสนับสนุนที่มากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุด Frequent 2-itemset

หลังจากได้ Frequent 2-itemset แล้วไอเท็มเซตเหล่านี้จะถูกนำไปจับคู่เพื่อสร้างเซตของสามไอเท็มและหาค่าสนับสนุนเพื่อหา Frequent 3-itemset (รูปที่ 2.6) จากรูปตารางด้านซ้ายมือมีไอเท็มเซต {Eggs, Coke, Bread} และ {Eggs, Coke, Diaper} มีค่าสนับสนุนเป็น 1 ซึ่งต่ำกว่าค่าสนับสนุนต่ำสุด ไอเท็มเซตทั้งสองจึงไม่นำมาพิจารณาในขั้นตอนต่อไป และตารางทางด้านขวาคือไอเท็มที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุด ไอเท็มเหล่านี้เรียกว่า Frequent 3-itemset และเนื่องจาก Frequent 3-itemset ที่ได้มีเพียงเซตเดียว คือ {Bread, Coke, Diaper} จึงไม่สามารถนำไปจับคู่เพื่อสร้างเป็นเซตของสี่ไอเท็มได้ ขั้นตอนแรกของการค้นหากฎความสัมพันธ์ที่เป็นการสร้างไอเท็มเซตที่พบบ่อยจึงสิ้นสุดที่การสร้าง Frequent 3-itemset

Item	Support
Eggs, Coke, Bread	1
Eggs, Coke, Diaper	1
Bread, Coke, Diaper	2

Item	Support
Bread, Coke, Diaper	2

รูปที่ 2.6 เซตของสามไอเท็มที่มีค่าสนับสนุนที่มากกว่าหรือเท่ากับค่าสนับสนุนต่ำสุด Frequent 3-itemset

ขั้นตอนที่สองของการหากฎความสัมพันธ์คือ การสร้างกฎความสัมพันธ์โดยการสร้างจะนำข้อมูล ตั้งแต่ Frequent 2-itemset ขึ้นไปมาใช้ จากขั้นตอนแรกทำให้ได้ Frequent 2-itemset ประกอบด้วย {Eggs, Coke}, {Bread, Coke}, {Bread, Diaper}, {Coke, Diaper} และ Frequent 3-itemset ประกอบด้วย {Bread, Coke, Diaper} จึงสามารถสร้างกฎได้ดังนี้

Eggs => Coke	(confidence = 2/2 = 100%)
Coke => Eggs	(confidence = 2/3 = 67%)
Bread => Coke	(confidence = 2/3 = 67%)
Coke => Bread	(confidence = 2/3 = 67%)
Bread => Diaper	(confidence = 3/3 = 100%)
Diaper => Bread	(confidence = 3/3 = 100%)
Coke => Diaper	(confidence = 2/3 = 67%)
Diaper => Coke	(confidence = 2/3 = 67%)
Bread, Coke => Diaper	(confidence = 2/2 = 100%)
Bread, Diaper => Coke	(confidence = 2/3 = 67%)
Coke, Diaper => Bread	(confidence = 2/2 = 100%)
Coke => Diaper, Bread	(confidence = 2/3 = 67%)
Diaper => Coke, Bread	(confidence = 2/3 = 67%)
Bread => Diaper, Coke	(confidence = 2/3 = 67%)

จากการกำหนดค่าความเชื่อมั่นตอนแรกไว้ที่ 100% จึงทำให้ได้รับกฎความสัมพันธ์ที่มีค่ามากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำจำนวน 5 กฎความสัมพันธ์ จากกฎความสัมพันธ์ทั้งหมด 14 กฎความสัมพันธ์ดังนี้

Eggs	=> Coke	(confidence = 2/2 = 100%)
Bread	=> Diaper	(confidence = 3/3 = 100%)
Diaper	=> Bread	(confidence = 3/3 = 100%)
Bread, Coke	=> Diaper	(confidence = 2/2 = 100%)
Coke, Diaper	=> Bread	(confidence = 2/2 = 100%)

2.3 ภาษาโปรล็อก (Prolog)

ภาษาโปรล็อกเป็นภาษาสำหรับการเขียนโปรแกรมเชิงตรรกะย่อมาจาก PROgrammation en LOGique (Logic programming) สร้างขึ้นโดย Alain Colmerauer ราว ค.ศ. 1972 ภาษาโปรล็อกเกิดจากความพยายามที่จะสร้างภาษาที่อาศัยวิธีการทางตรรกศาสตร์แทนที่จะกำหนดคำสั่งอย่างละเอียดให้กับคอมพิวเตอร์ (วิกิพีเดีย สารานุกรมเสรี, 2554)

ภาษาโปรล็อกถูกนำไปใช้ในการแก้ปัญหาทางปัญญาประดิษฐ์ที่เป็นการคำนวณเชิงสัญลักษณ์และใช้พื้นฐานของตรรกะ ภาษาโปรล็อกเป็นภาษาเชิงพรรณนา (Descriptive Language) ซึ่งเป็นภาษาชั้นสูงเกี่ยวกับการจัดการเรื่องความสัมพันธ์ของสัญลักษณ์ต่างๆ ภาษานี้มีความแตกต่างมากจากภาษาเชิงกระบวนคำสั่ง (Procedural Language) (บุญเสริม กิจศิริกุล, 2548)

ภาษา Prolog มีพื้นฐานมาจากแคลคูลัสภาคแสดง (Predicate Calculus) มีแนวคิดพื้นฐานที่เกี่ยวข้องได้แก่ การทำให้เท่ากัน (Unification) การเรียกซ้ำจากส่วนท้าย (Tail Recursion) การย้อนรอย (Backtracking)

2.3.1 ชนิดข้อมูลในภาษาโปรล็อก

1. อะตอม (Atom)

คือค่าที่แทนด้วยข้อความ ซึ่งประกอบด้วย ตัวเลข ตัวอักษร เส้นใต้อักขระและจะต้องเป็นตัวพิมพ์เล็กเพราะถ้าเป็นตัวพิมพ์ใหญ่โปรแกรมจะมองเป็นตัวแปร แต่ถ้าต้องการใช้อักขระพิเศษให้ใช้อัญประกาศครอบ เช่น '-' , '+'

2. ตัวเลข (Number)

ประกอบด้วยเลขจำนวนเต็มกับเลขจำนวนจริง เช่น 123 และ 123.45

3. ตัวแปร (Variables)

ตัวแปรจะแสดงด้วยสายอักขระที่ประกอบด้วย ตัวอักษร ตัวเลข และเส้นใต้อักขระ โดยจะต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่ ตัวแปรในภาษาโปรล็อกไม่ใช่ที่เก็บข้อมูล แต่จะมีลักษณะคล้ายรูปแบบ (Pattern) ซึ่งกำหนดไว้ในเรื่องการทำให้เท่ากัน

ตัวแปรนิรนาม (Anonymous Variable) จะเขียนโดยใช้เครื่องหมายเส้นใต้อักขระเพียงตัวเดียว (_)

4. พจน์ (Term)

พจน์ (Term) ประกอบด้วย ส่วนหัว (Head) เป็นอะตอม เรียกว่า ฟังก์เตอร์ (Functor) และพารามิเตอร์ต่างๆ ที่ไม่ต้องกำหนดประเภทข้อมูล จำนวนพารามิเตอร์จะเรียกว่าอาร์ริตี้ (Arity)

5. ลิสต์ (List)

คือโครงสร้างข้อมูลที่ใช้เก็บข้อมูลเป็นกลุ่มเป็นโครงสร้างที่นิยามแบบเรียกซ้ำ ลิสต์ (รูปที่ 2.7)

รายการ (รูปแบบทั่วไป)	ส่วนหัว	ส่วนหาง	รายการ (รูปแบบในเชิงฟังก์ชัน)
[a]	a	[ ]	.(a,[ ])
[a,b,c]	a	[b,c]	.(a,.(b,.(c,[ ])))
[ ]	–	–	[ ]
[[a,b],c]	[a,b]	[c]	.(.(a,.(b,[ ])),.(c,[ ]))
[X|Y]	X	Y	.(X,Y)
[X,Y|Z]	X	[Y|Z]	.(X,.(Y,Z))

รูปที่ 2.7 การเรียกซ้ำของลิสต์ในภาษาโปรล็อก (บุญเสริม กิจศิริกุล, 2548)

6. ข้อเท็จจริง (Facts)

ข้อเท็จจริง คือความจริงที่เก็บอยู่ในฐานความรู้ และสามารถถามตอบกับความรู้นั้นได้ซึ่งจะอยู่ในรูปแบบเพรดดิเคต (Predicates) ตัวอย่างเช่น

female(dang). % female คือ เพรดดิเคตประกอบด้วยหนึ่งอาร์ริตี้

female(dang, ying, meow). % female คือ เพรดดิเคตประกอบด้วยสามอาร์ริตี้

มีเพรดิเคตหลายตัว ที่กำหนดไว้ในตัวภาษา เพื่อทำให้สามารถเรียกใช้งานได้อย่างสะดวก ได้ เช่น เพรดิเคต writeln('Hello') จะแสดงผลออกหน้าจอว่า Hello และจะเว้นบรรทัด 1 บรรทัด

7. กฎ (Rule)

คือข้อความที่เขียนอยู่ในรูปแบบของความจริงที่มีเงื่อนไข เช่น

eat(apple) :- happy(zaguraba).

เครื่องหมาย " :- " แปลว่า " **ถ้า** " กฎนี้หมายความว่า ถ้า happy(zaguraba) เป็นจริง eat(apple) จะเป็นจริงด้วย

8. การเรียกซ้ำ (Recursive)

เป็นเครื่องมือสำคัญของโปรล็อก เพราะภาษาโปรล็อกไม่มีคำสั่งวนซ้ำ เช่น ถ้าต้องการนิยามว่าใครเป็นบรรพบุรุษ สามารถเขียนข้อความประกาศนิยามได้ดังนี้

ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

จากตัวอย่างจะเห็นว่าพจน์ที่สองเพรดดิเคต ancestor(Z,Y) จะมีการเรียกตัวเองซ้ำอีกครั้ง

9. การประเมินค่า (Evaluation)

การประเมินค่าหรือการประมวลผลของภาษาโปรล็อกนั้นเมื่อผู้ใช้มีการสอบถามข้อมูล โปรแกรมก็จะไปค้นหาข้อเท็จจริงลักษณะการค้นหาจะใช้ Search Tree โดยคำถามของผู้ใช้จะเป็นโหนดรากของ Tree จากโหนดรากจะค้นหาคำตอบลงลึกไปเรื่อย ๆ จนถึงโหนดใบ เช่น เมื่อต้องการสอบถามจากโปรแกรมต่อไปนี้ว่า sally เป็นพี่น้องกับ erica หรือไม่ การสอบถามจะอยู่ในลักษณะ ?- sibling(sally, erica). และคำตอบที่ได้คือ yes

```
sibling(X,Y) :- parent(Z,X), parent(Z,Y).
father(X,Y) :- parent(X,Y), male(X).
mother(X,Y) :- parent(X,Y), female(X).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
mother(trude, sally).
father(tom, sally).
father(tom, erica).
father(mike, tom).
male(tom).
female(trude).
male(mike).
```

ในการทำงานโปรแกรมจะไปหาคำตอบนี้เทียบกับกฎ sibling(X,Y) โดยจะแทนที่ค่า X เป็น sally แทนค่า Y เป็น erica และเพรดดิเคต parent แทนค่าตามตัวแปร X และ Y ด้วยจะได้ parent(Z,sally) และ parent(Z,erica) หลังจากนั้นก็จะไปหาว่าใครเป็นพ่อแม่ของ sally และ erica จะได้เพรดดิเคต father เป็น father(tom,sally) และ father(tom,erica) แล้วโปรแกรมก็จะตอบ yes แสดงว่ามีข้อเท็จจริงนี้อยู่ (วิกิพีเดีย สารานุกรมเสรี, 2554)

10. นิเสธ (Negation)

ในการทำงานส่วนใหญ่เราต้องการให้ความสัมพันธ์นั้นเป็นจริง แต่ในบางกรณีเราต้องการความสัมพันธ์ที่ไม่เป็นจริง เราจะใช้สัญลักษณ์นิเสธ \+ หรือ not

2.3.2 ระบบโปรล็อก

ซอฟต์แวร์ที่ใช้ในการพัฒนาโปรแกรมโปรล็อกประกอบด้วย (วิกิพีเดีย สารานุกรมเสรี, 2554)

- Turbo Prolog โดย Borland, เลิกสนับสนุนแล้ว
- Open Prolog (http://www.cs.tcd.ie/open-prolog/)
- Ciao Prolog (http://www.clip.dia.fi.upm.es/Software/Ciao)
- GNU Prolog (http://gnu-prolog.inria.fr)
- YAP Prolog (http://www.ncc.up.pt/~vsc/Yap)
- SWI Prolog (http://www.swi-prolog.org)
- Visual Prolog (http://www.visual-prolog.com)
- SICStus Prolog (http://www.sics.se/sicstus/)
- Amzi! Prolog (http://www.amzi.com/)
- B-Prolog (http://www.probp.com/)
- TuProlog (http://tuprolog.sourceforge.net/)
- XSB (http://xsb.sourceforge.net/)
- Trinc Prolog (http://www.trinc-prolog.com)
- Strawberry Prolog (http://www.dobrev.com/)

2.4 โปรแกรมอีคลิปส์ (ECLiPSe)

เป็นโปรแกรมที่เหมาะกับการโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ พัฒนาโดยชาวยุโรปในปี 1991 (Krzysztof and Mark, 2007) ปัจจุบันเป็นโอเพนซอฟต์แวร์โอเพนซอร์สที่ผู้ใช้สามารถ

ดาวน์โหลดได้จาก http://eclipseclp.org/ ความสามารถของโปรแกรมจะง่ายต่อการเขียนรีเคอชัน มีความสามารถในการย้อนกลับ และที่สำคัญมีไลบรารีและเพรดิเคตให้ใช้อย่างสะดวก ไลบรารีที่สำคัญในการวิจัยครั้งนี้ คือ ไลบรารีไอซีและไลบรารีเอ็สดี ซึ่งทั้งสองไลบรารีนี้เป็นไลบรารีที่เหมาะกับการแก้โจทย์สมการทางคณิตศาสตร์ และมีเพรดิเคตพิเศษให้ใช้อย่างสะดวก ดังนี้

1. เพรดิเคต foreach

เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อดึงค่าที่อยู่ในลิสต์ออกมาทีละตัว และจะหยุดทำงานเมื่อลิสต์นั้นเป็นลิสต์ว่าง โดยมีตัวอย่างการใช้งานดังนี้

```
?- (    foreach(X, [a, b, c] )    do    writeln(X)    ).
```

จากการสอบถามดังกล่าวโดยเพรดิเคต foreach จะทำให้ได้ผลลัพธ์ คือ

```
a
b
c
```

2. เพรดิเคต count

เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อนับค่า โดยเพรดิเคตนี้จะต้องใส่อาร์กิวเมนต์ 3 ตัว อาร์กิวเมนต์แรกใส่ตัวแปร อาร์กิวเมนต์ที่สองใส่ค่าเริ่มต้น อาร์กิวเมนต์ที่สามใส่ค่าสิ้นสุด โดยมีตัวอย่างการใช้งานดังนี้

```
?- (    count(I, 1, 3)    do    writeln(I)    ).
```

จากการสอบถามดังกล่าวโดยเพรดิเคต count จะทำให้ได้ผลลัพธ์ คือ

```
1
2
3
```

3. เพรดิเคต fromto

เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อดึงค่าออกจากลิสต์หรือเพิ่มค่าเข้าไปในลิสต์ โดยเพรดิเคตนี้ถ้าใช้ร่วมกับ foreach จะทำหน้าที่เป็นการเพิ่มค่าเข้าไปในลิสต์ เช่น

```
?- ( foreach(X,[a, b, c]),   fromto(R, S1, S0, [ ])   do
        ( S1 = [ X+1 | S0] )
    ),   writeln(R).
```

จากการสอบถามดังกล่าว เพรดิเคต fromto จะทำหน้าที่เพิ่มค่า และจะได้ผลลัพธ์ คือ

```
[a + 1, b + 1, c + 1]
```

4. เพรดิเคต length

เป็นเพรดิเคตที่ทำหน้าที่หาความยาวของลิสต์ โดยเพรดิเคตนี้จะต้องใส่อาร์กิวเมนต์ 2 ตัว อาร์กิวเมนต์แรกใส่ลิสต์ที่ต้องการหาความยาว อาร์กิวเมนต์ที่สองใส่ตัวแปรเพื่อเก็บค่าของความยาว โดยจะมีการใช้ดังนี้

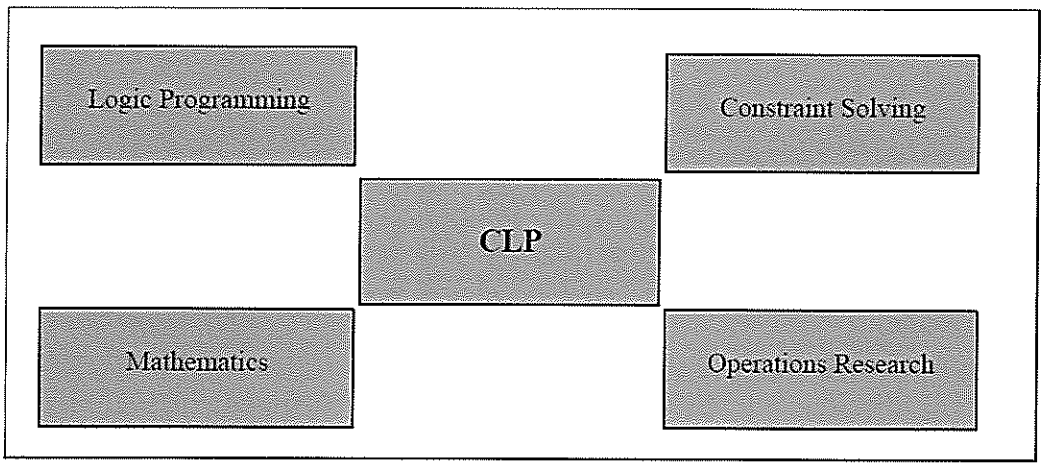
```
?- length([a, b, c], L).
```

จากการสอบถามดังกล่าวโดยเพรดิเคตเล้งจะได้ผลลัพธ์ คือ ขนาดของลิสต์เป็น 3

เพรดดิเคตข้างต้นเป็นเพรดิเคตที่ได้มีการเรียกใช้บ่อยในการวิจัยนี้ และช่วยให้การเขียนโปรแกรมสะดวกขึ้นและทำให้จำนวนบรรทัดของโค้ดนั้นน้อยลงด้วย

2.5 การโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (Constraint Logic Programming)

เป็นการเขียนโปรแกรมเหมือนโปรแกรมเชิงตรรกะปกติทั่วไปแต่ได้เพิ่มเทคนิคโดยการใส่เงื่อนไขบังคับเข้าไป โปรแกรมเชิงตรรกะปกติทั่วไปจะใช้วิธีแบบวนซ้ำด้วย Recursion แต่ใน CLP จะมี Special Predicate ซึ่งเป็น Built-in อยู่ภายในซึ่งจะทำให้การเขียนโปรแกรมสะดวกยิ่งขึ้น พื้นฐานแนวคิดของการโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับเป็นลักษณะสหสาขา ดังแสดงในรูปที่ 2.8



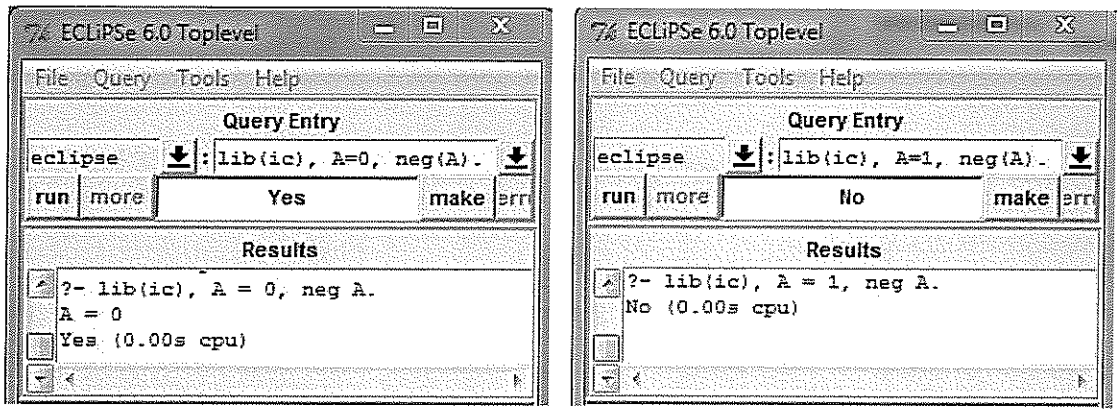
รูปที่ 2.8 เทคนิคที่อยู่เบื้องหลังของ CLP (Simonis, 2008)

ในการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ จะมีรูปแบบการใช้เงื่อนไขในแบบต่างๆ ดังนี้

2.5.1 เงื่อนไขบังคับเชิงตรรกะ (Boolean Constraints)

เป็นเงื่อนไขที่แสดงถึงการตัดสินใจแบบตรรกะ (Logic) ว่าข้อความหรือนิพจน์นั้นเป็นจริงหรือเท็จ และข้อมูลแบบตรรกศาสตร์จะมีตัวแปร 2 ค่า คือ true หรือ false โดยค่าของ false จะมีค่าเป็น 0 ส่วน true จะมีค่าเป็น 1 (หรือค่าจำนวนเลขบวกอื่นๆที่ไม่ใช่ 0) และจะมีตัวดำเนินการ คือ

1) นิเสธ (Negation) คือ ตัวดำเนินการที่จะทำการกลับค่าความจริงของตัวถูกดำเนินการไปเป็นตรงกันข้าม โดยจะมีสัญลักษณ์เป็น \+ เช่น \+true นิเสธของ true ก็คือ false ตัวอย่างการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ โดยการใช้ตัวดำเนินการนิเสธหรือใช้เพรดิเคต neg ในลาบรารี ic แสดงได้ดังรูปที่ 2.9



รูปที่ 2.9 การใช้ตัวดำเนินการนิเสธในการเขียนแบบ CLP

รูปทางด้านซ้ายมือ เป็นการสอบถามโดยกำหนดตัวแปร A ให้มีค่าเป็น 0 แทนความหมายของ false หรือค่าที่เป็นเท็จ แล้วใช้เพรดิเคตที่ชื่อ neg ซึ่งเป็นเพรดิเคตที่อยู่ใน library ชื่อ ic โดยเพรดิเคตนี้ทำหน้าที่เป็นตัวดำเนินการนิเสธของตัวแปร A ซึ่งผลจากการสอบถามจะให้ผลลัพธ์เป็น Yes หมายถึงค่าที่เป็นจริง

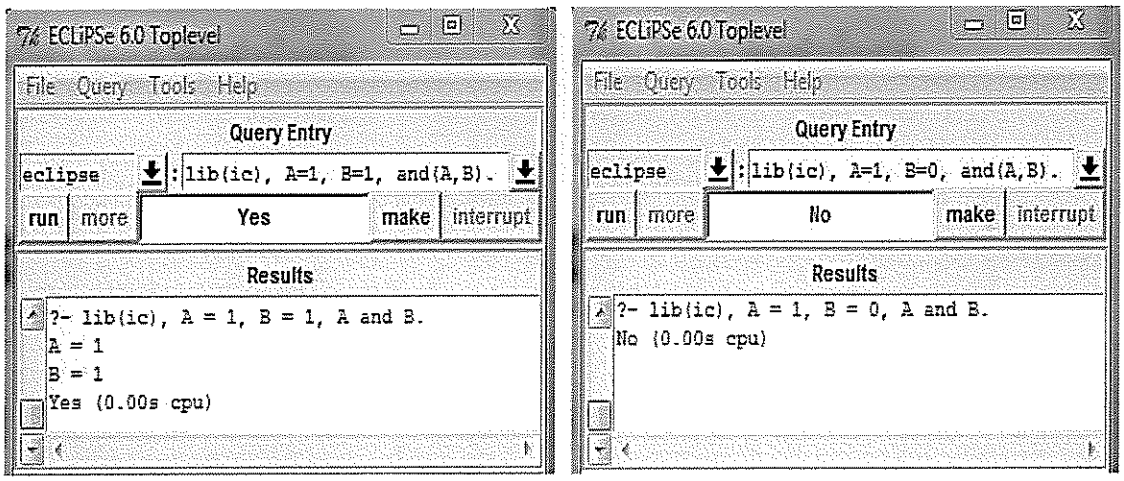
รูปทางด้านขวามือ เป็นการสอบถามโดยกำหนดตัวแปร A ให้มีค่าเป็น 1 แทนความหมายของ true หรือค่าที่เป็นจริง แล้วใช้เพรดิเคตที่ชื่อ neg เพื่อทำการนิเสธค่าของ A และผลที่ได้จากการสอบถามจะให้ผลลัพธ์เป็น No หมายถึงค่าที่เป็นเท็จ

2) ประพจน์เชื่อม (Conjunction) คือ ตัวดำเนินการที่จะทำการเชื่อมสองนิพจน์เข้าด้วยกันโดยจะมีค่าเป็นจริงก็ต่อเมื่อทั้งสองนิพจน์นั้นมีค่าเป็นจริง โดยจะมีสัญลักษณ์เป็น $\wedge$ หรือ AND ในกรณีที่มีการดำเนินการเชื่อมสองนิพจน์จะมีค่าความเป็นจริง ดังตารางที่ 2.2

ตารางที่ 2.2 การดำเนินการเชื่อมสองนิพจน์ (Conjunction)

A	B	A ∧ B
False	False	False
False	True	False
True	False	False
True	True	True

ตัวอย่างการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ โดยการใช้ตัวดำเนินการ “AND” แสดงได้ดังรูปที่ 2.10



รูปที่ 2.10 การใช้ตัวดำเนินการ “AND” ในการเขียนแบบ CLP

รูปทางด้านซ้ายมือ เป็นการสอบถามโดยกำหนดตัวแปร A และตัวแปร B ให้มีค่าเป็น 1 แล้วใช้เพรดิเคตที่ชื่อ and ซึ่งเป็นเพรดิเคตที่อยู่ใน library ชื่อ ic (ดังนั้นก่อนใช้เพรดิเคต and จะต้องใช้คำสั่ง lib(ic) เพื่อให้สามารถเรียกใช้เพรดิเคตต่าง ๆ ในไลบรารีนี้ได้) และเพรดิเคตนี้ทำหน้าที่เป็นตัวดำเนินการ “AND” ของตัวแปร A และ B ผลจากการสอบถามจะให้ผลลัพธ์เป็น Yes

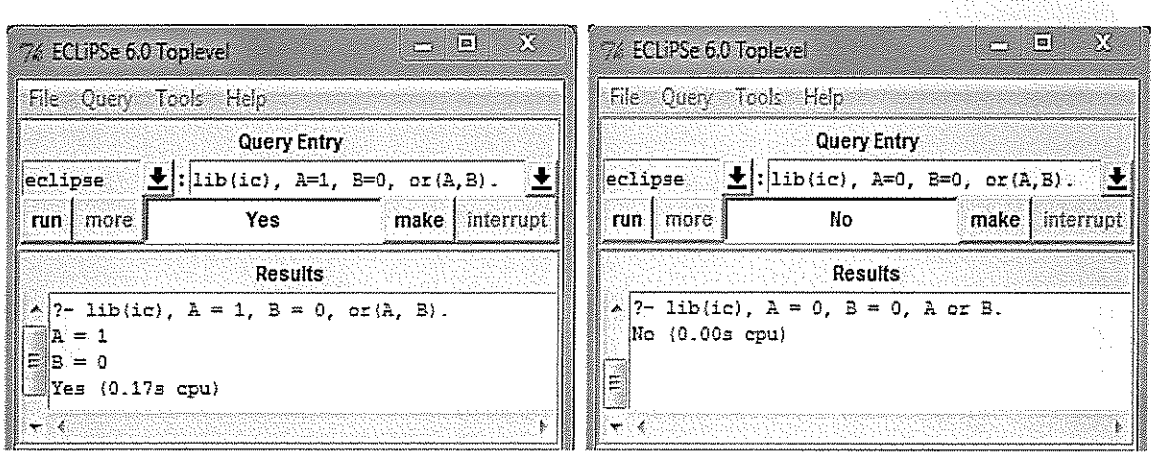
รูปทางด้านขวามือ เป็นการสอบถามโดยกำหนดตัวแปร A ให้มีค่าเป็น 1 และตัวแปร B มีค่าเป็น 0 แล้วใช้เพรดิเคตที่ชื่อ and เช่นเดียวกับการสอบถามทางด้านซ้ายมือ และผลที่ได้จากการสอบถามจะให้ผลลัพธ์เป็น No

3) ประพจน์เลือก (Disjunction) คือ ตัวดำเนินการกับนิพจน์ตรรกะที่จะให้ผลลัพธ์มีค่าเป็นจริงเมื่อบางนิพจน์มีค่าเป็นจริง โดยจะมีสัญลักษณ์เป็น ∨ หรือ OR และจะมีตารางแสดงค่าความจริง ดังตารางที่ 2.3

ตารางที่ 2.3 การดำเนินการเลือกจากสองนิพจน์

A	B	A ∨ B
False	False	False
False	True	True
True	False	True
True	True	True

ตัวอย่างการเขียนข้อความ โดยการใช้ตัวดำเนินการ “OR” แสดงได้ดังรูปที่ 2.11



รูปที่ 2.11 การใช้ตัวดำเนินการ “OR” ในการเขียนแบบ CLP

รูปทางด้านซ้ายมือ เป็นการสอบถามโดยกำหนดตัวแปร A เป็น 1 และตัวแปร B ให้มีค่าเป็น 0 แล้วใช้เพรดิเคตที่ชื่อ or ซึ่งเป็นเพรดิเคตที่อยู่ใน library ชื่อ ic และเพรดิเคตนี้ทำหน้าที่เป็นตัวดำเนินการ “OR” ของตัวแปร A และ B ซึ่งผลจากการสอบถามจะให้ผลลัพธ์เป็น Yes

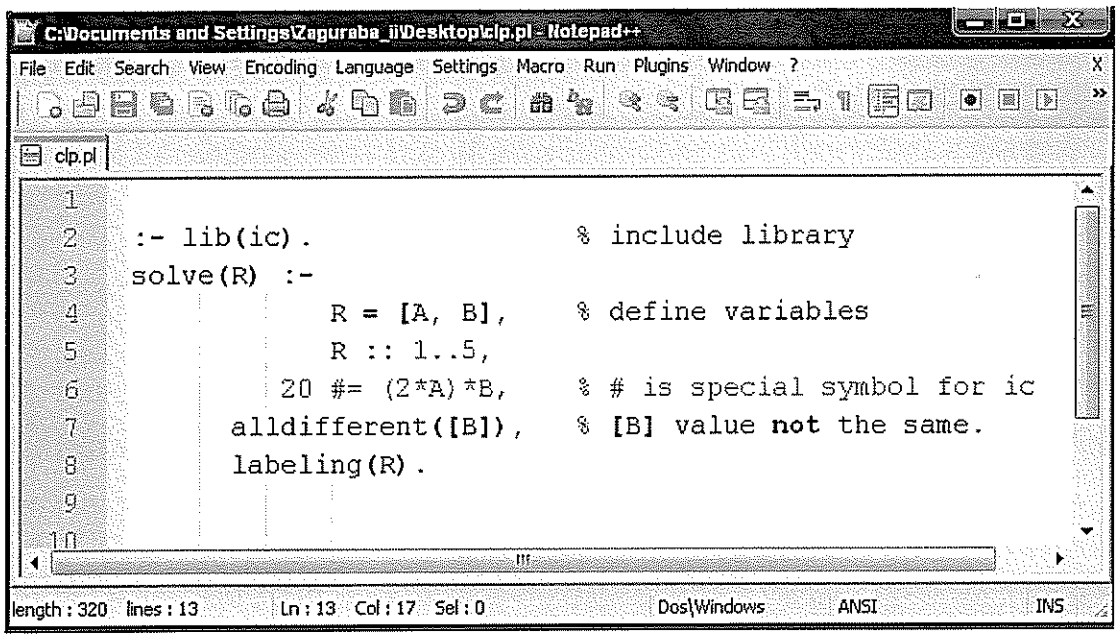
รูปทางด้านขวามือ เป็นการสอบถามโดยกำหนดตัวแปร A ให้มีค่าเป็น 1 และตัวแปร B มีค่าเป็น 0 แล้วใช้เพรดิเคตที่ชื่อ or เช่นเดียวกับการสอบถามทางด้านซ้ายมือ และผลที่ได้จากการสอบถามจะให้ผลลัพธ์เป็น No

2.5.2 เงื่อนไขบังคับเชิงเส้น (Linear Constraints)

เป็นเงื่อนไขที่แสดงถึงการหาค่าของตัวแปรที่เราต้องการในรูปแบบสมการเชิงเส้น โดยจะมีตัวดำเนินการเป็น +, -, *, / หรือตัวดำเนินการในการเปรียบเทียบคือ $<, \leq, =, \neq, \geq, >$ โดยมีตัวอย่างของสมการเชิงเส้น เช่น

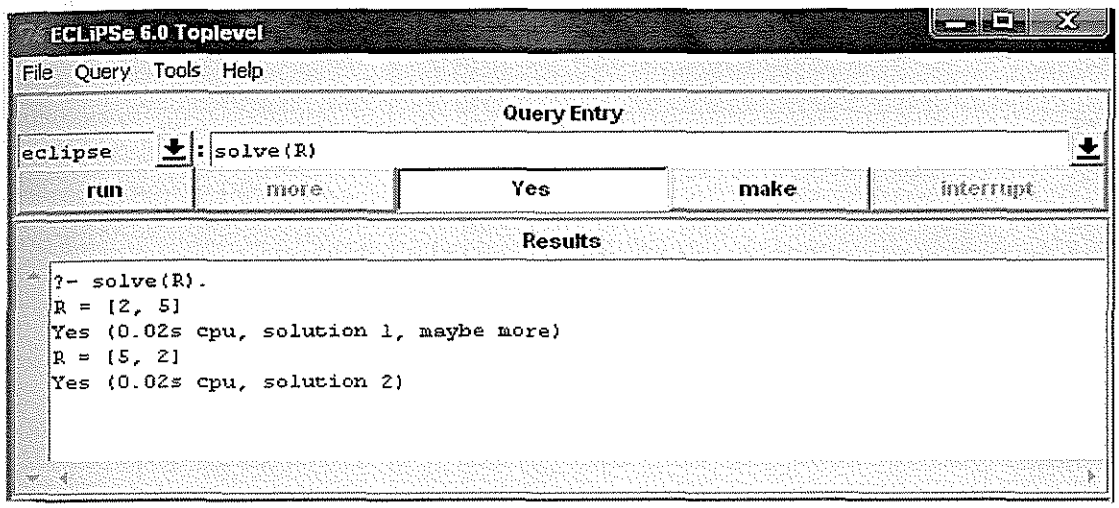
$$3a + 4b - 5c = 60*y+31-2x$$

การแก้โจทย์ปัญหาตามสมการตัวอย่างนี้ จะทำให้ได้คำตอบที่เป็นค่าตัวแปรของ a, b, c, x และ y การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (นิตยา เกิดประสพ, 2554) เพื่อแก้โจทย์ปัญหาของสมการเชิงเส้น 2AB = 20 แสดงได้ดังรูป 2.12



รูปที่ 2.12 การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ เพื่อแก้สมการเชิงเส้น 2AB = 20

ชุดคำสั่งในรูปที่ 2.12 เป็นการเขียนโปรแกรมเพื่อหาค่าของตัวแปร A และ B โดยส่วนของโค้ดจะมีรายละเอียดดังต่อไปนี้ บรรทัดที่ 2 จะเป็นการเรียกใช้ไลบรารีไอซี บรรทัดที่ 3 กำหนดชื่อเพรดิเคต solve เพื่อใช้เป็นชื่อคำสั่งเมื่อต้องการเรียกใช้โปรแกรมนี้ บรรทัดที่ 4 เป็นการกำหนดตัวแปรของ R ซึ่งจะประกอบด้วยตัวแปร A และ B ในบรรทัดที่ 5 กำหนดค่าช่วงของตัวแปร R ที่เป็นไปได้ซึ่งจะเป็นช่วงค่า 1 ถึง 5 บรรทัดที่ 6 เป็นสมการหาค่าของ A และ B โดยที่ (A*2)*B จะมีค่าเท่ากับ 20 บรรทัดที่ 7 ค่าของตัวแปร B จะไม่ซ้ำค่าเดิม และผลลัพธ์จากการเรียกใช้เพรดิเคต solve แสดงได้ดังรูปที่ 2.13



รูปที่ 2.13 ผลลัพธ์จากการสอบถามด้วยการเรียกเพรดิเคต solve เพื่อหาค่าของ A และ B จากสมการเชิงเส้น

จากผลลัพธ์จะเห็นได้ว่าในการสอบถามโดยใช้คำสั่ง solve(R). จะได้คำตอบของ R ซึ่งเป็นลิสต์ของตัวแปร A และ B สองคำตอบ คำตอบแรกเป็น A = 2 และ B = 5 และเมื่อกดปุ่ม more เพื่อหาคำตอบต่อไป คำตอบที่สองจะได้ A = 5 และ B = 2 ในการกดหาคำตอบอื่น ๆ ที่เป็นไปได้ใช้เทคนิคการค้นหาคำตอบที่เรียกว่าเทคนิคการย้อนกลับ (Backtrack)

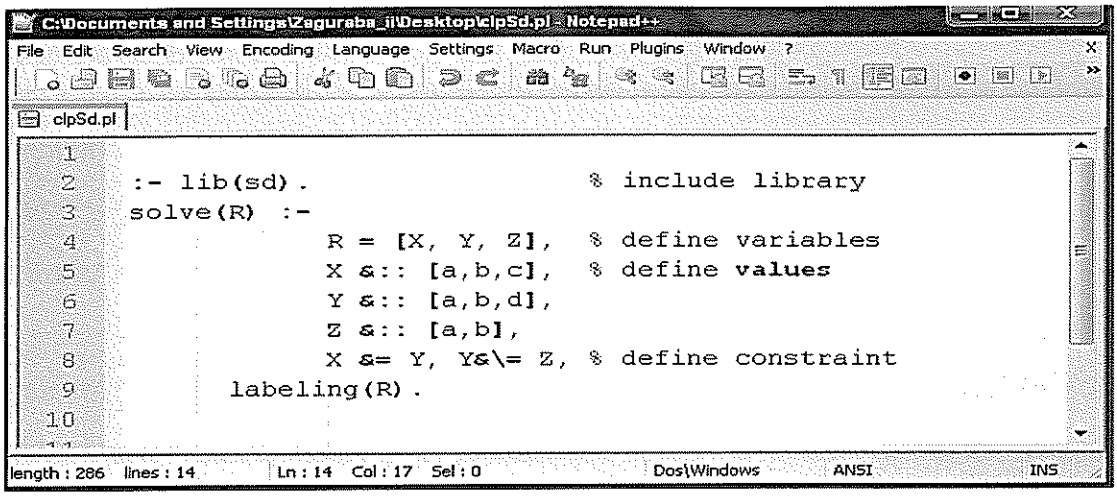
2.5.3 เงื่อนไขบังคับเชิงสัญลักษณ์ (Symbolic Constraints)

ในการใช้เงื่อนไขบังคับกับตัวแปรที่มีค่าเป็นสัญลักษณ์หรือตัวอักษรนั้นจำเป็นต้องเรียกใช้ library (sd) ซึ่งในการใช้ไลบรารีนี้จะมีการประกาศตัวแปรหรือกำหนดค่าให้ตัวแปรด้วยรูปแบบ ดังนี้

X &:: [a, b, c] % เป็นการกำหนดกลุ่มของค่าที่เป็นไปได้ให้ตัวแปร X โดยจะมีค่าที่เป็นไปได้คือ a, b, c

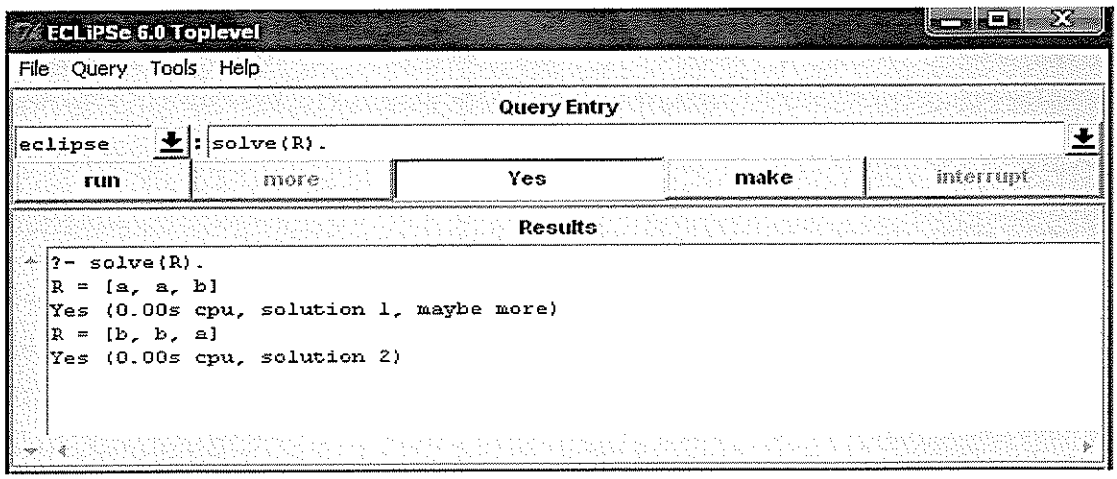
[Y, Z] &:: [a, e] % เป็นการกำหนดกลุ่มของค่าให้ตัวแปร Y และ Z โดยจะมีค่าที่เป็นไปได้คือ a, e

ในการเปรียบเทียบ จะใช้เครื่องหมาย &= มีความหมายว่า เท่ากับ และใช้เครื่องหมาย &\= มีความหมายว่าไม่เท่ากับ โดยจะมีตัวอย่างการเขียนคำสั่งดังรูปที่ 2.14



รูปที่ 2.14 การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับเชิงสัญลักษณ์

ชุดคำสั่งในรูปที่ 2.13 เป็นการเขียนโปรแกรมเพื่อหาค่า X, Y, Z โดยส่วนของโค้ดจะมีรายละเอียดดังต่อไปนี้ บรรทัดที่ 2 จะเป็นการเรียกใช้ไลบรารี sd บรรทัดที่ 3 กำหนดเพรดิเคต solve บรรทัดที่ 4 เป็นการกำหนดตัวแปรของ R ซึ่งจะประกอบด้วยตัวแปร X, Y และ Z ในบรรทัดที่ 5 กำหนดค่าที่เป็นไปได้ของตัวแปร X ซึ่งค่าที่เป็นไปได้คือ a, b, c บรรทัดที่ 6 กำหนดค่าที่เป็นไปได้ของตัวแปร Y ซึ่งค่าที่เป็นไปได้คือ a, b, d บรรทัดที่ 7 กำหนดค่าที่เป็นไปได้ของตัวแปร Z ซึ่งค่าที่เป็นไปได้คือ a, b บรรทัดที่ 8 เป็นการกำหนดเงื่อนไขบังคับ โดยค่าของ X จะต้องเท่ากับ Y และค่าของ Y จะต้องไม่เท่ากับ Z คำสั่งบรรทัดที่ 9 เป็นการสั่งค้นหาคำตอบเพื่อหาค่าของ X, Y และ Z ที่เป็นกลุ่มของตัวแปรใน R ซึ่งผลลัพธ์จากการสอบถามจะได้ดังรูปที่ 2.15



รูปที่ 2.15 ผลลัพธ์จากการสอบถามเพื่อหาค่าของ X, Y และ Z

จากผลลัพธ์จะเห็นได้ว่าในการสอบถามโดยใช้ solve(R). ซึ่งจะได้คำตอบของ R สองคำตอบ คำตอบแรกเป็น X = a, Y = a, Z = b และเมื่อกดหาคำตอบต่อไป คำตอบที่สองจะได้ X = b, Y = b, Z = a จะเห็นได้ว่าการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ จะสามารถแก้โจทย์ปัญหาทางคณิตศาสตร์ได้อย่างมีประสิทธิภาพ

2.6 งานวิจัยที่เกี่ยวข้อง

ในการวิจัยเกี่ยวกับการพัฒนาวิธีการค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ ผู้วิจัยได้ทำการศึกษาค้นคว้างานวิจัยที่มีความเกี่ยวข้องกับงานวิจัยโดยมีรายละเอียดดังนี้

Srikant, Vu และ Agrawal (1997) ได้นำเสนอแนวคิดการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ของข้อมูลโดยใช้เงื่อนไขบังคับ โดยใช้หลักการของการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ (Association Rule) และได้ใช้เทคนิคเงื่อนไขบังคับ (Constraints) เพื่อให้ผู้ใช้ (User) สามารถระบุได้ว่ากฎที่มีความสัมพันธ์หรือรูปแบบของข้อมูล (Rule) ที่ได้มานั้นจะต้องประกอบด้วยไอเท็ม (Item) ที่ผู้ใช้ระบุด้วย เพื่อให้ได้กฎตามที่ผู้ใช้ต้องการ ทั้งนี้ในการใช้เงื่อนไขบังคับเข้ามาในการทำเหมืองข้อมูลแบบค้นหากฎความสัมพันธ์ สามารถช่วยลดโครงสร้างแลตทิซ (Lattice Structure) การที่ขนาดของโครงสร้างลดลงเนื่องมาจากเทคนิคการพรุน (Prunning) และเมื่อโครงสร้างลดลง ผลที่ตามมาก็คือทรัพยากร (Memory) ที่ใช้ในการประมวลผลและระยะเวลาในการทำงานของการทำเหมืองข้อมูลแบบค้นหากฎความสัมพันธ์ก็จะลดลงเช่นกัน แต่งานวิจัยนี้ยังเป็นเพียงแค่แนวคิด โดยยังไม่มีการดำเนินการทดสอบ (Implement) จริง

Arianna Gallo, Roberto Esposito, Rosa Meo และ Marco Botta (2005) เสนอเกี่ยวกับการเพิ่มประสิทธิภาพของกฎความสัมพันธ์โดยใช้เงื่อนไขบังคับ ในงานวิจัยนี้เขาได้พบปัญหาว่าการค้นหากฎความสัมพันธ์จากฐานข้อมูลจะทำให้พบไอเท็มที่ปรากฏบ่อยที่มีลักษณะคล้ายกันอยู่มาก จึงทำให้ใช้เวลาในการประมวณผลและใช้ทรัพยากรมาก ทีมงานนี้ได้เสนออัลกอริทึมซึ่งช่วยในการสอบถามข้อมูล (Query) ในฐานข้อมูลเพื่อให้ได้กฎความสัมพันธ์ตามที่ผู้ใช้ต้องการและช่วยลดเวลาในการค้นหา

Bapliste Jeudy และ Jean-Francois Boulicaut (2002) ได้นำเสนอการทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์โดยใช้เงื่อนไขบังคับในฐานข้อมูลอุปนัย (Inductive Database) เพื่อเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์และได้ใช้เครื่องมือ Mine Rule Operator เพื่อง่ายต่อการค้นหากฎความสัมพันธ์

Trifonov และ Georgieva (2009) ได้นำเสนอการประยุกต์ใช้การทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับเพื่อหาความสัมพันธ์ของเสียงระฆัง โดยเขาได้นำเทคนิคการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับมาสร้างโปรแกรมด้วยภาษาจาวาและภาษาสอบถามข้อมูล SQL เพื่อง่าย สะดวกและรวดเร็วต่อคนที่ต้องการค้นหากฎความสัมพันธ์และผู้ใช้ไม่จำเป็นต้องมีความรู้ภาษาสอบถามข้อมูล แต่การหาความกฎสัมพันธ์ของงานวิจัยนี้ได้เฉพาะข้อมูลของเสียงระฆัง

Roberto J. และคณะ (2000) ได้นำเสนอเงื่อนไขพิเศษนอกจากเงื่อนไขการกำหนดค่าสนับสนุนต่ำสุดและค่าเชื่อมั่นต่ำสุดของการหากฎความสัมพันธ์คือเงื่อนไข minimp (Minimum Improvement) เป็นเงื่อนไขที่มีแนวคิดคล้ายการกำหนดค่าเชื่อมั่นต่ำสุดเพื่อลดความซับซ้อนของกฎความสัมพันธ์ที่ได้จากฐานข้อมูลขนาดใหญ่และให้ผลลัพธ์เป็นกฎความสัมพันธ์ที่เข้าใจง่ายแก่ผู้ใช้

จากการศึกษางานวิจัยที่เกี่ยวข้องพบว่ามีงานวิจัยจำนวนมากที่ใช้เงื่อนไขบังคับเข้ามาช่วยในการทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์ เพื่อช่วยให้ผู้ใช้งานสามารถกำหนดกฎความสัมพันธ์ที่ได้รับจากการทำเหมืองข้อมูลได้ เพราะว่าหากไม่มีเงื่อนไขบังคับในการทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์นั้นจะทำให้ได้รับกฎจำนวนมากและอาจจะทำให้ผู้ใช้งานสับสนกับจำนวนกฎที่มีมาก และมีหลายงานวิจัยที่ใช้เงื่อนไขบังคับเพื่อเพิ่มประสิทธิภาพของกฎ ทำให้กฎที่ได้รับมีความน่าเชื่อถือหรืออาจไม่ซ้ำซ้อนกัน ในงานวิจัยส่วนมากได้นำภาษาสอบถามข้อมูลมาใช้ในการทำเหมืองข้อมูลเพราะว่าในระบบฐานข้อมูลนั้นมีเครื่องมือใช้ในการวิเคราะห์กฎความสัมพันธ์ เรียกว่า Mine Rule Operator ซึ่งง่ายต่อการทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์ และบางงานวิจัยได้สร้างโปรแกรมเพื่อหากฎความสัมพันธ์และให้ผู้ใช้สามารถกำหนดสมาชิกในกฎที่ได้รับได้ แต่อย่างไรก็ตามในงานวิจัยนี้ผู้วิจัยได้พัฒนาโปรแกรมหากฎความสัมพันธ์โดยใช้ภาษาโปรล็อกและมีรูปแบบการเขียนแบบใช้เงื่อนไขบังคับ เพื่อให้ผู้ใช้สามารถกำหนดขนาดของกฎ สมาชิกของกฎและเป้าหมายของกฎได้ สาระสำคัญของงานวิจัยนี้เมื่อเปรียบเทียบกับงานวิจัยอื่นสรุปได้ดังตารางที่ 2.4 ในส่วนรายละเอียดของการออกแบบขั้นตอนต่าง ๆ อธิบายอยู่ในบทต่อไป

ตารางที่ 2.4 สรุปเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับการพัฒนาระบบเพื่อค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ

กระบวนการทำงาน	งานวิจัยที่เกี่ยวข้อง*					
	1	2	3	4	5	6
การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์						
ประเภทของฐานข้อมูล						
ฐานข้อมูลประเภท Relational Database		✓			✓	
ฐานข้อมูลประเภท Inductive Database		✓				
ฐานข้อมูลประเภท Logic Database						✓
การพัฒนา						
พัฒนาโดยใช้แนวคิดกฎข้อบังคับ (Constraint)	✓	✓	✓	✓	✓	✓
พัฒนาโดยการเพิ่มประสิทธิภาพของกฎความสัมพันธ์		✓	✓	✓		
พัฒนาโดยการให้ผู้ใช้กำหนดสมาชิกของกฎได้	✓	✓		✓	✓	✓
พัฒนาโดยการเพิ่มความเร็วในการประมวลผล	✓			✓		✓
พัฒนาโดยใช้รูปแบบของภาษา JAVA					✓	
พัฒนาโดยใช้รูปแบบของภาษา CLP						✓
พัฒนาโดยการให้ผู้ใช้กำหนดขนาดของกฎได้						✓
พัฒนาโดยการให้ผู้ใช้กำหนดเป้าหมายของกฎได้					✓	✓
ขอบเขตของการวิจัย						
วิจัยเพื่อทดสอบประสิทธิภาพ			✓	✓		✓
วิจัยเพื่อเสนอแนวคิดเท่านั้น	✓	✓				
วิจัยเพื่อทดสอบความถูกต้อง			✓	✓		✓
มีการประยุกต์ใช้กับข้อมูลจริง					✓	

*งานวิจัยที่เกี่ยวข้องประกอบด้วย

1 = Ramakrishnan Srikant และคณะ (1997)

2 = Bapliste Jeudy และคณะ (2002)

3 = Roberto J. และคณะ (2000)

4 = Arianna Gallo และคณะ (2005)

5 = Tihomir Trifonov และคณะ (2009)

6 = การพัฒนาระบบเพื่อค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (งานวิจัยของวิทยานิพนธ์ฉบับนี้)

บทที่ 3

วิธีดำเนินการวิจัย

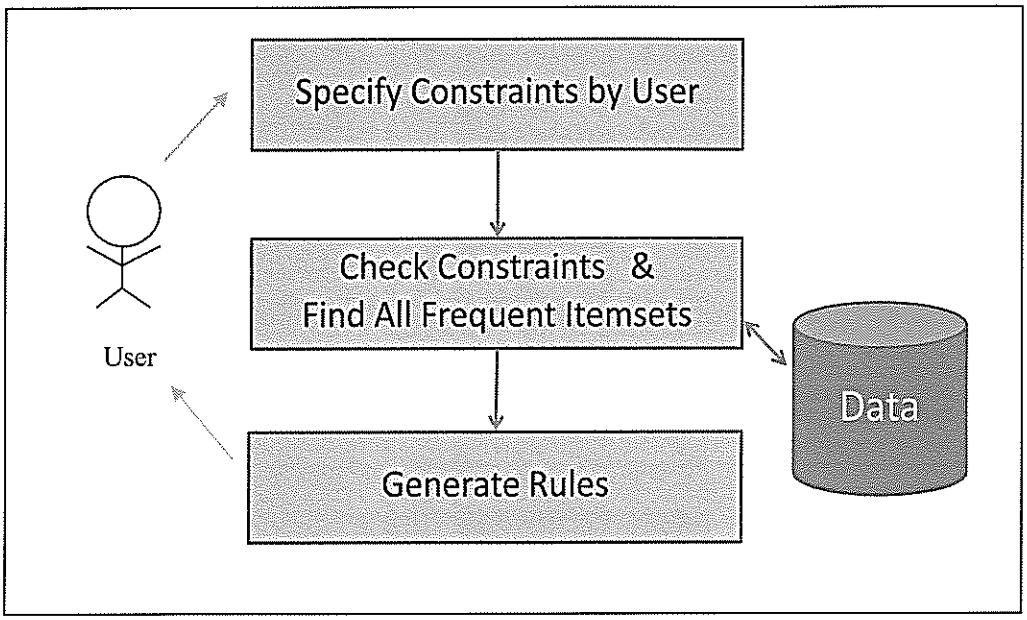
งานวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาการทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์ที่ผู้ใช้สามารถระบุส่วนประกอบและความยาวของกฎได้ โดยโปรแกรมที่พัฒนาขึ้นได้ใช้ภาษาโปรล็อก และได้ใช้รูปแบบการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ และได้นำอัลกอริทึมเอไพรออริ (Apirori) มาช่วยในการหารูปแบบของกฎความสัมพันธ์ ในบทนี้จะนำเสนอถึง วิธีการวิจัย เครื่องมือที่ใช้ในการวิจัย และกระบวนการต่างๆ ของการวิจัย โดยมีรายละเอียดดังนี้

3.1 กรอบแนวคิดของการวิจัย

แนวคิดหลักของงานวิจัยนี้คือ การปรับปรุงกระบวนการค้นหากฎความสัมพันธ์ให้สามารถผนวกเงื่อนไขบังคับที่ผู้ใช้ระบุเข้าเป็นส่วนหนึ่งของกระบวนการเพื่อลดขอบเขตการค้นหาและเพื่อให้ผลลัพธ์ที่ตรงกับความต้องการของผู้ใช้ การผนวกเงื่อนไขบังคับนี้งานวิจัยนี้ได้ออกแบบไว้ 2 แนวทาง คือ แบบที่ 1 ใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย และแบบที่ 2 ใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย

3.1.1 กรอบแนวคิดแบบที่ 1 : การใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย

จุดมุ่งหมายของโปรแกรม คือ การหากฎความสัมพันธ์ตามที่ผู้ใช้งานระบุไอเท็มที่ต้องการหรือไม่ต้องการและขนาดของกฎได้ โดยมีกรอบแนวคิดการหากฎความสัมพันธ์ ดังรูปที่ 3.1



รูปที่ 3.1 กรอบแนวคิดการหากฎความสัมพันธ์แบบใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย

จากกรอบแนวคิดการหากฎความสัมพันธ์จะประกอบไปด้วย 4 ส่วน คือ

1) ผู้ใช้ (User) ผู้ใช้มีหน้าที่ระบุค่าสนับสนุนต่ำสุด (Minimum Support) ค่าความเชื่อมั่นขั้นต่ำ (Minimum Confidence) และเงื่อนไขบังคับ (Constraints) การกำหนดสมาชิก

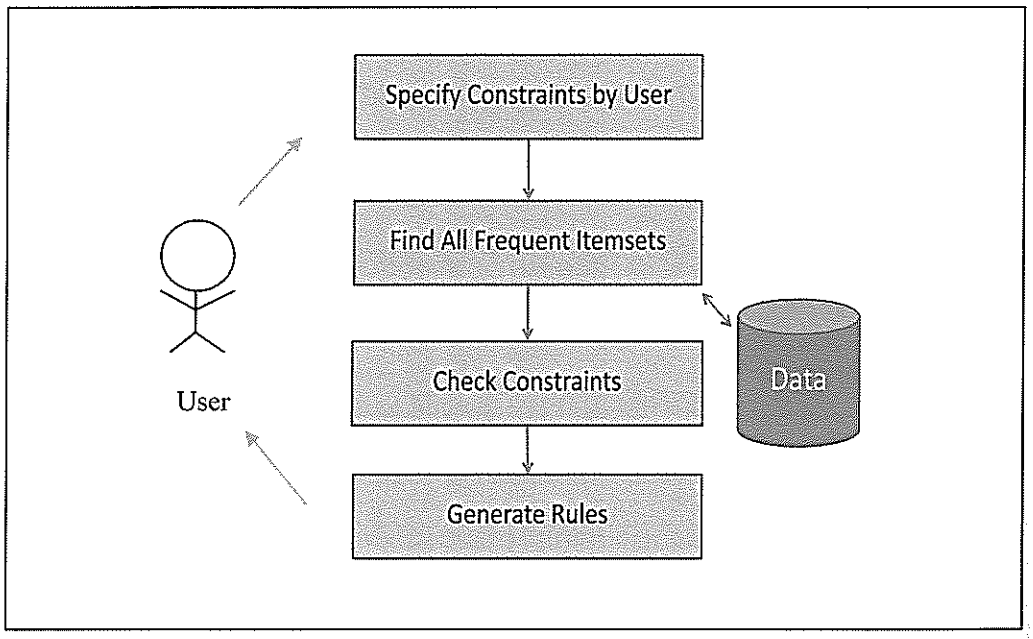
2) การระบุเงื่อนไขโดยผู้ใช้ (Specify Constraints by User) ผู้ใช้จะต้องระบุค่าสนับสนุนขั้นต่ำ อย่างเช่น ถ้าผู้ใช้ต้องการหากฎความสัมพันธ์ที่มีความถี่เท่ากับสอง หมายถึงกฎที่พบในทรานแซคชันสองทรานแซคชันจากทรานแซคชันทั้งหมด ผู้ใช้ก็จะต้องระบุค่าสนับสนุนขั้นต่ำเท่ากับสอง และผู้ใช้จะต้องระบุค่าความเชื่อมั่นขั้นต่ำ ถ้าหากผู้ใช้ต้องการเพียงแค่บางกฎความสัมพันธ์ที่ประกอบไปด้วยไอเท็มที่สนใจผู้ใช้ก็จะต้องระบุเงื่อนไขบังคับ (Constraints) ด้วย

3) การตรวจสอบเงื่อนไขบังคับและการหาไอเท็มเซตปรากฏบ่อย (Check Constraints & Find All Frequent Itemsets) ในส่วนนี้จะประกอบด้วยการใช้เงื่อนไขบังคับตามที่ผู้ใช้ได้ระบุมา (Check Constraints) เพื่อให้ได้ไอเท็มเซตตามที่ผู้ใช้ได้ระบุและหาไอเท็มเซตปรากฏบ่อยทั้งหมด (Find All Frequent Itemset) ได้แก่ไอเท็มเซตที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ

4) การสร้างกฎความสัมพันธ์ (Generate Rules) จะสร้างกฎจากไอเท็มเซตปรากฏบ่อยจากการใช้เงื่อนไขบังคับ ซึ่งการสร้างกฎความสัมพันธ์นั้น กฎจะต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

3.1.2 กรอบแนวคิดแบบที่ 2 : การใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย

ในการออกแบบกรอบแนวคิดนี้ได้ใช้แนวคิดของอัลกอริทึมเอไพรออริเป็นหลักและได้เพิ่มส่วนของเงื่อนไขบังคับเข้าไปในอัลกอริทึมด้วย (รูปที่ 3.2)



รูปที่ 3.2 กรอบแนวคิดการหากฎความสัมพันธ์แบบใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย

จากกรอบแนวคิดการหากฎความสัมพันธ์จะประกอบไปด้วย 5 ส่วน คือ

1) ผู้ใช้ (User) ผู้ใช้มีหน้าที่ระบุค่าสนับสนุนต่ำสุด (Minimum Support) ค่าความเชื่อมั่นขั้นต่ำ (Minimum Confidence) และเงื่อนไขบังคับ (Constraints) การกำหนดสมาชิก

2) การระบุเงื่อนไขบังคับโดยผู้ใช้ (Specify Constraints by User) ผู้ใช้จะต้องระบุค่าสนับสนุนขั้นต่ำ ค่าความเชื่อมั่นขั้นต่ำ และถ้าหากผู้ใช้ต้องการเพียงแค่บางกฎความสัมพันธ์ที่ประกอบไปด้วยไอเท็มที่สนใจ ผู้ใช้ก็จะต้องระบุเงื่อนไขบังคับด้วย

3) หาไอเท็มเซตปรากฏบ่อยทั้งหมด (Find All Frequent Itemset) ได้แก่ไอเท็มเซตที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ

4) ใช้เงื่อนไขบังคับตามที่ผู้ใช้ได้ระบุมา (Check Constraints) เพื่อให้ได้ไอเท็มเซตตามที่ผู้ใช้ได้ระบุ เช่น ผู้ใช้ต้องการไอเท็มเซตที่มีสมาชิกคือ {A} จากทรานแซคชัน{A, B}, {B, D}, {E, C}, {A, B, D} คำตอบที่ได้จากการใช้เงื่อนไขบังคับจะทำให้ได้ไอเท็มเซตคือ {A, B},{A, B, D} ซึ่งจะทำให้ได้ผลลัพธ์เป็นจำนวนไอเท็มเซตที่ลดลงและตรงกับความต้องการของผู้ใช้

5) การสร้างกฎความสัมพันธ์ (Generate Rules) จากขั้นตอนที่ 4 จะได้ไอเท็มเซตปรากฏบ่อยจากการใช้เงื่อนไขบังคับ ไอเท็มเซตที่ปรากฏบ่อยเหล่านี้จะถูกนำไปสร้างกฎความสัมพันธ์ ซึ่งการสร้างกฎความสัมพันธ์นั้นกฎจะต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

3.2 การออกแบบอัลกอริทึม

จากกรอบแนวคิดของงานวิจัยทั้ง 2 รูปแบบ ผู้วิจัยได้พัฒนาอัลกอริทึมของทั้งสองกรอบแนวคิด โดยกรอบแนวคิดในแบบแรกได้พัฒนาเป็นอัลกอริทึม ACIF (Association Rule Discovery With Constraints In Frequent Itemset Mining) กรอบแนวคิดในแบบที่สองได้พัฒนาเป็นอัลกอริทึม ACAF (Association Rule Discovery With Constraints After Frequent Itemset Mining) ในการทดสอบการทำงานของอัลกอริทึมทั้งสองจะเปรียบเทียบกับอัลกอริทึมเอไพรออริ (Apriori) ซึ่งเป็นอัลกอริทึมมาตรฐานในการค้นหากฎความสัมพันธ์ รายละเอียดของอัลกอริทึม Apriori, ACIF และ ACAF ปรากฏหัวข้อ 3.2.1, 3.2.2, และ 3.2.3 ตามลำดับ

3.2.1 อัลกอริทึมเอไพรออริ (Apriori)

ในงานวิจัยนี้ใช้อัลกอริทึมเอไพรออริ (Agrawal and Srikant, 1994) เป็นรูปแบบพื้นฐานในการเปรียบเทียบประสิทธิภาพการค้นหาและการแสดงผลลัพธ์ อัลกอริทึมมีการสร้างไอเท็มเซตปรากฏบ่อยดังรูปที่ 3.3, 3.4

Algorithm Apriori

//Input : Database D , Minimum_support.

//Output : L, frequent itemsets in D.

(1) L_1 = find_ frequent_1itemset(D)

(2) for($k = 2; L_{k-1} \neq \emptyset; k$++){

(3) C_k = apriori_gen(L_{k-1}, Minimum_support);

(4) for each transaction $t \in D$ { // scan D for counts

(5) C_1 = subset(C_k, t)

(6) for each cadidate $c \in C_1$ {

(7) c.count++ }

(8) }

(9) $L_k = \{c \in C_1 \mid c.count \geq$ Minimum_support$\}$

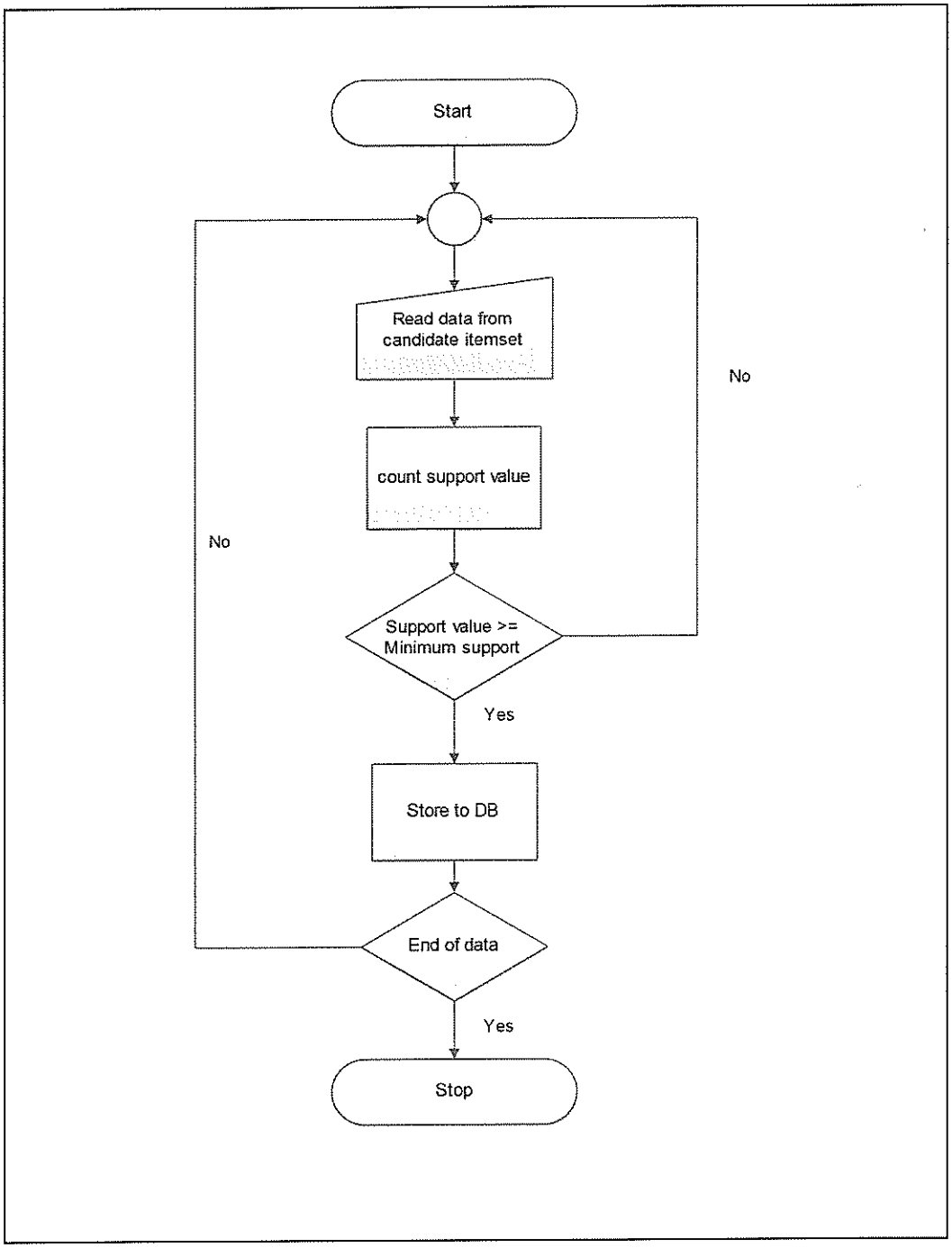
(10) }

(11) return $\cup_k L_k$

รูปที่ 3.3 การสร้างไอเท็มเซตปรากฏบ่อยในอัลกอริทึม Apriori

จากรูปมีขั้นตอนการทำงานดังนี้ อ่านข้อมูลจากฐานข้อมูลและทำการนับค่าสนับสนุนของแต่ละไอเท็ม จากนั้นนำไอเท็มที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ (Minimum Support) มาสร้างเป็นไอเท็มเซตปรากฏบ่อยขนาดหนึ่งไอเท็ม (บรรทัดที่ 1) ในการทำงานรอบที่ k (โดย k จะเริ่มจาก 2) จะพิจารณาจากรอบที่ k-1 โดยการนำข้อมูลในไอเท็มเซตปรากฏบ่อยขนาด k-1 มาสร้างเป็นไอเท็มเซตขนาด k โดยใช้ฟังก์ชัน apriori_gen เป็นตัวสร้าง (บรรทัดที่ 3) หลังจากนั้นอ่านข้อมูลจากฐานข้อมูลและนับค่าสนับสนุนของไอเท็มเซตนั้นจาก

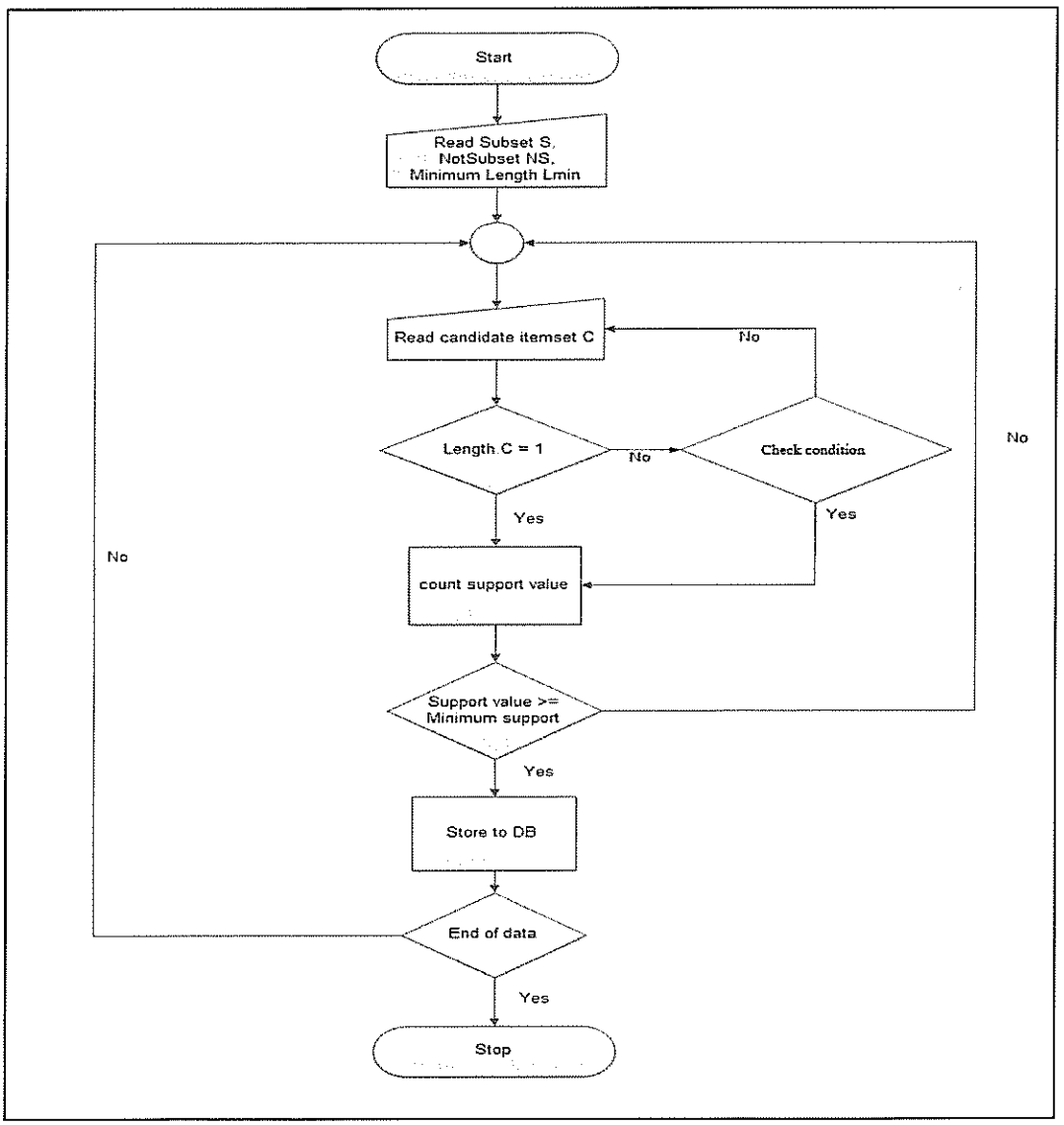
ทรานแซคชันทั้งหมด จากนั้นนำไอเท็มเซตที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำมาสร้างเป็นไอเท็มเซตปรากฏบ่อยขนาด k (บรรทัดที่ 9) และจะสิ้นสุดลงเมื่อไม่สามารถสร้างไอเท็มเซตปรากฏบ่อยต่อไปได้ ขั้นตอนหล่านี้แสดงเป็น Flow chart ได้ดังรูปที่ 3.4



รูปที่ 3.4 Flow chart แสดงขั้นตอนการสร้างไอเท็มเซตปรากฏบ่อยของอัลกอริทึม Apriori

3.2.2 อัลกอริทึม ACIF (Association Rule Discovery With Constraints In Frequent Itemset Mining)

อัลกอริทึม ACIF ที่ผู้วิจัยได้ออกแบบขึ้นนี้มีการทำงานส่วนใหญ่เหมือนอัลกอริทึมเอไพรออริแต่ได้เพิ่มฟังก์ชันการรับเงื่อนไขจากผู้ใช้เข้าไป โดยจะมีการใช้เงื่อนไขในส่วนของการหาไอเท็มเซตปรากฏบ่อย เพื่อให้ได้กฎความสัมพันธ์ตามที่ผู้ใช้กำหนดสมาชิกหรือขนาดของกฎความสัมพันธ์ แต่วิธีการนี้จะมีข้อเสียคืออาจจะไม่ได้รับกฎความสัมพันธ์บางกฎ ขั้นตอนการสร้างไอเท็มเซตปรากฏบ่อยแสดงได้ดังรูปที่ 3.5 และ 3.6



รูปที่ 3.5 Flow chant แสดงขั้นตอนการสร้างไอเท็มเซตปรากฏบ่อยของอัลกอริทึม ACIF

Algorithm ACIF

//Input : Database D,Length, Subset, NotSubset, Minimum_support.

//Output : L, frequent itemsets in D.

(1) L_1 = find_ frequent_1itemset(D)

(2) for($k = 2; L_{k-1} \neq \emptyset; k++$){

(3) C_k = apriori_gen(L_{k-1}, Minimum_support);

(4) for each transaction $t \in D$ { // scan D for counts

(5) C_1 = subset(C_k, t)

(6) for each cadidate $c \in C_1$ {

(7) c.count++

(8) }

(9) C_2 = checkcondition(Length, Subset, NotSubset, C_1)

(10) $L_k = \{c \in C_2 \mid c.count \geq$ Minimum_support$\}$

(11) } }

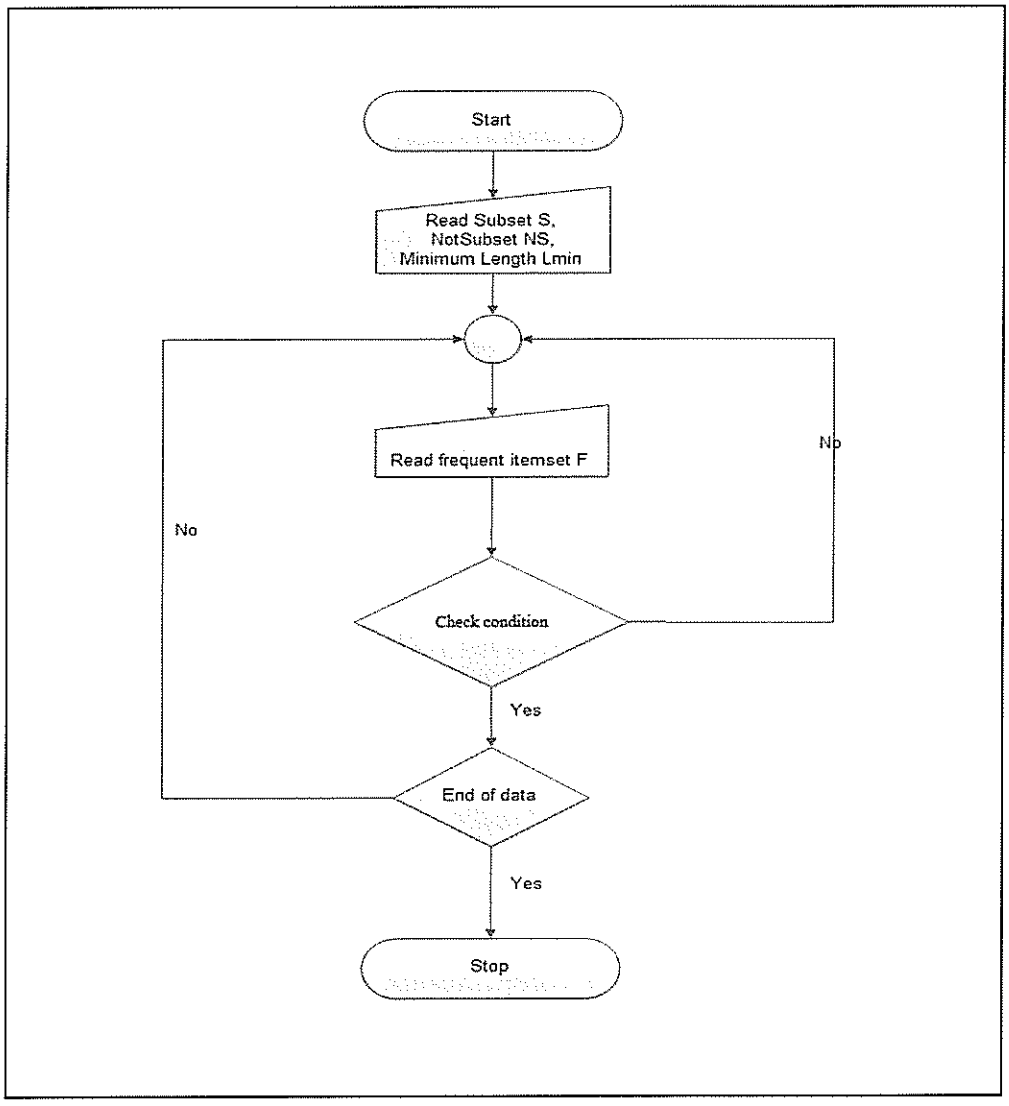
(12) return $\cup_k L_k$

รูปที่ 3.6 การสร้างไอเท็มเซตปรากฏบ่อยของอัลกอริทึม ACIF

จากรูปที่ 3.6 จะเห็นได้ว่าอัลกอริทึม ACIF มีขั้นตอนการทำงานเหมือนกับเอไพรออริแบบดั้งเดิมเพียงแต่ว่ามีส่วนที่เช็คเงื่อนไขเพิ่มขึ้นมา (บรรทัดที่ 9) เป็นการใช้เงื่อนไขบังคับโดยมีการเช็คขนาดและหาสมาชิกที่ผู้ใช้กำหนดของแคนดิเดตไอเท็มเซต หลังจากที่ได้ไอเท็มเซตปรากฏบ่อยแล้วก็จะถูกนำไปสร้างกฎความสัมพันธ์ตามปกติ

3.2.3 อัลกอริทึม ACAF (Association Rule Discovery With Constraints After Frequent Itemset Mining)

อัลกอริทึม ACAF จะมีการใช้เงื่อนไขหลังจากการหาไอเท็มปรากฏบ่อยทั้งหมด เพื่อให้ผู้ใช้กำหนดไอเท็มที่เป็นสมาชิกและขนาดของกฎความสัมพันธ์ได้และจะได้จำนวนกฎที่ครบถ้วน มีการทำงานของอัลกอริทึมดังรูปที่ 3.7 และ 3.8



รูปที่ 3.7 การทำงานของอัลกอริทึม ACAF ส่วนของการเช็คเงื่อนไข

Algorithm ACAF

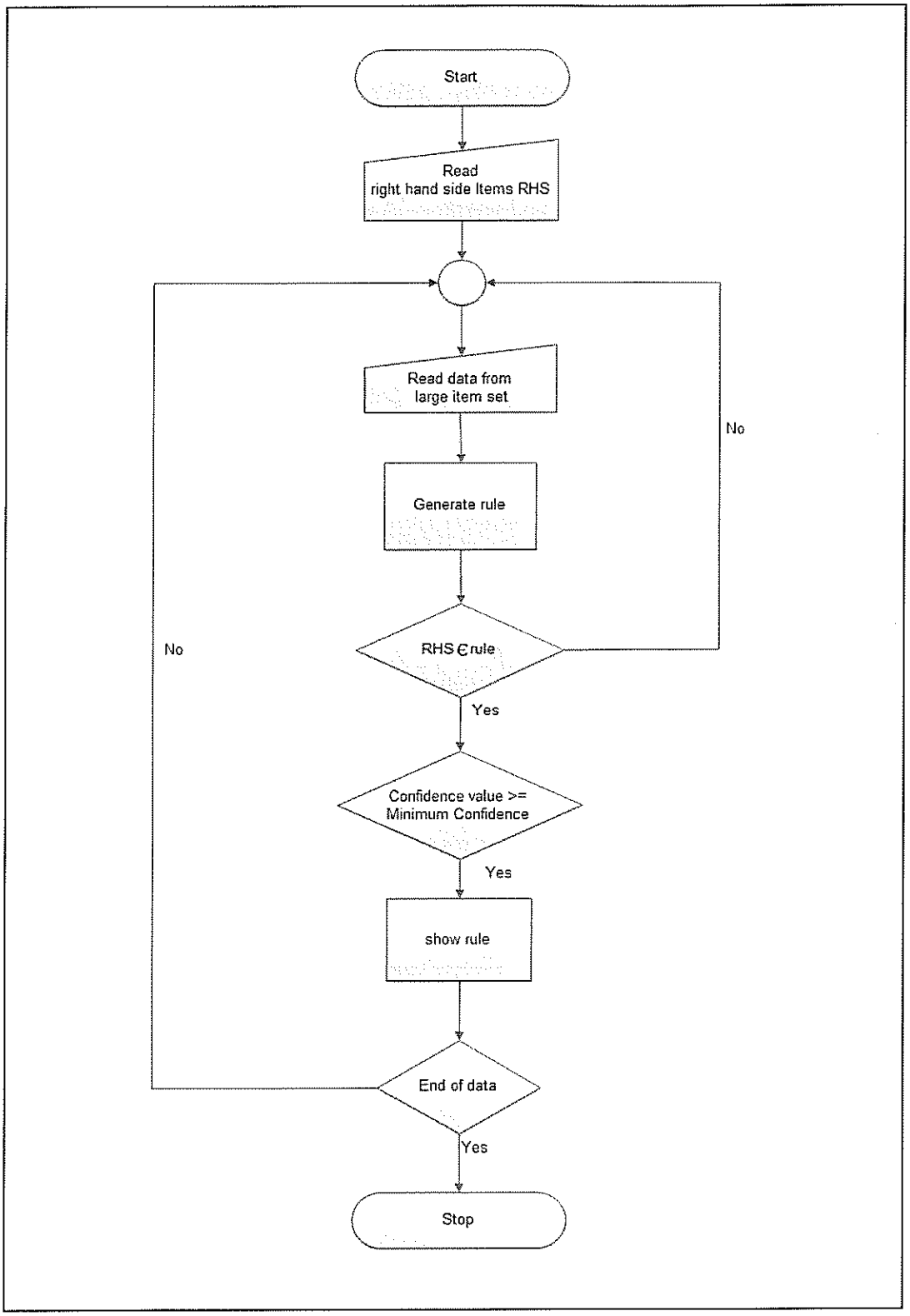
//Input : Length, Subset, NotSubset, Minimum_support, Min_conf : Minimum confident, RHS : right hand side Items.

//Output : Rules.

```
(1)   L = find_all_frequent_itemset(Minimum_support)
(2)   For each l ∈ L  {      // l = frequent_itemset
(3)      If  l.length > 1 {
(4)           Lk = checkcondition(Length, Subset, NotSubset, L)
(5)           }
(6)   }
(7)   Rules = generate_rule(Lk, Min_conf, RHS)
(8)   return Rules
```

รูปที่ 3.8 อัลกอริทึม ACAF ในการสร้างไอเท็มเซตปรากฏบ่อยและกฎความสัมพันธ์

ในขั้นตอนการทำงานของอัลกอริทึม ในบรรทัดแรกจะเป็นการสร้างไอเท็มเซตปรากฏบ่อยทั้งหมดซึ่งมีการทำงานเหมือนอัลกอริทึมเอไพรออริ (รูปที่ 3.3) หลังจากนั้นจะได้ไอเท็มเซตปรากฏบ่อยโดยที่ไอเท็มเซตนั้นจะต้องมีสมาชิกมากกว่า 1 และจะถูกนำไปตรวจสอบหาขนาดและสมาชิกของไอเท็มปรากฏบ่อย (บรรทัดที่ 4) และขั้นตอนต่อไปเป็นการสร้างกฎจากไอเท็มปรากฏบ่อยที่ผ่านการตรวจสอบเงื่อนไขและจะต้องมีค่ามากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำโดยมีขั้นตอนการสร้างกฎความสัมพันธ์ดังรูปที่ 3.9 และ 3.10



รูปที่ 3.9 ขั้นตอนการสร้างกฎความสัมพันธ์ของอัลกอริทึม ACAF

```
Procedure  generate_rule(L_k : frequent-items, Min_conf : Minimum
confident, RHS : right hand side Items);

(1)   For each l ∈ L_k        // l is frequent-itemset.
(2)       k = | l |          //   size of frequent itemset
(3)       m = |H_m|          //   size of right hand side Items
(4)       For each h_m+1 ∈ H_m+1 {
(5)         If h_m+1 = RHS  {
(6)             conf = σ(f_k) / σ (f_k – h_m+1) ;
(7)             If conf >=  Min_conf {
(8)                 Rule = rule(f_k – h_m+1) ⇒ h_m+1
(9)             } Else
(10)                delete h_m+1 from H_m+1
(11)        }
(12)      }
(13)  return Rule
```

รูปที่ 3.10 การสร้างกฎความสัมพันธ์

3.3 การพัฒนาและการใช้งานโปรแกรม

เนื้อหาในส่วนนี้จะอธิบายขั้นตอนการทำงานของอัลกอริทึมค้นหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับ เพื่อสร้างความเข้าใจการทำงานของอัลกอริทึมมากขึ้น เนื่องจากอัลกอริทึม ACIF และ ACAF มีรูปแบบข้อมูล ขั้นตอนการสร้างกฎความสัมพันธ์ และวิธีการสอบถามเพื่อหากฎ

ความสัมพันธ์ที่คล้ายกัน ดังนั้นเนื้อหาในส่วนนี้จะอธิบายเฉพาะการทำงานของโปรแกรม ACAF โดยใช้ตัวอย่างข้อมูลขนาดเล็กดังรายละเอียดต่อไปนี้

ตัวอย่าง ณ ร้านขายของแห่งหนึ่งมีการบันทึกข้อมูลการซื้อของของลูกค้าทุกวัน พอใกล้จะถึงวันปีใหม่ผู้จัดการร้านอยากจะจัดโปรโมชันลดราคาสินค้า โดยที่ผู้จัดการต้องการขายผ้าอ้อมซึ่งผ้าอ้อมมียอดขายน้อย เพื่อจะกระตุ้นยอดขายผ้าอ้อมและสินค้าอื่นที่ขายได้พร้อมผ้าอ้อมไม่เกิน 2 ชิ้นและสินค้าอื่นนั้นจะต้องไม่เป็นโค้กเพราะว่าราคาโค้กมีราคาสูงอยู่แล้ว พนักงานด้านจัดทำโปรโมชันจึงได้ใช้การทำเหมืองข้อมูลเพื่อหาสินค้าอื่นที่ไม่เกิน 2 ชิ้นและไม่เป็นโค้ก พบว่าใช้เวลาในการหาสินค้านั้นนานมากและได้กฎมากมาย จึงได้เปลี่ยนมาใช้อัลกอริทึมเอคาฟ (ACAF) ซึ่งมีวิธีการหาดังนี้

ตารางที่ 3.1 ตัวอย่างลูกค้าที่ซื้อของ 4 ราย

TID	**Items**
001	Eggs, Coke, Milk
002	Bread, Coke, Diaper
003	Eggs, Bread, Coke, Diaper
004	Bread, Diaper

ในขั้นตอนแรกจะเป็นการหาไอเท็มที่พบบ่อยจากข้อมูล (ตารางที่ 3.1) โดยกำหนดค่าสนับสนุนต่ำสุดเป็น 2 ซึ่งจะได้ 9 ไอเท็มเซตปรากฏบ่อยทั้งหมด คือ [bread, coke], [bread, diaper], [bread, eggs], [coke, diaper], [coke, eggs], [diaper, eggs], [bread, coke, diaper], [bread, coke, eggs], [coke, diaper, eggs]

ขั้นตอนที่สองเป็นการใช้เงื่อนไขข้อบังคับจากผู้ใช้โดยจากโจทย์ผู้จัดการต้องการขนาดของกฎที่ไม่เกิน 2 ไอเท็ม โปรแกรมจึงทำให้ไอเท็มเซตปรากฏบ่อยทั้งหมดลดลงเหลือ 6 เซต คือ [bread, coke], [bread, diaper], [bread, eggs], [coke, diaper], [coke, eggs], [diaper, eggs] และผู้จัดการต้องการกฎที่มี diaper และไม่ต้องการ coke เป็นสมาชิกของกฎโปรแกรมจึงทำให้ไอเท็มเซตปรากฏบ่อยทั้งหมดลดลงเหลือ 2 เซต คือ [bread, diaper], [diaper, eggs]

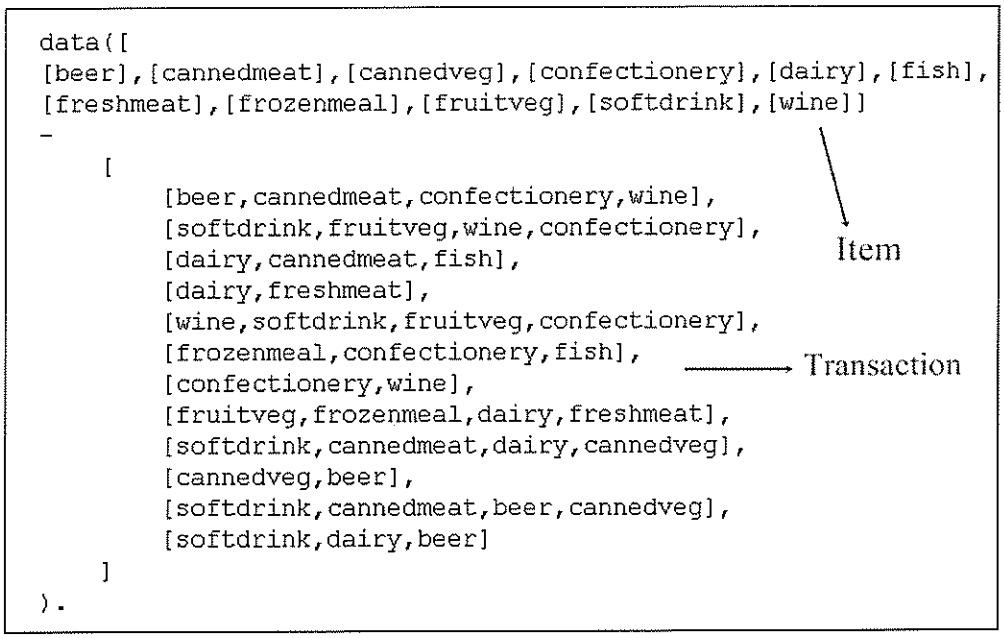
ขั้นตอนสุดท้ายเป็นการสร้างกฎจากไอเท็มปรากฏบ่อยโดยกำหนดค่าเชื่อมั่นต่ำสุดเป็น 100% จากไอเท็มเซตปรากฏบ่อย 2 เซต นำมาหาค่าความเชื่อมั่นที่ไม่น้อยกว่าค่าเชื่อมั่นต่ำสุดจึงทำให้ได้กฎความสัมพันธ์ คือ bread => diaper

สรุปได้ว่าร้านค้าแห่งนี้จะต้องลดราคาขนมปังและผ้าอ้อมในการจัดโปรโมชัน และการใช้อัลกอริทึมเอคาฟ (ACAF) ในการหากฎความสัมพันธ์ สามารถลดจำนวนไอเท็มเซตปรากฏบ่อยในขั้นตอนที่ 2 ซึ่งทำให้ได้รับกฎตรงตามเป้าหมาย

การใช้งานโปรแกรมเอคาฟจะประกอบด้วยสองส่วน คือ การเตรียมข้อมูลและการสอบถามเพื่อหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับ

3.3.1 การเตรียมข้อมูล

การใช้งานโปรแกรมเอคาฟเพื่อหากฎความสัมพันธ์นั้น ข้อมูลที่จะนำมาใช้จะต้องอยู่ในรูปแบบของภาษาโปรล็อก (รูปที่ 3.11)



รูปที่ 3.11 รูปแบบของข้อมูลที่จะนำมาหากฎความสัมพันธ์

จากรูปข้อมูลจะประกอบด้วย 2 ส่วนคือ

1) ไอเท็ม (Item) เป็นการระบุให้โปรแกรมรู้จักกับไอเท็มแต่ละชนิดที่นำมาทดสอบ จากรูปจะประกอบด้วยไอเท็มทั้งหมด 11 ไอเท็ม

2) ทรานแซคชัน (Transaction) เป็นการระบุชุดข้อมูล ซึ่งข้อมูลแต่ละชุดจะต้องอยู่ในเครื่องหมาย [ ] ซึ่งจากรูปจะประกอบด้วยชุดข้อมูลทั้งหมด 12 ทรานแซคชัน

3.3.2 การสอบถามเพื่อหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับ

ในการสอบถามเพื่อหาคำตอบของโปรแกรมจะอยู่ในรูปแบบการสอบถามเชิงตรรกะ (รูปที่ 3.12)

```
?- association(R, Length, MinSup, MinConf).
```

รูปที่ 3.12 รูปแบบการสอบถามเพื่อหากฎความสัมพันธ์

จากรูปที่ 3.12 คือ การสอบถามเพื่อหากฎความสัมพันธ์ โดยจะต้องกำหนดค่าของตัวแปร ดังนี้

Length คือ ขนาดของกฎ เช่นกำหนดเป็น 2 หมายถึงกฎที่ได้รับจะมีสมาชิกมากกว่า 2 ไอเท็ม

MinSup คือ ค่าสนับสนุนต่ำสุด

MinConf คือ ค่าความเชื่อมั่นต่ำสุด

นอกจากนี้เงื่อนไข Length, Minsup และ MinConf ที่ปรากฏดังรูปที่ 3.12 แล้ว โปรแกรมเอคาฟ (ACAF) ยังอำนวยความสะดวกให้ผู้ใช้ระบุเงื่อนไขเจาะจงกับไอเท็มได้อีก 3 ลักษณะ คือ

(1) การระบุให้ต้องมีบางไอเท็มปรากฏในกฎความสัมพันธ์ รายการไอเท็มนี้สามารถใช้ตัวดำเนินการทางตรรกะ and/or ได้

(2) การระบุว่าไม่ให้มีบางไอเท็มปรากฏในกฎความสัมพันธ์ รายการไอเท็มนี้สามารถใช้ตัวดำเนินการทางตรรกะ and/or ได้

(3) การระบุไอเท็มที่ต้องมีในเป้าหมายของกฎ นั่นคือ ในส่วน then ของกฎความสัมพันธ์

หลังจากประมวลคำสั่งตามรูปที่ 3.12 โปรแกรมจะขึ้นกล่องข้อความขึ้นมา 3 ครั้ง เพื่อสอบถามเงื่อนไขพิเศษเพิ่มเติม ในครั้งแรกให้ใส่สมาชิกของกฎที่เราต้องการ โดยจะต้องอยู่ในรูปแบบของลิสต์ที่มีสัญลักษณ์ [ ] ครอบ สมมุติต้องการกฎที่มีสมาชิกเป็น beer และ wine ให้ใส่เป็น [[beer], [wine]] หรือ ถ้าต้องการเป็น beer หรือ wine ให้ใส่เป็น [[beer, wine]] หลังจากนั้นให้กด ok แล้วโปรแกรมจะมีกล่องข้อความขึ้นมาอีกครั้ง ในครั้งที่สองนี้เป็นการระบุสมาชิกที่ไม่ต้องการในกฎ โดยจะใช้รูปแบบคล้ายกับครั้งแรก หลังจากใส่สมาชิกไปแล้วให้กด ok ต่อจากนั้นโปรแกรม จะให้ใส่เป้าหมายของกฎ คือ สมาชิกที่อยู่หลัง then ว่าต้องการสมาชิกตัวไหนเป็นเป้าหมาย ซึ่งมีรูปแบบดังนี้ สมมุติต้องการ wine เป็นเป้าหมายจะมีรูปแบบคือ [wine] หลังจากนั้นให้กด ok โปรแกรมก็จะประมวลผลเพื่อหากฎความสัมพันธ์

ในการสอบถามโดยใช้เงื่อนไขบังคับนั้นจะประกอบด้วยเงื่อนไขต่าง ๆ ที่จะทำให้ได้กฎความสัมพันธ์เป็นไปตามที่ผู้ใช้ต้องการและยังสามารถลดระยะเวลาในการหากฎได้ด้วย และผู้ใช้อาจกำหนดเงื่อนไขหรือไม่กำหนดก็ได้ ซึ่งโปรแกรมมีการสอบถามแบบไม่มีเงื่อนไขพิเศษและแบบมีเงื่อนไขต่างๆ ดังนี้

1. การสอบถามแบบไม่มีเงื่อนไขพิเศษ คือ มีขั้นตอนการทำงานเหมือนอัลกอริทึมเอไพรออริที่มีการกำหนดเฉพาะค่าสนับสนุนต่ำสุดและค่าเชื่อมั่นต่ำสุดและมีรูปแบบการสอบถามดังนี้

?- association(R, 0, 2, 100).

พารามิเตอร์หรืออาร์กิวเม้นต์ต่าง ๆ มีความหมายดังนี้ R คือตัวแปรที่จะทำหน้าที่แสดงผลลัพธ์เป็นกฎความสัมพันธ์ ตัวเลข 0 คือ ขนาดของกฎนั้นจะต้องมากกว่า 0 ตัวเลข 2 หมายถึงค่าสนับสนุนต่ำสุดและตัวเลข 100 หมายถึงค่าความเชื่อมั่นต่ำสุด หลังจากนั้นโปรแกรมจะขึ้นกล่องข้อความมาสามครั้ง ครั้งแรกให้ใส่ [[]] มีความหมายว่าไม่ต้องการใส่เงื่อนไข ครั้งที่สองและสามก็ให้ใส่แบบเดียวกับครั้งแรกเป็น [[]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์ 24 กฎดังรูปที่ 3.13 จากกฎที่ได้จะอยู่ในรูปแบบ If [wine] 4 then [confectionery] 4 ซึ่งมีความหมายว่าถ้า มีคนซื้อ wine สี่คนแล้วทั้งสี่คนจะซื้อ confectionery ด้วย

จากการสอบถามในรูปแบบนี้จะทำให้ได้รับกฎออกมาจำนวนมาก จะเห็นได้ว่าใช้ข้อมูลเพียง 12 ทรานแซคชันแต่ได้กฎความสัมพันธ์มากกว่านั้นเป็นเท่าตัว

If [wine] 4 then [confectionery] 4	If [softdrink, wine] 2 then [fruitveg] 2
If [freshmeat] 2 then [dairy] 2	If [fruitveg, wine] 2 then [softdrink] 2
If [cannedveg, softdrink] 2 then [cannedmeat] 2	If [fruitveg, softdrink] 2 then [wine] 2
If [cannedmeat, softdrink] 2 then [cannedveg] 2	If [softdrink, wine] 2 then [confectionery, fruitveg] 2
If [cannedmeat, cannedveg] 2 then [softdrink] 2	If [fruitveg, wine] 2 then [confectionery, softdrink] 2
If [fruitveg, softdrink] 2 then [confectionery] 2	If [fruitveg, softdrink] 2 then [confectionery, wine] 2
If [confectionery, softdrink] 2 then [fruitveg] 2	If [fruitveg, softdrink, wine] 2 then [confectionery] 2
If [confectionery, fruitveg] 2 then [softdrink] 2	If [confectionery, softdrink] 2 then [fruitveg, wine] 2
If [fruitveg, wine] 2 then [confectionery] 2	If [confectionery, softdrink, wine] 2 then [fruitveg] 2
If [confectionery, fruitveg] 2 then [wine] 2	If [confectionery, fruitveg] 2 then [softdrink, wine] 2
If [softdrink, wine] 2 then [confectionery] 2	If [confectionery, fruitveg, wine] 2 then [softdrink] 2
If [confectionery, softdrink] 2 then [wine] 2	If [confectionery, fruitveg, softdrink] 2 then [wine] 2

รูปที่ 3.13 กฎความสัมพันธ์ที่ได้จากการสอบถามแบบไม่มีเงื่อนไข

2. การสอบถามแบบใช้เงื่อนไขกำหนดขนาดของกฎ คือ การสอบถามเพื่อให้ได้ขนาดสมาชิกของกฎความสัมพันธ์ตามผู้ใช้ต้องการ เช่น ถ้าผู้ใช้ต้องการกฎที่มีสมาชิกมากกว่า 3 ไอเท็ม การสอบถามจะมีรูปแบบดังนี้

?- association(R, 3, 2, 100).

จากการสอบถามอาร์กิวเมนท์ตัวที่สองมีค่าเป็น 3 ซึ่งหมายความว่าต้องการกฎความสัมพันธ์ที่มีสมาชิกมากกว่า 3 หลังจากนั้นโปรแกรมจะขึ้นกล่องข้อความมาสามครั้ง และทั้งสามครั้งใส่ [[]] ข้อมูล (ใส่ลิสต์ว่างเนื่องจากไม่มีเงื่อนไขเพิ่มเติมเกี่ยวกับรายชื่อไอเท็มที่จะปรากฏในกฎความสัมพันธ์) ในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์ 9 กฎ ดังรูปที่ 3.14

If [softdrink, wine] 2 then [confectionery, fruitveg] 2

If [fruitveg, wine] 2 then [confectionery, softdrink] 2

If [fruitveg, softdrink] 2 then [confectionery, wine] 2

If [fruitveg, softdrink, wine] 2 then [confectionery] 2

If [confectionery, softdrink] 2 then [fruitveg, wine] 2

If [confectionery, softdrink, wine] 2 then [fruitveg] 2

If [confectionery, fruitveg] 2 then [softdrink, wine] 2

If [confectionery, fruitveg, wine] 2 then [softdrink] 2

If [confectionery, fruitveg, softdrink] 2 then [wine] 2

รูปที่ 3.14 กฎความสัมพันธ์ที่ได้จากการสอบถามแบบกำหนดจำนวนสมาชิกที่ต้องการ

3. การสอบถามแบบกำหนดสมาชิกที่ต้องการ เป็นการสอบถามเพื่อให้ได้สมาชิกตามที่ผู้ใช้กำหนดโดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้งและจะใช้เงื่อนไขได้ในครั้งแรก เช่น ถ้าผู้ใช้ต้องการ fruitveg เป็นสมาชิกในกฎ ผู้ใช้จะต้องสอบถามด้วยข้อคำถาม ?- association(R, 0, 2, 100). หลังจากนั้นจะขึ้นกล่องข้อความมาสามครั้ง

ในครั้งแรกใส่ [[fruitveg]] ครั้งที่สองและสามใส่ [[]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์จำนวน 17 กฎ ดังรูปที่ 3.15

If [fruitveg, wine] 2 then [confectionery, softdrink] 2	If [fruitveg, softdrink] 2 then [confectionery] 2
If [fruitveg, softdrink] 2 then [confectionery, wine] 2	If [confectionery, softdrink] 2 then [fruitveg] 2
If [fruitveg, softdrink, wine] 2 then [confectionery] 2	If [confectionery, fruitveg] 2 then [softdrink] 2
If [confectionery, softdrink] 2 then [fruitveg, wine] 2	If [fruitveg, wine] 2 then [confectionery] 2
If [confectionery, softdrink, wine] 2 then [fruitveg] 2	If [confectionery, fruitveg] 2 then [wine] 2
If [confectionery, fruitveg] 2 then [softdrink, wine] 2	If [softdrink, wine] 2 then [fruitveg] 2
If [confectionery, fruitveg, wine] 2 then [softdrink] 2	If [fruitveg, wine] 2 then [softdrink] 2
If [confectionery, fruitveg, softdrink] 2 then [wine] 2	If [fruitveg, softdrink] 2 then [wine] 2
If [softdrink, wine] 2 then [confectionery, fruitveg] 2	

รูปที่ 3.15 กฎความสัมพันธ์ที่ได้จากการสอบถามแบบกำหนดสมาชิกที่ต้องการเป็น fruitveg

4. การสอบถามแบบกำหนดสมาชิกที่ไม่ต้องการ เป็นการสอบถามเพื่อให้ได้กฎที่ไม่มีสมาชิกที่เราไม่ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้งและจะใช้เงื่อนไขได้ในกล่องข้อความครั้งที่สอง เช่น ถ้าผู้ใช้ไม่ต้องการ fruitveg เป็นสมาชิกในกฎ ผู้ใช้จะต้องสอบถามโดย ?- association(R, 0, 2, 100). หลังจากนั้นจะขึ้นกล่องข้อความมาสามครั้ง ในครั้งแรกและครั้งที่สามใส่ [[]] ครั้งที่สองใส่ [[fruitveg]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์จำนวน 7 กฎ ดังรูปที่ 3.1

If [wine] 4 then [confectionery] 4

If [freshmeat] 2 then [dairy] 2

If [cannedveg, softdrink] 2 then [cannedmeat] 2

If [cannedmeat, softdrink] 2 then [cannedveg] 2

If [cannedmeat, cannedveg] 2 then [softdrink] 2

If [softdrink, wine] 2 then [confectionery] 2

If [confectionery, softdrink] 2 then [wine] 2

รูปที่ 3.16 กฎความสัมพันธ์ได้จากการสอบถามแบบกำหนดสมาชิกที่ไม่ต้องการเป็น fruitveg

5. การสอบถามแบบใช้เงื่อนไขหรือ (OR) คือ การสอบถามเพื่อให้ได้กฎความสัมพันธ์ที่ประกอบด้วยสมาชิกบางตัวตามที่ผู้ใช้ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้ง เช่น ถ้าผู้ใช้ต้องการ fruitveg หรือ softdrink ไม่ต้องการ cannedmeat หรือ cannedveg โดยใช้การสอบถาม ?- association(R, 0, 2, 100). หลังจากนั้นจะขึ้นกล่องข้อความมาสามครั้ง ในครั้งแรกใส่ [[fruitveg, softdrink]] ครั้งที่สองใส่ [[cannedmeat, cannedveg]] และครั้งที่สามใส่ [[]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์จำนวน 19 กฎ ดังรูปที่ 3.17

If [fruitveg, softdrink] 2 then [confectionery] 2	If [softdrink, wine] 2 then [confectionery, fruitveg] 2
If [confectionery, softdrink] 2 then [fruitveg] 2	If [fruitveg, wine] 2 then [confectionery, softdrink] 2
If [confectionery, fruitveg] 2 then [softdrink] 2	If [fruitveg, softdrink] 2 then [confectionery, wine] 2
If [fruitveg, wine] 2 then [confectionery] 2	If [fruitveg, softdrink, wine] 2 then [confectionery] 2
If [confectionery, fruitveg] 2 then [wine] 2	If [confectionery, softdrink] 2 then [fruitveg, wine] 2
If [softdrink, wine] 2 then [confectionery] 2	If [confectionery, softdrink, wine] 2 then [fruitveg] 2
If [confectionery, softdrink] 2 then [wine] 2	If [confectionery, fruitveg] 2 then [softdrink, wine] 2
If [softdrink, wine] 2 then [fruitveg] 2	If [confectionery, fruitveg, wine] 2 then [softdrink] 2
If [fruitveg, wine] 2 then [softdrink] 2	If [confectionery, fruitveg, softdrink] 2 then [wine] 2
If [fruitveg, softdrink] 2 then [wine] 2	

รูปที่ 3.17 กฎความสัมพันธ์ที่ได้จากการสอบถามแบบใช้เงื่อนไขหรือ (OR)

6. การสอบถามแบบใช้เงื่อนไขและ (AND) คือ การสอบถามเพื่อให้ได้กฎความสัมพันธ์ที่ประกอบด้วยสมาชิกทุกตัวตามที่ผู้ใช้ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะคล้ายกับการสอบถามแบบใช้เงื่อนไขหรือ (OR) ใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้ง เช่น ถ้าผู้ใช้ต้องการ fruitveg และ softdrink แต่ไม่ต้องการ cannedmeat และ cannedveg โดยใช้การสอบถาม ?- association(R, 0, 2, 100). หลังจากนั้นจะขึ้นกล่องข้อความมาสามครั้ง ในครั้งแรกใส่ [[fruitveg], [softdrink]] ครั้งที่สองใส่ [[cannedmeat], [cannedveg]] และครั้งที่สามใส่ [[]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์จำนวน 15 กฎ ดังรูปที่ 3.18

If [softdrink, wine] 2 then [confectionery, fruitveg] 2	If [fruitveg, softdrink] 2 then [confectionery] 2
If [fruitveg, wine] 2 then [confectionery, softdrink] 2	If [confectionery, softdrink] 2 then [fruitveg] 2
If [fruitveg, softdrink] 2 then [confectionery, wine] 2	If [confectionery, fruitveg] 2 then [softdrink] 2
If [fruitveg, softdrink, wine] 2 then [confectionery] 2	If [softdrink, wine] 2 then [fruitveg] 2
If [confectionery, softdrink] 2 then [fruitveg, wine] 2	If [fruitveg, wine] 2 then [softdrink] 2
If [confectionery, softdrink, wine] 2 then [fruitveg] 2	If [fruitveg, softdrink] 2 then [wine] 2
If [confectionery, fruitveg] 2 then [softdrink, wine] 2	
If [confectionery, fruitveg, wine] 2 then [softdrink] 2	
If [confectionery, fruitveg, softdrink] 2 then [wine] 2	

รูปที่ 3.18 กฎความสัมพันธ์ที่ได้จากการสอบถามแบบใช้เงื่อนไขและ (AND)

7. การสอบถามแบบใช้เงื่อนไขกำหนดเป้าหมาย คือ การสอบถามโดยมีการกำหนดสมาชิกหลัง then เพื่อให้ได้กฎความสัมพันธ์ตามผู้ใช้ต้องการ และสามารถใช้เงื่อนไขนี้ได้ในขณะที่โปรแกรมขึ้นกล่องข้อความครั้งที่สาม เช่น ถ้าต้องการเป้าหมายที่ประกอบด้วย wine และ softdrink ใช้การสอบถาม ?- association(R, 0, 2, 100). หลังจากนั้นจะขึ้นกล่องข้อความมาสามครั้ง ในครั้งแรกใส่ [[]] ครั้งที่สองใส่ [[]] และครั้งที่สามใส่ [[wine], [softdrink]] ข้อมูลในการสอบถามใช้ข้อมูลจากรูปที่ 3.11 ซึ่งจะได้กฎความสัมพันธ์ 1 กฎ คือ

If [confectionery, fruitveg] 2 then [softdrink, wine] 2

3.4 เครื่องมือที่ใช้ในงานวิจัย

ในส่วนนี้จะกล่าวถึงเครื่องมือที่ใช้ในการพัฒนา โดยมีรายละเอียดดังนี้

3.4.1 เครื่องมือที่ใช้ในการพัฒนา

1) เครื่องคอมพิวเตอร์สำหรับการพัฒนา มีรายละเอียดดังนี้

- หน่วยประมวลผลกลาง : Intel Core i7
- หน่วยความจำสำรอง : 320 GB
- หน่วยความจำหลัก : 4 GB
- อุปกรณ์เสริมอื่นๆ เช่น เมาส์ แป้นพิมพ์ เป็นต้น

2) ระบบปฏิบัติการและโปรแกรมประยุกต์สำหรับการพัฒนา ประกอบไปด้วย

- ระบบปฏิบัติการ : Windows xp service pack 3 32bits
- เครื่องมือที่ใช้ในการพัฒนา : EClipSe 6.0

3.4.2 เครื่องมือที่ใช้ในการประเมินประสิทธิภาพ

เครื่องมือที่ใช้ในการประเมินประสิทธิภาพของระบบ คือ โปรแกรมอีคลิปส์ โปรแกรมจะมีการคำนวณคำตอบและเวลาที่ใช้ในการประมวลผล และจะแสดงออกทางหน้าจอหลังจากประมวลผลเสร็จ การประเมินประสิทธิภาพจะนำเวลาที่ได้จากการประมวลผลของโปรแกรมเปรียบเทียบกันระหว่าง โปรแกรม Apriori โปรแกรม ACIF ซึ่งมีการใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อยและโปรแกรม ACAF ที่มีการใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย การเปรียบเทียบมีจุดมุ่งหมายที่จะแสดงผลว่าโปรแกรมในรูปแบบใดใช้เวลาน้อยที่สุดและมีความถูกต้องสูงสุด

บทที่ 4

การทดสอบและอภิปรายผล

การทดสอบประสิทธิภาพของระบบนั้น จะเป็นการทดสอบประสิทธิภาพในการค้นหากฎความสัมพันธ์ (Association Rule) ด้วยโปรแกรม ACAF ซึ่งในการเปรียบเทียบการหากฎความสัมพันธ์นั้น จะเปรียบเทียบกับอีกสองโปรแกรม คือ โปรแกรม Apriori และโปรแกรม ACIF โดยจะทำการเปรียบเทียบจากระยะเวลาในการค้นพบกฎความสัมพันธ์และจะเปรียบเทียบความถูกต้องว่าจำนวนกฎความสัมพันธ์ที่ได้รับตรงตามเงื่อนไขที่ผู้ใช้ระบุหรือไม่

4.1 ข้อมูลที่ใช้ในการทดสอบ

ในการทดสอบการค้นหากฎความสัมพันธ์ของทั้ง 3 โปรแกรมจะใช้ข้อมูลมาตรฐาน ซึ่งเป็นข้อมูลเกี่ยวกับหมากรุก (Chess) สามารถโหลดได้ที่ http://fimi.ua.ac.be/data/ โดยมีข้อมูลทั้งหมด 3,196 แถว ประกอบด้วยแอททริบิวท์จำนวน 37 แอททริบิวท์ และข้อมูลจะถูกแปลงให้อยู่ในรูปแบบของข้อความในภาษาโปรล็อกเพื่อให้เหมาะสมกับการทำงานของโปรแกรม โดยมีรายละเอียดตัวอย่างของข้อมูลดังรูปที่ 4.1 โดยข้อมูลที่อยู่ใน [ ] ส่วนแรก เป็นการระบุไอเท็มให้โปรแกรมได้รู้ว่าข้อมูลที่นำมามีไอเท็มอะไรบ้าง จากข้อมูลจะมีไอเท็มทั้งหมด 75 ไอเท็ม และข้อมูลที่อยู่ใน [ ] ส่วนที่สอง คือ ทรานแซคชัน (Transaction) จากรูปแสดงตัวอย่างเพียง 7 ทรานแซคชัน แต่ในการทดลองจะใช้ 3,196 ทรานแซคชัน

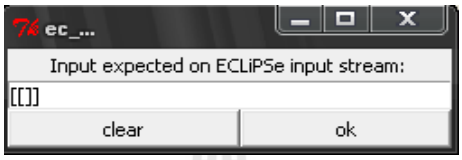
```
data([
        [1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],[15],[16],[17],[18],[19],[20],[21],[22],[23],[24],[25],[26],[27],[28],
        [29],[30],[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[41],[42],[43],[44],[45],[46],[47],[48],[49],[50],[51],[52],[53],[54],
        [55],[56],[57],[58],[59],[60],[61],[62],[63],[64],[65],[66],[67],[68],[69],[70],[71],[72],[73],[74],[75]
]-[
        [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74],
        [1,3,5,7,9,12,13,15,17,19,21,23,25,27,29,31,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74],
        [1,3,5,7,9,12,13,16,17,19,21,23,25,27,29,31,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74],
         [2,3,5,7,10,12,14,15,17,19,21,23,25,27,29,31,34,36,38,40,42,44,46,48,50,52,55,57,58,60,62,64,66,69,70,72,75],
        [2,3,6,7,9,11,14,15,17,19,21,23,25,27,29,31,35,36,38,40,42,44,46,48,50,52,55,56,58,60,62,64,66,69,71,72,75],
         [2,3,5,7,10,11,13,15,17,19,21,23,25,27,29,32,34,36,39,40,43,44,46,48,50,52,54,56,58,61,62,64,67,68,70,73,74],
        [2,4,5,8,9,11,13,16,17,19,21,23,26,27,30,31,35,36,38,40,42,44,46,48,51,52,54,56,58,61,62,64,67,68,71,73,74]
]       ).
```

รูปที่ 4.1 ข้อมูลหมากรุก (Chess)

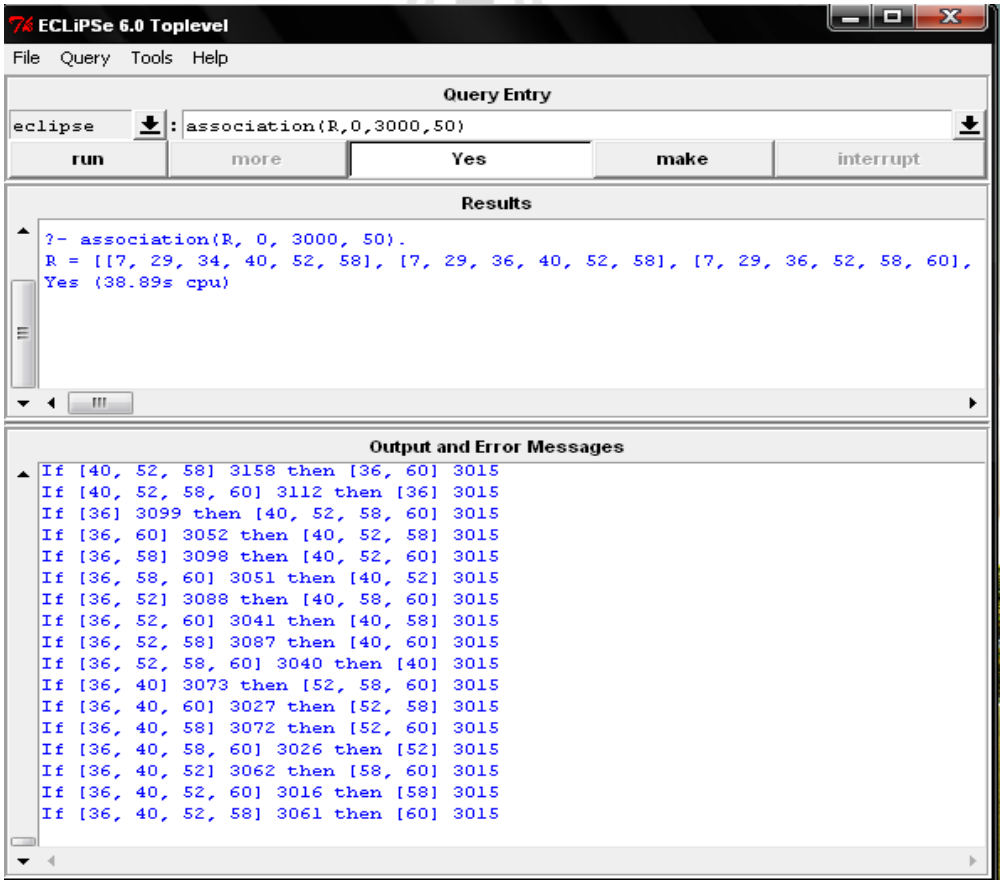
4.2 ผลของการสอบถามในรูปแบบต่าง ๆ

4.2.1 ผลของการสอบถามในรูปแบบไม่มีเงื่อนไขพิเศษ

ทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไขเป็นลิสต์ว่างหมายถึงไม่มีเงื่อนไขพิเศษเพิ่มเติมนอกเหนือจากค่า Minimum Support = 3000 และ Minimum Confidence = 50% ดังรูปที่ 4.2 และจะได้ผลลัพธ์ทั้งหมด 1,268 กฎ ดังรูปที่ 4.3 และตัวอย่างกฎความสัมพันธ์ที่ได้รับดังรูปที่ 4.4



รูปที่ 4.2 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้งของการสอบถามในรูปแบบไม่มีเงื่อนไขพิเศษ



รูปที่ 4.3 จอภาพแสดงผลของการสอบถามในรูปแบบไม่มีเงื่อนไขพิเศษ

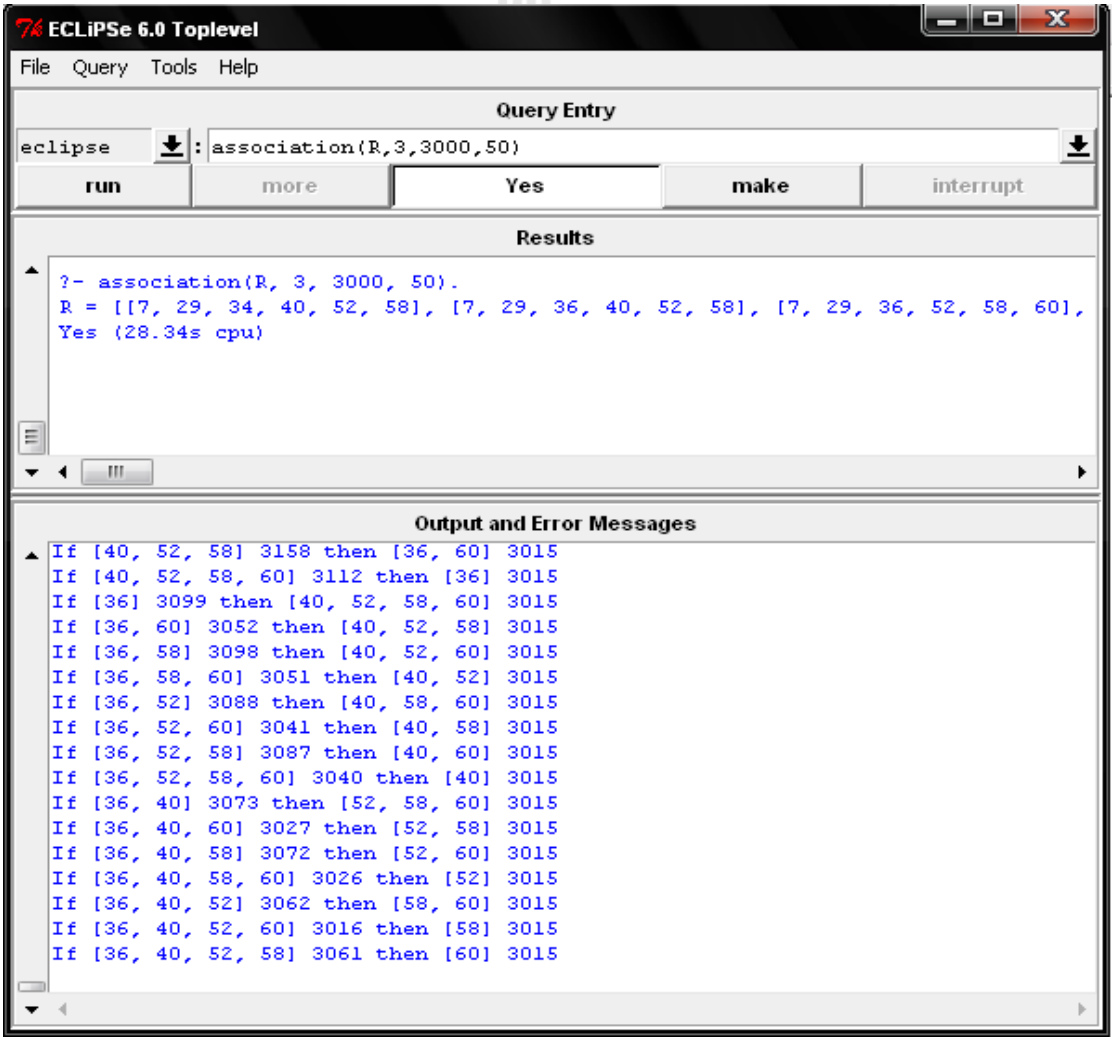
If [7] 3076 then [52] 3065	If [7] 3076 then [29, 52] 3058
If [58] 3195 then [7] 3075	If [7, 52] 3065 then [29] 3058
If [7] 3076 then [58] 3075	If [34, 40] 3017 then [29, 52] 3004
If [60] 3149 then [7] 3031	If [34, 40, 52] 3008 then [29] 3004
If [7] 3076 then [60] 3031	If [29] 3181 then [34, 40, 52] 3004
If [34] 3040 then [29] 3036	If [29, 52] 3170 then [34, 40] 3004
If [29] 3181 then [34] 3036	If [29, 40] 3155 then [34, 52] 3004
If [40] 3170 then [7, 29] 3043	If [36, 52, 58] 3087 then [40, 60] 3015
If [29] 3181 then [7, 40] 3043	If [36, 52, 58, 60] 3040 then [40] 3015
If [29, 40] 3155 then [7] 3043	If [36, 40] 3073 then [52, 58, 60] 3015
If [7] 3076 then [29, 40] 3043	If [36, 40, 60] 3027 then [52, 58] 3015
If [7, 40] 3050 then [29] 3043	If [36, 40, 58] 3072 then [52, 60] 3015
If [7, 29] 3069 then [40] 3043	If [36, 40, 58, 60] 3026 then [52] 3015
If [52] 3185 then [7, 29] 3058	If [36, 40, 52] 3062 then [58, 60] 3015
If [29] 3181 then [7, 52] 3058	If [36, 40, 52, 60] 3016 then [58] 3015
If [29, 52] 3170 then [7] 3058	If [36, 40, 52, 58] 3061 then [60] 3015

รูปที่ 4.4 ตัวอย่าง 32 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 1,268 กฎ
ในการสอบถามแบบไม่มีเงื่อนไขพิเศษ

จากรูปที่ 4.4 จะได้กฎความสัมพันธ์จำนวนมากและการทำงานของโปรแกรมเป็นการเลียนแบบการทำงานของอัลกอริทึมเอไพรออริโดยไม่มีเงื่อนไขเพิ่มเติม และกฎที่ได้จะมีขนาดของสมาชิกอยู่ที่ 2,3,4 และ 5 ไอเท็ม

4.2.2 ผลของการสอบถามในรูปแบบใช้เงื่อนไขกำหนดขนาดของกฎ

ทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 3, 3000, 50) หมายถึงขนาดของกฎจะต้องมากกว่า 3 นั่นคือมีจำนวนไอเท็มปรากฏในกฎตั้งแต่ 4, 5, 6, ...โดยทั้งสามกล่องข้อความจะใส่เงื่อนไขเป็นลิสต์ว่าง ดังรูปที่ 4.2 และจะได้ผลลัพธ์ทั้งหมด 862 กฎ ดังรูปที่ 4.5 และตัวอย่างกฎความสัมพันธ์ที่ได้รับแสดงดังรูปที่ 4.6



รูปที่ 4.5 ผลของการสอบถามในรูปแบบใช้เงื่อนไขกำหนดขนาดของกฎ

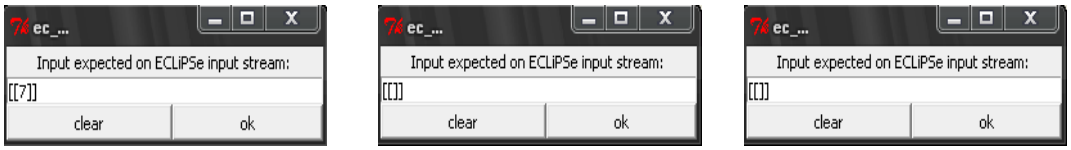
If [52] 3185 then [7, 29, 40] 3032	If [40] 3170 then [36, 52, 58, 60] 3015
If [40] 3170 then [7, 29, 52] 3032	If [40, 60] 3124 then [36, 52, 58] 3015
If [40, 52] 3159 then [7, 29] 3032	If [40, 58] 3169 then [36, 52, 60] 3015
If [29] 3181 then [7, 40, 52] 3032	If [40, 58, 60] 3123 then [36, 52] 3015
If [29, 52] 3170 then [7, 40] 3032	If [40, 52] 3159 then [36, 58, 60] 3015
If [29, 40] 3155 then [7, 52] 3032	If [40, 52, 60] 3113 then [36, 58] 3015
If [29, 40, 52] 3144 then [7] 3032	If [40, 52, 58] 3158 then [36, 60] 3015
If [7] 3076 then [29, 40, 52] 3032	If [40, 52, 58, 60] 3112 then [36] 3015
If [7, 52] 3065 then [29, 40] 3032	If [36] 3099 then [40, 52, 58, 60] 3015
If [7, 40] 3050 then [29, 52] 3032	If [36, 60] 3052 then [40, 52, 58] 3015
If [7, 40, 52] 3039 then [29] 3032	If [36, 58] 3098 then [40, 52, 60] 3015
If [7, 29] 3069 then [40, 52] 3032	If [36, 58, 60] 3051 then [40, 52] 3015
If [7, 29, 52] 3058 then [40] 3032	If [36, 52] 3088 then [40, 58, 60] 3015
If [7, 29, 40] 3043 then [52] 3032	If [36, 52, 60] 3041 then [40, 58] 3015
If [58] 3195 then [7, 29, 40] 3042	If [36, 52, 58] 3087 then [40, 60] 3015
If [40] 3170 then [7, 29, 58] 3042	If [36, 52, 58, 60] 3040 then [40] 3015

รูปที่ 4.6 ตัวอย่าง 32 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 862 กฎ ในการสอบถามที่กำหนดขนาดของกฎให้ปรากฏมากกว่า 3 ไอเท็ม

จากรูปที่ 4.6 จะเห็นได้ว่ากฎที่ได้รับมีจำนวนลดลงเนื่องจากได้มีการกำหนดขนาดของสมาชิกกฎเข้าไปโดยกำหนดให้กฎที่ได้รับนั้นจะต้องมีสมาชิกของกฎมากว่า 3 ไอเท็ม กฎที่ได้รับจึงมีสมาชิกอยู่ที่ 4, 5 ไอเท็ม และจากรูปที่ 4.5 จะเห็นได้ว่าเวลาในการประมวลผลเมื่อเทียบกับแบบไม่มีเงื่อนไขแล้วแบบกำหนดขนาดของกฎจะใช้เวลาที่น้อยกว่า

4.2.3 ผลของการสอบถามแบบกำหนดสมาชิกที่ต้องการ

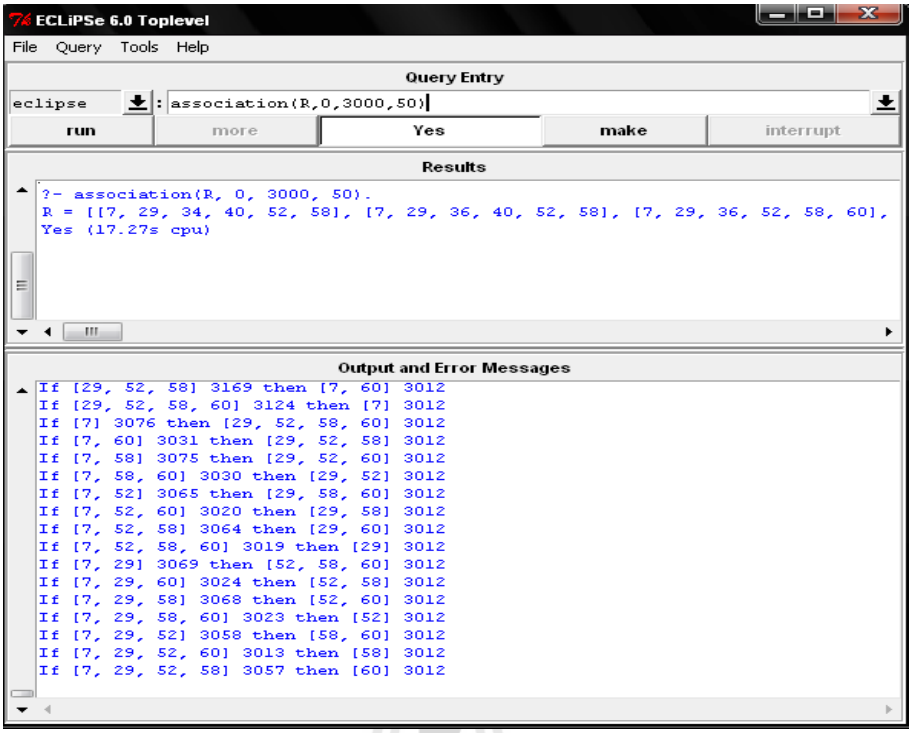
การทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไข ดังรูปที่ 4.7 และจะได้ผลลัพธ์ทั้งหมด 242 กฎ ดังรูปที่ 4.8 และ 4.9



ครั้งที่ 1 : ระบุไอเท็มที่ต้องการ ครั้งที่ 2 : ไอเท็มที่ไม่ต้องการ ครั้งที่ 3 : ระบุเป้าหมายของกฎ

รูปที่ 4.7 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้งของการสอบถามแบบกำหนดสมาชิกที่ ต้องการให้ปรากฏในกฎความสัมพันธ์

จากรูปที่ 4.7 ในการใส่เงื่อนไขครั้งแรกเป็น [[7]] มีความหมายว่าต้องการกฎที่ประกอบด้วยสมาชิกเป็น 7 ในการใส่เงื่อนไขในครั้งที่สองและสามเป็น [[]] มีความหมายว่าไม่ใส่เงื่อนไขพิเศษอื่นใดเพิ่มเติม



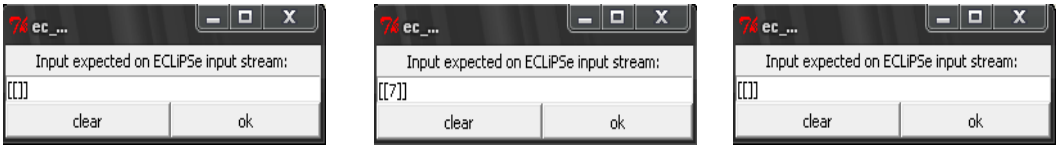
รูปที่ 4.8 ผลของการสอบถามแบบกำหนดสมาชิกที่ต้องการเป็นไอเท็ม 7

If [58] 3195 then [7] 3075	If [52] 3185 then [7, 29, 40] 3032
If [7] 3076 then [58] 3075	If [40] 3170 then [7, 29, 52] 3032
If [60] 3149 then [7] 3031	If [40, 52] 3159 then [7, 29] 3032
If [7] 3076 then [60] 3031	If [7, 29, 58, 60] 3023 then [52] 3012
If [40] 3170 then [7, 29] 3043	If [7, 29, 52] 3058 then [58, 60] 3012
If [29] 3181 then [7, 40] 3043	If [7, 29, 52, 60] 3013 then [58] 3012
If [29, 40] 3155 then [7] 3043	If [7, 29, 52, 58] 3057 then [60] 3012
If [7] 3076 then [29, 40] 3043	If [29] 3181 then [7, 40, 52, 58] 3031

รูปที่ 4.9 ตัวอย่าง 16 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 242 กฎ ที่ระบุว่าต้องปรากฏไอเท็ม 7

4.2.4 ผลของการสอบถามแบบกำหนดสมาชิกที่ไม่ต้องการ

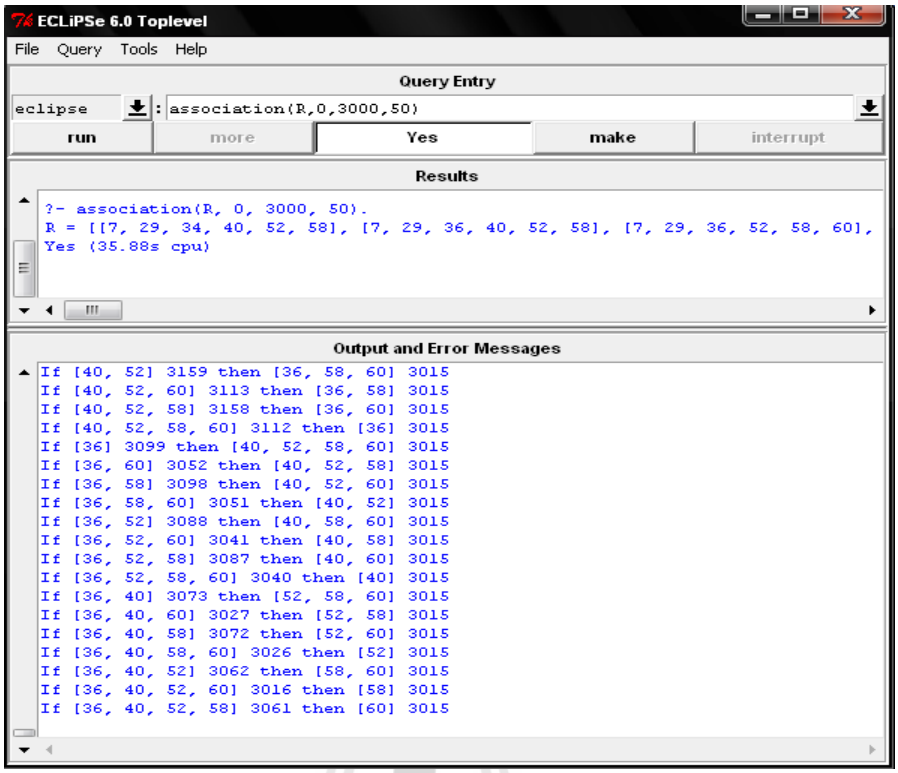
การทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไข ดังรูปที่ 4.10 และจะได้ผลลัพธ์ทั้งหมด 1026 กฎ ดังรูปที่ 4.11 และ 4.12



ครั้งที่ 1 : ระบุไอเท็มที่ต้องการ ครั้งที่ 2 : ไอเท็มที่ไม่ต้องการ ครั้งที่ 3 : ระบุเป้าหมายของกฎ

รูปที่ 4.10 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้งของการสอบถาม
แบบกำหนดสมาชิกที่ไม่ต้องการ

จากรูปที่ 4.10 ในการใส่เงื่อนไขครั้งแรกเป็น [[]] มีความหมายว่าไม่ใส่เงื่อนไขพิเศษเพิ่มเติมว่าต้องการให้ไอเท็มใดบ้างปรากฏในกฎ ครั้งที่สองใส่เป็น [[7]] มีความหมายว่าไม่ต้องการกฎที่มี 7 เป็นสมาชิก ในการใส่เงื่อนไขในครั้งที่สามเป็น [[]] มีความหมายว่าไม่ใส่เงื่อนไขเพิ่มเติมเกี่ยวกับเป้าหมายของกฎ



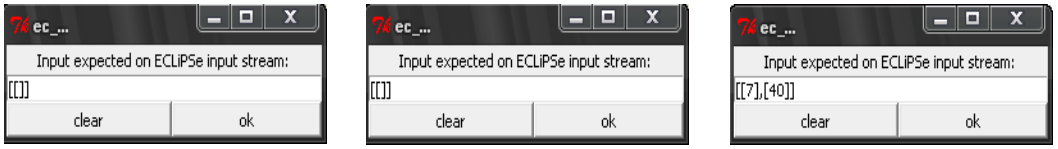
รูปที่ 4.11 ผลของการสอบถามแบบกำหนดสมาชิกที่ไม่ต้องการโดยระบุไอเท็มเป็น 7

If [60] 3149 then [52] 3138	If [52, 60, 66] 3010 then [58] 3009
If [52] 3185 then [60] 3138	If [52, 58] 3184 then [60, 66] 3009
If [62] 3060 then [52] 3049	If [52, 58, 66] 3009 then [60] 3009
If [52] 3185 then [62] 3049	If [52, 58, 60] 3137 then [66] 3009
If [58, 62] 3060 then [60] 3014	If [36, 40, 58, 60] 3026 then [52] 3015
If [58, 60] 3148 then [62] 3014	If [36, 40, 52] 3062 then [58, 60] 3015
If [66] 3021 then [58, 60] 3020	If [36, 40, 52, 60] 3016 then [58] 3015
If [60] 3149 then [58, 66] 3020	If [36, 40, 52, 58] 3061 then [60] 3015

รูปที่ 4.12 ตัวอย่าง 16 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 1026 กฎกรณีระบุว่าไม่ต้องการไอเท็ม7

4.2.5 ผลของการสอบถามแบบใช้เงื่อนไขกำหนดเป้าหมายของกฎ

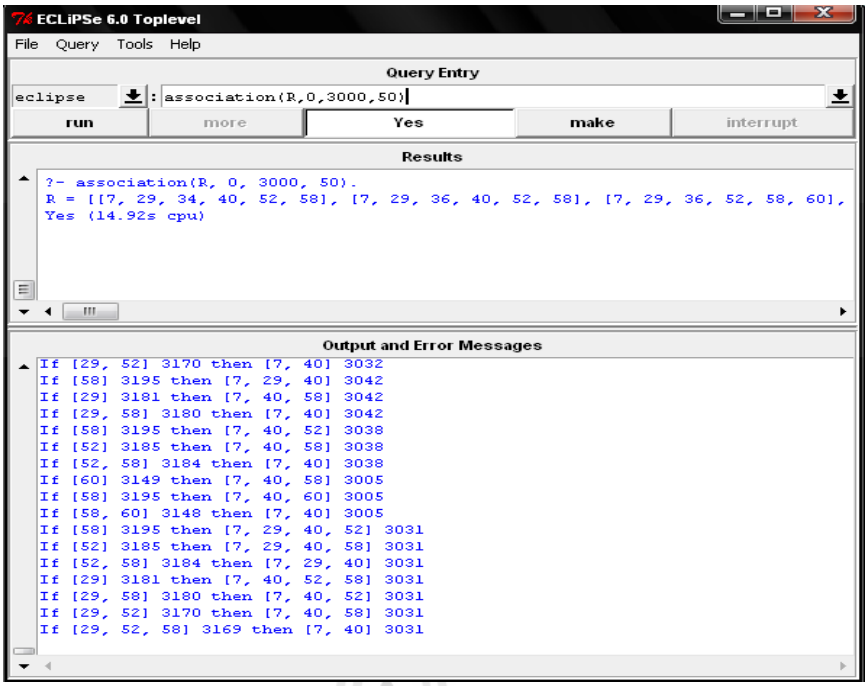
การทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไข ดังรูปที่ 4.13 และจะได้ผลลัพธ์ทั้งหมด 23 กฎ ดังรูปที่ 4.14 และ 4.15



ครั้งที่ 1 : ระบุไอเท็มที่ต้องการ ครั้งที่ 2 : ไอเท็มที่ไม่ต้องการ ครั้งที่ 3 : ระบุเป้าหมายของกฎ

รูปที่ 4.13 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้ง

จากรูปที่ 4.13 ในการใส่เงื่อนไขครั้งแรกและครั้งที่สองเป็นลิสต์ว่างคือไม่ใส่เงื่อนไขอื่นใดเพิ่มเติมเกี่ยวกับไอเท็มที่ต้องการ/ไม่ต้องการให้ปรากฏ แต่ในครั้งที่สามจะใส่เงื่อนไขเป็น [[7],[40]] มีความหมายว่าต้องการกฎที่มีสมาชิกหลัง then เป็น 7 และ 40



รูปที่ 4.14 ผลของการสอบถามแบบใช้เงื่อนไขกำหนดเป้าหมายของกฎให้ต้องมีไอเท็ม 7 และ 40

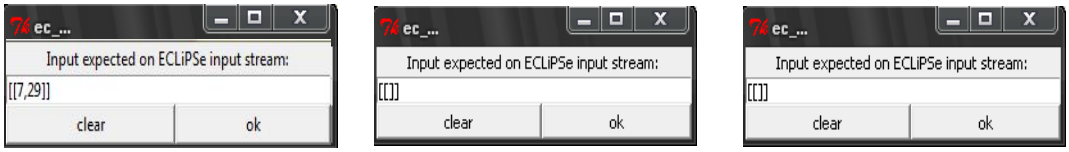
If [29] 3181 then [7, 40] 3043	If [52, 58] 3184 then [7, 40] 3038
If [52] 3185 then [7, 40] 3039	If [60] 3149 then [7, 40, 58] 3005
If [58] 3195 then [7, 40] 3049	If [58] 3195 then [7, 40, 60] 3005
If [60] 3149 then [7, 40] 3006	If [58, 60] 3148 then [7, 40] 3005
If [52] 3185 then [7, 29, 40] 3032	If [58] 3195 then [7, 29, 40, 52] 3031
If [29] 3181 then [7, 40, 52] 3032	If [52] 3185 then [7, 29, 40, 58] 3031
If [29, 52] 3170 then [7, 40] 3032	If [52, 58] 3184 then [7, 29, 40] 3031
If [58] 3195 then [7, 29, 40] 3042	If [29] 3181 then [7, 40, 52, 58] 3031

รูปที่ 4.15 ตัวอย่าง 16 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 23 กฎ กรณีระบุเป้าหมายของกฎเป็นไอเท็ม 7 และ 40

จากรูปที่ 4.14 เวลาที่ใช้ในการประมวลผลและจำนวนกฎที่ได้มีจำนวนที่ลดลงเพราะว่าในการสอบถามได้กำหนดสมาชิกที่อยู่ข้างหลัง then คือ 7 และ 40 ซึ่งจากกฎทั้งหมด (รูปที่ 4.4) 1,268 กฎ มีเพียง 23 กฎที่มีสมาชิกหลัง then เป็น 7 และ 40 จึงทำให้เวลาและจำนวนกฎลดลง แต่ถ้าทั้ง 1,268 กฎ มีสมาชิกหลัง then เป็น 7 และ 40 ทุกกฎ กฎที่จะได้รับจะเป็น 1,268 กฎ ซึ่งการกำหนดเงื่อนไขก็จะไม่ช่วยในการลดจำนวนกฎและเวลาในการประมวลผล

4.2.6 ผลของการสอบถามในรูปแบบใช้เงื่อนไขหรือ (OR)

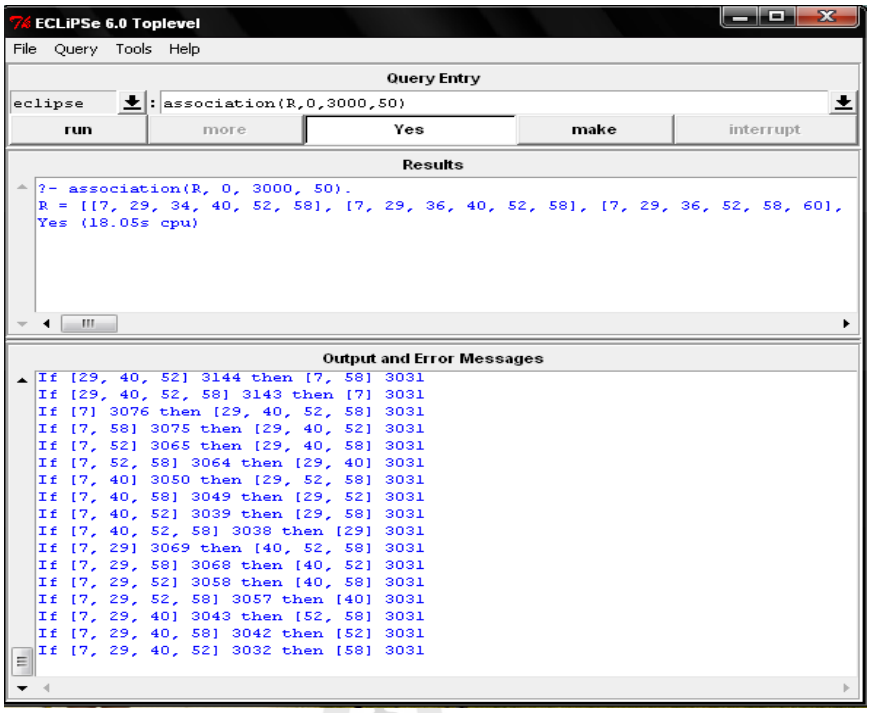
การทดสอบกับโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไข ดังรูปที่ 4.16 และจะได้ผลลัพธ์ทั้งหมด 926 กฎ ดังรูปที่ 4.17 และ 4.18



ครั้งที่ 1 : ระบุไอเท็มที่ต้องการ ครั้งที่ 2 : ไอเท็มที่ไม่ต้องการ ครั้งที่ 3 : ระบุเป้าหมายของกฎ

รูปที่ 4.16 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้งกรณีมีการใช้เงื่อนไข OR

จากรูปที่ 4.16 ในการใส่เงื่อนไขครั้งแรกเป็น [[7,29]] มีความหมายว่าต้องการกฎที่ประกอบด้วยสมาชิกเป็น 7 หรือ 29 ในการใส่เงื่อนไขครั้งที่สองและครั้งที่สามเป็นลิสต์ว่างคือไม่ใส่เงื่อนไขอื่นใดเพิ่มเติมเกี่ยวกับไอเท็มที่ไม่ต้องการ/ไอเท็มเป้าหมายของกฎ



รูปที่ 4.17 ผลของการสอบถามในรูปแบบใช้เงื่อนไขหรือ (OR)

If [7] 3076 then [40] 3050	If [7, 60] 3031 then [58] 3030
If [52] 3185 then [7] 3065	If [7, 58] 3075 then [60] 3030
If [7] 3076 then [52] 3065	If [40] 3170 then [29, 34] 3013
If [58] 3195 then [7] 3075	If [34] 3040 then [29, 40] 3013
If [7] 3076 then [58] 3075	If [52] 3185 then [7, 29, 40] 3032
If [60] 3149 then [7] 3031	If [40] 3170 then [7, 29, 52] 3032
If [7] 3076 then [60] 3031	If [40, 52] 3159 then [7, 29] 3032
If [34] 3040 then [29] 3036	If [29] 3181 then [7, 40, 52] 3032

รูปที่ 4.18 ตัวอย่าง 16 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 962 กฎ เมื่อเงื่อนไขคือ ต้องปรากฏไอเท็ม 7 หรือ 29

จากรูปที่ 4.16 สังเกตได้ว่าเวลาที่ใช้ในการประมวลผลและจำนวนกฎมีจำนวนที่ลดลงกว่าการสอบถามในรูปแบบไม่มีเงื่อนไขพิเศษเพราะว่าในการสอบถามมีการกำหนดเงื่อนไขที่เฉพาะเจาะจงคือต้องการสมาชิกบางตัวจากทั้งหมด

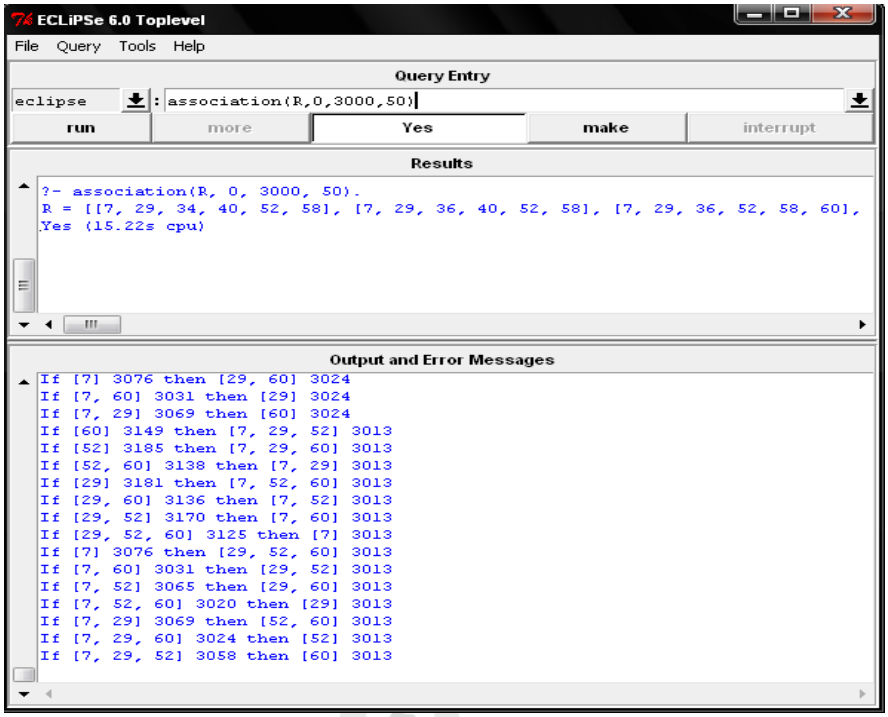
4.2.7 ผลของการสอบถามในรูปแบบใช้เงื่อนไขและ (AND)

การทดสอบกับโปรแกรมโปรแกรม ACAF และมีการสอบถามเป็น ?- association(R, 0, 3000, 50) โดยทั้งสามกล่องข้อความจะใส่เงื่อนไข ดังรูปที่ 4.19 และจะได้ผลลัพธ์ 156 กฎ ดังรูปที่ 4.20 และ 4.21

ครั้งที่ 1 : ระบุไอเท็มที่ต้องการ ครั้งที่ 2 : ไอเท็มที่ไม่ต้องการ ครั้งที่ 3 : ระบุเป้าหมายของกฎ

รูปที่ 4.19 การใส่เงื่อนไขในกล่องข้อความทั้งสามครั้ง กรณีมีการใช้เงื่อนไข AND

จากรูปที่ 4.19 ในการใส่เงื่อนไขครั้งแรกเป็น [[7],[29]] มีความหมายว่าต้องการกฎที่ประกอบด้วยสมาชิกเป็น 7 และ 29 ในการใส่เงื่อนไขครั้งที่สองและครั้งที่สามเป็นลิสต์ว่างคือไม่ใส่เงื่อนไขอื่นใดเพิ่มเติมเกี่ยวกับไอเท็มที่ไม่ต้องการ/ไอเท็มเป้าหมายของกฎ



รูปที่ 4.20 ผลของการสอบถามในรูปแบบใช้เงื่อนไขและ (AND)

If [29] 3181 then [7] 3069	If [29, 52, 60] 3125 then [7] 3013
If [7] 3076 then [29] 3069	If [7] 3076 then [29, 52, 60] 3013
If [52] 3185 then [7, 29] 3058	If [7, 60] 3031 then [29, 52] 3013
If [29] 3181 then [7, 52] 3058	If [7, 52] 3065 then [29, 60] 3013
If [29, 52] 3170 then [7] 3058	If [7, 52, 60] 3020 then [29] 3013
If [7] 3076 then [29, 52] 3058	If [7, 29] 3069 then [52, 60] 3013
If [7, 29] 3069 then [60] 3024	If [7, 29, 60] 3024 then [52] 3013
If [60] 3149 then [7, 29, 52] 3013	If [7, 29, 52] 3058 then [60] 3013

รูปที่ 4.21 ตัวอย่าง 16 กฎความสัมพันธ์ที่ได้รับจากทั้งหมด 156 กฎ เมื่อเงื่อนไขคือต้องปรากฏไอเท็ม 7 และ 29

จากรูปที่ 4.19 เมื่อเปรียบเทียบรูปที่ 4.16 ที่ใช้เงื่อนไข OR พบว่าเวลาที่ใช้ในการประมวลผลน้อยลง (จาก 18.05 วินาทีเหลือเพียง 15.22 วินาที) จำนวนกฎมีจำนวนที่ลดลง (จาก 962 กฎเหลือเพียง 156 กฎ) เนื่องจากว่าในการสอบถามมีการกำหนดเงื่อนไขที่เฉพาะเจาะจงคือต้องการกฎที่มีสมาชิกเป็น 7 และ 29 ซึ่งถ้าเทียบกับการใช้หรือ (OR) จะเห็นได้ว่ามีการกำหนดสมาชิกเป็น 7 หรือ 29 ซึ่งจะทำให้ได้รับกฎที่มีสมาชิกเป็น 7 ตัวเดียวในกฎหรือมี 29 ตัวเดียวในกฎ จึงเป็นเหตุที่ทำให้ได้ผลลัพธ์ที่มากกว่า และใช้เวลาประมวลผลนานกว่ากรณีของ AND

4.3 เปรียบเทียบผลการทดลองการค้นพบกฎในรูปแบบต่าง ๆ

การค้นหาความสัมพันธ์ในข้อมูลโดยการใช้เงื่อนไขในรูปแบบต่าง ๆ ทำให้ได้รับกฎที่มีจำนวนที่แตกต่างกันทั้งนี้ขึ้นอยู่กับรูปแบบของเงื่อนไข ในงานวิจัยนี้ได้ใช้ข้อมูลหมากรุก (Chess) ในการทดสอบและได้เปรียบเทียบกับทั้งสามโปรแกรม คือ โปรแกรม Apriori โปรแกรม ACIF และโปรแกรม ACAF การเปรียบเทียบจะใช้ตัวชี้วัด คือ เวลาในการประมวลผล (หน่วยเป็นวินาที) และจำนวนกฎความสัมพันธ์ โดยจะแสดงการเปรียบเทียบเวลาในตารางที่ 4.1 แสดงการเปรียบเทียบจำนวนกฎความสัมพันธ์ในตารางที่ 4.2 และในการใช้เงื่อนไขในรูปแบบต่าง ๆ มีคำอธิบายดังนี้

1) ไม่มีเงื่อนไข เป็นการกำหนดให้โปรแกรมไม่ใช้เงื่อนไขใดๆนอกเหนือจากเงื่อนไขของอัลกอริทึมเอไพรออริที่ให้ผู้ใช้ระบุค่า Minimum Support และ Minimum Confidence

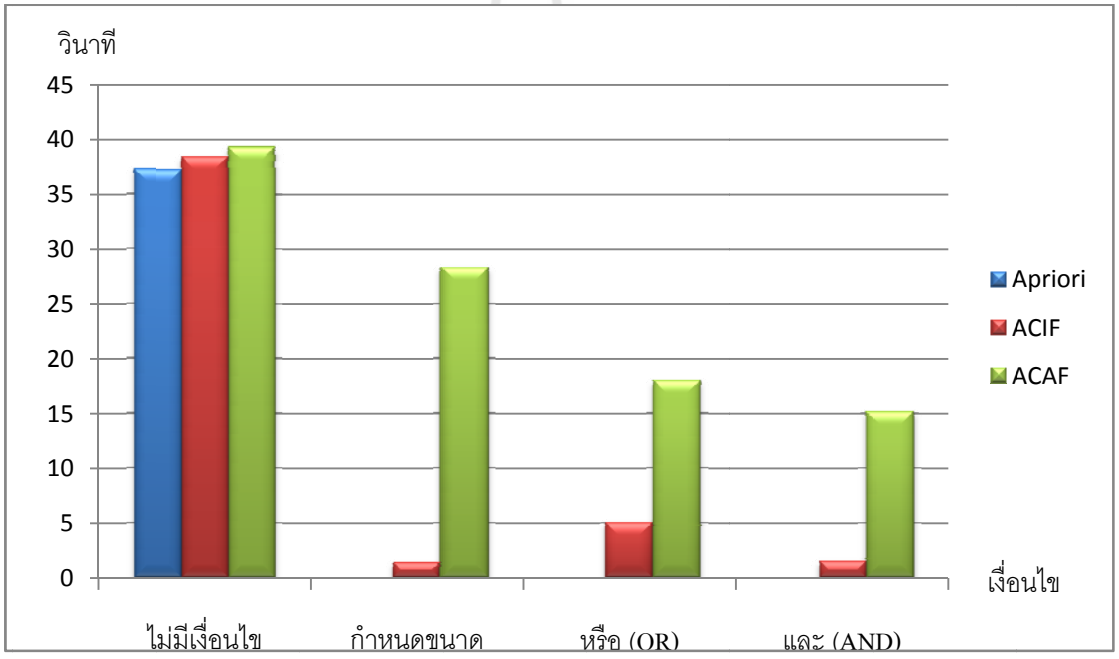
2) กำหนดขนาดของกฎ ในการทดสอบครั้งนี้ได้กำหนดให้ กฎที่ได้รับนั้นจะต้องมีขนาดสมาชิกมากกว่า 3 ไอเท็ม

3) ระบุไอเท็มด้วยเงื่อนไขหรือ (OR) ในการทดสอบได้กำหนดเงื่อนไขให้สมาชิกของกฎจะต้องมี 7 หรือ 8 และไม่มี 60 หรือ 61 เป็นสมาชิกในกฎ

4) ระบุไอเท็มด้วยเงื่อนไขและ (AND) ในการทดสอบได้กำหนดเงื่อนไขให้สมาชิกของกฎจะต้องมี 7 และ 29 และไม่มี 58 และ 40 เป็นสมาชิกในกฎ

ตารางที่ 4.1 เวลาในการประมวลผลของโปรแกรม Apriori ACIF และ ACAF

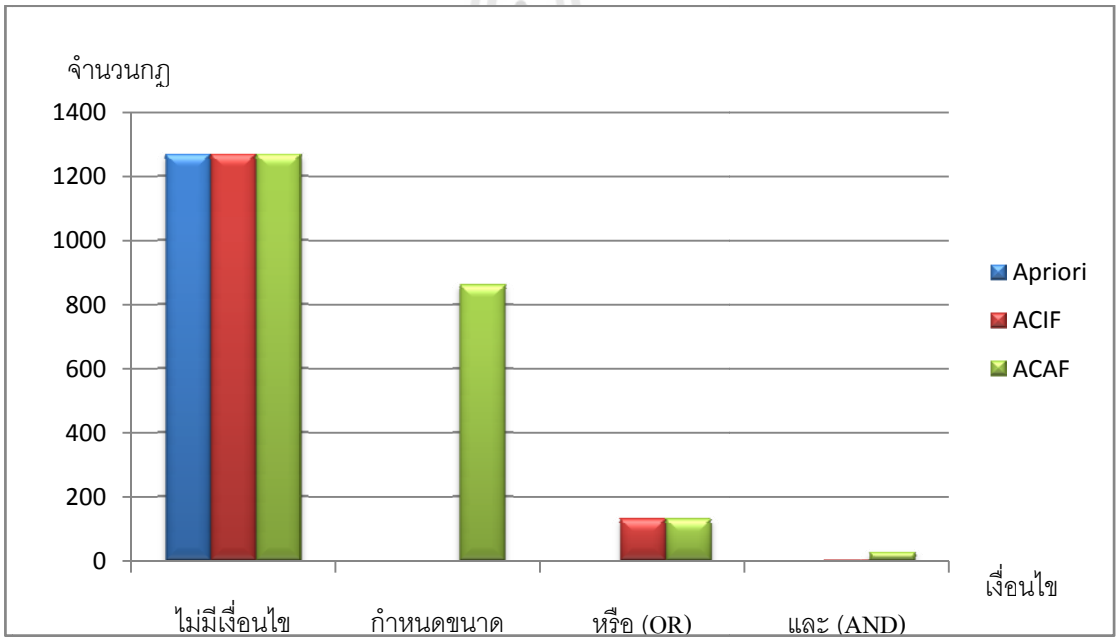
โปรแกรม	เวลาในการประมวลผลเมื่อมีการใช้เงื่อนไขรูปแบบต่าง ๆ			
	ไม่มีเงื่อนไข	กำหนดขนาดของกฎ	ระบุไอเท็มด้วยเงื่อนไขหรือ (OR)	ระบุไอเท็มด้วยเงื่อนไขและ (AND)
Apriori	37.29 วินาที	ไม่สามารถทำได้	ไม่สามารถทำได้	ไม่สามารถทำได้
ACIF	38.48 วินาที	1.45 วินาที	5.09 วินาที	1.56 วินาที
ACAF	39.35 วินาที	28.34 วินาที	18.05 วินาที	15.22 วินาที



รูปที่ 4.22 แผนภูมิแสดงการเปรียบเทียบเวลาในการประมวลผล
ของโปรแกรม Apriori ACIF และ ACAF

ตารางที่ 4.2 จำนวนกฎความสัมพันธ์ที่เป็นผลลัพธ์จากการประมวลผลโปรแกรม Apriori ACIF และ ACAF

โปรแกรม	การใช้เงื่อนไขรูปแบบต่าง ๆ เพื่อหากฎความสัมพันธ์			
	ไม่มีเงื่อนไข	กำหนดขนาดของกฎ	ระบุไอเท็มด้วยเงื่อนไขหรือ (OR)	ระบุไอเท็มด้วยเงื่อนไขและ (AND)
Apriori	1268 กฎ	ไม่สามารถทำได้	ไม่สามารถทำได้	ไม่สามารถทำได้
ACIF	1268 กฎ	0 กฎ	130 กฎ	2 กฎ
ACAF	1268 กฎ	862 กฎ	130 กฎ	28 กฎ



รูปที่ 4.23 แผนภูมิแสดงการเปรียบเทียบจำนวนกฎความสัมพันธ์ที่เป็นผลลัพธ์จากการประมวลผลโปรแกรม Apriori ACIF และ ACAF

4.4 อภิปรายผล

จากผลการทดสอบโปรแกรม Apriori ACIF และ ACAF ด้วยข้อมูล Chess จำนวน 3,196 รายการ และใช้รูปแบบการระบุเงื่อนไขหลายลักษณะสรุปผลการทดสอบเปรียบเทียบได้ดังนี้

1) รูปแบบไม่มีเงื่อนไข หมายถึง การค้นหากฎความสัมพันธ์ใช้เฉพาะเงื่อนไข Minimum Support และ Minimum Confidence เท่านั้น รูปแบบนี้โปรแกรมทั้งสามจะใช้เวลาที่ไม่ต่างกันมาก ทั้งนี้เพราะทั้งสามโปรแกรมมีพื้นฐานการทำงานที่เหมือนกันคือทำงานเหมือนอัลกอริทึมเอไพรออริอยู่แล้ว ซึ่งถ้าไม่ใช้เงื่อนไขในการหาความสัมพันธ์ของข้อมูลในการทำงานของทั้งสามโปรแกรมนี้ก็แทบจะไม่แตกต่างกันเลยและจำนวนกฎที่ได้รับก็มีจำนวนที่เท่ากันด้วย

2) รูปแบบกำหนดขนาดของกฎ การประมวลผลในรูปแบบนี้พบว่าโปรแกรมทั้งสามใช้เวลาที่แตกต่างกันมาก โปรแกรม Apriori ไม่มีฟังชันก์การกำหนดขนาดของกฎจึงไม่สามารถหากฎที่มีเฉพาะกฎที่กำหนดขนาดได้ ส่วนโปรแกรม ACIF นั้นได้ใช้เวลาเพียง 1.45 วินาที แต่ไม่พบกฎที่มีขนาดมากกว่า 3 ไอเท็มเพราะว่าโปรแกรมไม่สามารถหากฎความสัมพันธ์ที่มีขนาดมากกว่า 3 ได้ เนื่องจากในการใช้เงื่อนไขระหว่างการหาไอเท็มเซตปรากฏบ่อยนั้น ได้มีการเช็คขนาดแคนดิเดตไอเท็มเซตว่าเซตที่จะเป็นไอเท็มเซตปรากฏบ่อยนั้นจะต้องมีสมาชิกมากกว่า 3 ไอเท็ม ตัวอย่างการเช็คขนาดมีดังนี้ ในการหาไอเท็มเซตรอบแรกจะไม่มีการเช็คขนาดแต่พอถึงรอบสองจะมีการเช็คขนาด ซึ่งรอบสองนั้นจะประกอบด้วยแคนดิเดตไอเท็มเซตที่มีสมาชิก 2 ตัวและจะถูกนำไปเช็คกับเงื่อนไขว่าสมาชิกของแคนดิเดตจะต้องมากกว่า 3 ไอเท็ม ซึ่งในรอบนี้ไม่มีเซตไหนผ่านเงื่อนไขจึงไม่มีการสร้างไอเท็มเซตและแคนดิเดตไอเท็มเซตขนาด 3 ไอเท็มขึ้นไปเกิดขึ้นจึงทำให้จำนวนกฎที่ได้รับนั้นเป็นศูนย์ ในส่วนโปรแกรม ACAF นี้จะใช้เวลามากที่สุดเพราะการใช้เงื่อนไขกำหนดขนาดจะใช้หลังจากหาไอเท็มเซตปรากฏบ่อยทั้งหมดแล้วจะได้ไอเท็มเซตทุกขนาด จึงทำให้สามารถเช็คเงื่อนไขได้และจะใช้เวลามากกว่าโปรแกรมอื่นแต่จะได้กฎที่ต้องการครบถ้วน

3) รูปแบบระบุไอเท็มด้วยเงื่อนไขหรือ (OR) โปรแกรม Apriori ไม่สามารถประมวลผลเพราะว่าในโปรแกรมไม่สามารถใช้เงื่อนไขนี้ได้ จึงไม่สามารถหากฎที่มีเฉพาะกฎที่กำหนดเงื่อนไขไว้ได้ ส่วนโปรแกรม ACIF จะใช้เวลาค่อนข้างน้อยกว่า โปรแกรม ACAF เพราะโปรแกรม ACIF มีการใช้เงื่อนไขในระหว่างการหาไอเท็มเซตปรากฏบ่อยจึงทำให้การประมวลผลใช้เวลาน้อยกว่าการประมวลผลหลังจากการหาไอเท็มเซตปรากฏบ่อยด้วยโปรแกรม ACAF แต่กฎที่ได้ก็มีจำนวนที่เท่ากันและเหมือนกัน

4) รูปแบบระบุไอเท็มด้วยเงื่อนไขและ (AND) โปรแกรม Apriori ไม่สามารถใช้เงื่อนไขนี้ได้ ส่วนโปรแกรม ACIF จะใช้เวลาน้อยที่สุดจากทั้งสามโปรแกรมและจะได้ผลลัพธ์เป็นกฎความสัมพันธ์เพียง 2 กฎ ซึ่งกฎที่ได้รับนั้นจะได้รับกฎไม่ครบตามเงื่อนไขและกฎที่ได้รับมีขนาด 2 ไอเท็ม เป็นเพราะว่าในการใช้รูปแบบจะเป็นการเลือกไอเท็มเซตที่มีสมาชิกประกอบด้วยเงื่อนไขที่ผู้ใช้กำหนด เช่นถ้ามีแคนดิเดตไอเท็มเซตขนาด 2 ไอเท็มอยู่ 5 ชุด คือ {a, b}, {a, c}, {a, d}, {b, e}, {a, f} สมมุติผู้ใช้กำหนดเงื่อนไขที่ต้องการเป็น a และ b ไอเท็มเซตที่ผ่านเงื่อนไขก็คือ {a, b} เพียงชุดเดียว ซึ่งในการสร้างแคนดิเดตจำนวน 3 ไอเท็มไม่สามารถสร้างได้เพราะมีไอเท็มเซตที่ผ่านเงื่อนไขมาเพียงชุดเดียวจึงเป็นสาเหตุที่ทำให้

ได้รับกฎที่น้อยกว่าความเป็นจริง ซึ่งจะต่างจากโปรแกรม ACAF ที่จะใช้เวลาค่อนข้างมากแต่จะให้กฎที่ครบถ้วนตามเงื่อนไข

จากผลการทดสอบโปรแกรมทั้งสามโดยใช้เงื่อนไขต่าง ๆ และใช้ข้อมูลหมากรุก ผลการทดสอบสรุปได้ว่า โปรแกรม Apriori จะสามารถทดสอบได้เพียงรูปแบบเดียวคือรูปแบบไม่มีเงื่อนไขเพิ่มเติมจากเกณฑ์ Minimum Support และ Minimum Confidence ซึ่งถ้าผู้ใช้ต้องการแค่บางกฎก็จะใช้เวลาในการหากฎค่อนข้างมากเพราะผู้ใช้จะต้องค้นหาจากกฎความสัมพันธ์ทั้งหมดด้วยตัวเองหรือใช้โปรแกรมอื่นช่วยในการค้นหา ส่วนโปรแกรม ACIF ในการประมวลผลด้วยเงื่อนไขต่าง ๆ นี้จะใช้เวลาน้อยที่สุดแต่ก็อาจจะค้นพบกฎได้ไม่ครบถ้วน ซึ่งจะเห็นได้จากการใช้เงื่อนไขกำหนดขนาด การระบุไอเท็มด้วยเงื่อนไขหรือ (OR) และการระบุไอเท็มด้วยเงื่อนไขและ (AND) แต่โปรแกรม ACAF จะค้นพบกฎได้ครบถ้วนตามเงื่อนไขที่ผู้ใช้กำหนดแม้จะใช้เวลาในการค้นพบค่อนข้างมาก

บทที่ 5

สรุปผลการวิจัยและข้อเสนอแนะ

ปัจจุบันการค้นหารูปแบบของข้อมูลเพื่อช่วยในการสนับสนุนการตัดสินใจ หรือการค้นหาความสัมพันธ์ในข้อมูลเพื่อปรับมูลค่าในคลังสินค้า มีผู้นิยมเป็นจำนวนมาก แต่การค้นหาความรู้จากข้อมูลนั้นค่อนข้างยากและซับซ้อนเนื่องจากข้อมูลที่นำมาหารูปแบบนั้นมักจะมีจำนวนมาก จึงต้องใช้เวลาเตรียมข้อมูล และใช้เวลาในการค้นหาความรู้เป็นเวลานาน โดยในการค้นหาความสัมพันธ์ของข้อมูลนั้นจะทำให้ได้รับความรู้หรือกฎความสัมพันธ์เป็นจำนวนมาก ซึ่งจะยุ่งยากต่อการนำไปใช้เพราะจะต้องมาคัดแยกกฎที่เราต้องการนำไปใช้ออกจากกฎทั้งหมด ซึ่งในขั้นตอนคัดแยกอาจจะเกิดข้อผิดพลาดทำให้เสียกฎที่ต้องการไป หรือว่าได้กฎที่เกินจำเป็นนำไปใช้งาน

ในงานวิจัยนี้จึงมุ่งเน้นพัฒนากระบวนการในการออกแบบอัลกอริทึมและพัฒนาโปรแกรมเพื่อหากฎความสัมพันธ์จากข้อมูลตามที่ผู้ใช้กำหนด ซึ่งจะทำให้ผู้ใช้ได้รับกฎที่มีความจำเป็นต่อผู้ใช้และลดจำนวนของกฎที่ไม่จำเป็น ผู้ใช้สามารถกำหนดเงื่อนต่าง ๆ เพื่อให้ได้กฎตามที่ผู้ใช้ต้องการ เช่น การกำหนดสมาชิกที่ผู้ใช้ต้องการให้ปรากฏ การกำหนดสมาชิกของกฎให้ไม่ปรากฏไอเท็มที่ผู้ใช้ไม่ต้องการ การกำหนดขนาดของกฎ การกำหนดเป้าหมายของกฎ เป็นต้น ซึ่งปัจจัยที่ทำให้เวลาในการค้นพบและจำนวนกฎที่ลดลงก็คือ การใช้เงื่อนไขเพื่อลดขอบเขตการค้นหากฎที่จะเป็นคำตอบที่ผู้ใช้ต้องการ กระบวนการที่ใช้ค้นหากฎความสัมพันธ์ในงานวิจัยนี้ได้ใช้เทคนิคการทำเหมืองข้อมูลประเภทค้นหาความสัมพันธ์ของข้อมูล โดยใช้อัลกอริทึมที่มีความนิยมสูงคืออัลกอริทึมเอไพรออริโดยนำมาผสมผสานกับการใช้เงื่อนไขและในขั้นตอนการพัฒนาโปรแกรมได้ใช้วิธีการเขียนเชิงตรรกะด้วยเงื่อนไขบังคับ เนื่องจากเป็นเทคนิคการเขียนโปรแกรมที่สามารถระบุเงื่อนไขบังคับ เพื่อลดขนาดของขอบเขตการค้นหาคำตอบ

ขั้นตอนของงานวิจัยนี้แบ่งออกเป็น

1) การศึกษาการทำงานของอัลกอริทึมเอไพรออริ (Apriori) และการศึกษาการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ ซึ่งในงานวิจัยนี้ได้พัฒนาโปรแกรมโดยใช้รูปแบบโปรแกรมอีคลิปส์ (ECLiPSe) เป็นโปรแกรมที่เหมาะกับการหาคำตอบเชิงตรรกะที่มีการระบุเงื่อนไขโดเมนของตัวแปรและสามารถเขียนโปรแกรมเพื่อแก้โจทย์ทางคณิตศาสตร์ได้สะดวก

2) การออกแบบอัลกอริทึม ACIF และ ACAF ที่พัฒนาเพิ่มเติมจากอัลกอริทึม Apriori โดยได้เพิ่มเงื่อนไขบังคับที่สามารถกำหนดขนาดของกฎ และกำหนดรูปแบบของกฎเป็นลักษณะต่างๆ ตามที่ผู้ใช้ต้องการได้

3) การพัฒนาโปรแกรม Apriori ACIF และ ACAF โปรแกรม ACIF เป็นโปรแกรมที่มีการทำงานเหมือนอัลกอริทึมเอไพรออริ แต่จะมีการใช้เงื่อนไขในระหว่างการค้นหาไอเท็มเซตที่ปรากฏบ่อย ส่วนโปรแกรม ACAF เป็นโปรแกรมที่มีการทำงานเหมือนอัลกอริทึมเอไพรออริเช่นกัน แต่จะมีการใช้เงื่อนไขบังคับหลังจากการค้นหาไอเท็มเซตปรากฏบ่อยทั้งหมด

การทดสอบประสิทธิภาพของโปรแกรมทั้งสามโปรแกรมทำการทดสอบกับข้อมูลหมากรุก (Chess) ซึ่งเป็นชุดข้อมูลมาตรฐานโดยในการเปรียบเทียบการทำงานของสามโปรแกรม จะทำการเปรียบเทียบเวลาที่ใช้ในการประมวลผลและจำนวนกฎความสัมพันธ์ที่ได้รับว่ามีความถูกต้องครบถ้วนหรือไม่ โดยจะใช้การสอบถามด้วยเงื่อนไขในแบบต่าง ๆ เพื่อแสดงให้เห็นถึงความแตกต่างของกฎที่ได้รับตามแต่ละรูปแบบของการใช้เงื่อนไข

5.1 สรุปผลการวิจัย

จากผลการทดสอบการทำงานของทั้งสามโปรแกรม โดยทำการทดสอบด้วยการใช้เงื่อนไขแบบต่าง ๆ เพื่อระบุกฎความสัมพันธ์ที่ต้องการ ทำให้เห็นทั้งข้อแตกต่างและข้อที่เหมือนกันคือ ถ้าไม่ใช้เงื่อนไขเพิ่มเติมนอกเหนือจากการระบุ Minimum Support และ Minimum Confidence ในการค้นหาความสัมพันธ์ ทั้งสามโปรแกรมจะให้ผลลัพธ์เป็นกฎความสัมพันธ์ที่มีจำนวนกฎที่เท่ากันและเวลาในการประมวลผลไม่แตกต่างกันมาก แต่ถ้าหากใช้เงื่อนไขกำหนดขนาดของสมาชิกโปรแกรม Apriori จะไม่สามารถกำหนดขนาดของกฎได้และโปรแกรม ACIF จะสามารถกำหนดขนาดได้ที่เงื่อนไขสมาชิกที่ไม่เกิน 2 ไอเท็ม แต่ถ้ากำหนดขนาดมากกว่านั้นโปรแกรมไม่สามารถค้นหากฎได้ (ข้อสังเกตุข้อมูลหมากรุกเท่านั้น) ส่วนโปรแกรม ACAF จะให้กฎที่ครบถ้วนตามเงื่อนไขที่กำหนดขนาด แต่ถ้าใช้เงื่อนไขแบบหรือ (OR) โปรแกรม ACIF จะใช้เวลาในการค้นหากฎน้อยมากและให้กฎที่ครบถ้วน แต่โปรแกรม ACAF จะใช้เวลาค่อนข้างมากแต่จะได้รับกฎที่ครบถ้วนเช่นกัน และการใช้เงื่อนไขแบบสุดท้ายคือเงื่อนไขแบบและ (AND) โปรแกรม ACIF จะใช้เวลาน้อยแต่ให้กฎที่ถูกต้องไม่ครบถ้วน ส่วนโปรแกรม ACAF จะใช้เวลามากกว่าแต่จะให้กฎที่ครบถ้วน จึงสรุปได้ว่าโปรแกรม ACAF ถึงจะใช้เวลามากกว่าโปรแกรม ACIF แต่จะให้กฎความสัมพันธ์ที่ครบถ้วนและถูกต้อง

5.2 ปัญหาและข้อเสนอแนะ

ในขั้นตอนการค้นหากฎความสัมพันธ์เป็นการค้นหากฎความสัมพันธ์จากข้อมูลขนาดใหญ่ จึงทำให้ได้รับกฎความสัมพันธ์จำนวนมาก ทั้งนี้ถ้าใช้การสอบถามที่ทำให้ได้รับกฎมากเกินไป อาจจะทำให้โปรแกรมไม่สามารถค้นหากฎความสัมพันธ์ได้หมดและโปรแกรมจะหยุดทำงานในระหว่างการค้นหากฎความสัมพันธ์เพราะหน่วยความจำไม่พอ การแก้ไขอาจจะต้องปรับค่าการใช้หน่วยความจำของโปรแกรม ECLiPSe ให้มีค่าที่มากขึ้น หรืออาจจะต้องเพิ่มขนาดของหน่วยความจำให้มีขนาดมากขึ้น

ในอนาคตถ้านำจุดเด่นของอัลกอริทึม ACIF ซึ่งมีจุดเด่นในการประมวลผลที่รวดเร็ว และนำอัลกอริทึม ACAF ที่มีจุดเด่นในการประมวลผลที่ให้กฎความสัมพันธ์ครบถ้วนมาประยุกต์เข้าด้วยกัน อาจจะทำให้อัลกอริทึมใหม่นี้มีประสิทธิภาพการประมวลผลที่รวดเร็วและให้กฎความสัมพันธ์ที่ครบถ้วน

รายการอ้างอิง

กิตติศักดิ์ เกิดประสพ และ นิตยา เกิดประสพ. (2551). การค้นพบรูปแบบที่ปรากฏบ่อยด้วยวิธีการโปรแกรมเชิงประกาศ**. การประชุมวิชาการมหาวิทยาลัยเทคโนโลยีราชมงคลครั้งที่ 1** จังหวัดตรัง วันที่ 27-29 กันยายน.

ณัฏภัทรศญา ทับทิมเทศ. (2550). ระบบสนับสนุนการตัดสินใจ [ออนไลน์].ได้จาก **http://www.no-poor.com/dssandos/Chapter5-dss.htm**

ณัฐพล พันนุรัตน์. (2551). การปรับรูปแบบบรรทัดฐานในฐานข้อมูลเชิงสัมพันธ์ด้วยเทคนิคการวิเคราะห์ความสัมพันธ์. **วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรมหาบัณฑิต มหาวิทยาลัยเทคโนโลยีสุรนารี**.

นิตยา เกิดประสพ. (2554). การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ. **เอกสารประกอบการสอน วิชา 423429 Constraint Logic Programming**. มหาวิทยาลัยเทคโนโลยีสุรนารี.

บุญเสริม กิจศิริกุล. (2554). ปัญญาประดิษฐ์. [ออนไลน์]. **www.cp.eng.chula.ac.th /~boonserm /teaching/ai1.0.2.pdf**.

วิกิพีเดีย สารานุกรมเสรี. (2554). ภาษาโปรล็อก [ออนไลน์]. **http://th.wikipedia.org/wiki/ภาษาโปรล็อก**.

Agrawal, R., Imielinski, T., Swami, A. (1993). Mining association rules between sets of items in large databases. In **Proceedings of the ACM SIGMOD International Conference on Management of Data,** Washington.: 207-216.

Agrawal, R., Srikant, R. (1994). Fast algorithms for mining association rules. In **Proceedings of the International Conference on Very Large Data Bases**.: 487-499.

Bayardo, R.J., Agrawal, R., Gunopulos, D. (2000). Costraint-based rule mining in large, dense database. In **Proceedings of the Data Mining and Knowledge Discovery**.: 217-240.

Gallo, A., Esposito, R., Meo, R., Botta, M. (2005). Optimization of association rules extraction through exploitation of context dependent constraints. In **Proceedings of the 9th Conference on Advances in Artificial Intelligene (AI*AI'05)**.: 258-269.

Han, J., Kamber, M. (2000). **Data Mining: Concepts and Techniques.** Morgan Kaufmann Publishers.

Jeudy, B., Boulicaut, J. (2002). Costraint-based discovery and inductive query: application to association rule mining. In **Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery**.: 110-124.

Krzysztof R. A., Mark W. (2007). **Constraint Logic Programming Using ECLiPSe**. Cambridge University Press.

Ng, R.T., Lakshmanan, L.V.S., Han, J., Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. In **Proceedings of 1998 ACM SIGMOD International Conference on Management of Data**.: 13–24.

Simonis, H. (2008). **Constraint Logic Programming**. COSYTEC.

Seifer, J.W. (2004). CRS report for congress received through the CRS Web. **Congressional research service the library of congress**.

Squier, L. (2001). What is Data Mining? [Online]. **http://www.damancr.org/Library/2001 .11.14 -Laura% 20Squier.ppt**.

Srikant, R., Vu, Q., Agrawal, R. (1997). Mining association rules with item constraints. In **Proceedings of 1997 ACM KDD**.: 67–73.

Trifonov, T., Georgieva, T. (2009). Application for discovering the constraint-based association rules in an archive for unique Bulgarian bells. **European Journal of Scientific,** Vol.31, No.3.: 366-3.

ภาคผนวก ก

บทความวิชาการที่ได้รับการตีพิมพ์เผยแพร่

รายชื่อบทความวิชาการที่ได้รับการตีพิมพ์เผยแพร่

ไพชยนต์ ไชย, กิตติศักดิ์ เกิดประสพ และนิตยา เกิดประสพ. 2555. **Association Rule Discovery with Constraint Logic Programming**. ในการประชุมวิชาการ CIMMACS'12 The 11th International Conference on Computational Intelligence, Man_Machine System. And Cybernetics, ประเทศสิงคโปร์. 11 – 13 พฤษภาคม 2555.

Association Rule Discovery with Constraint Logic Programming

PHAICHAYON KONGCHAI, NITTAYA KERDPRASOP, and KITTISAK KERDPRASOP
Data Engineering Research Unit, School of Computer Engineering
Suranaree University of Technology
111 University Avenue, Nakhon Ratchasima 30000
THAILAND
Zaguraba_ii@hotmail.com, nittaya@sut.ac.th, kerdpras@sut.ac.th

Abstract: - Association rule discovery is one of a major data mining task that has gained much interest from researchers and general users. The knowledge from association mining can be used to recommend product, design catalogs and promotional management. But data processing for association rule discovery has expensive time because the relationship from data can be tremendously many more than other data mining tasks such as classification. As a consequence, most association mining software generally create so many rules from the association mining process and some of these rules are not beneficial to any users. To solve this problem, we propose to incorporate Apriori algorithm with constraint function for user to specify subset of association rules of interest items. Users can also identify length of the rules. Our Apriori-with-constraint algorithm can reduce processing time and reduce a great number of useless rules.

Key-Words: - Data mining, Association, Constraints Logic Programming

1 Introduction

Nowadays, data mining is popular as a result of data storage devices has greater capacity but less cost, therefore people can access and use these devices and grow the massive information. Data mining is used to find the patterns of information those are available in large databases [4]. Based on statistics, remember, machine learning and mathematical principles. The knowledge is gained from data mining in many forms such as Association Rule, Data Classification and Data Clustering. Derived knowledge from data mining can be used in prediction and decision.

Association rule will model data by finding the relationship of data in databases. In business, they are interested in the relationship of data. Because the derived knowledge from the association rule may be used for planning the product placement, recommended product or to stimulate the product sales and analysis of customer shopping [8]. However, Association rule has an expensive time in processing and deriving many disinterested rule.

In this paper proposes defining constraint approach, method for reducing time of information processing as well as disinterested rules. To define the constraint, the user can specify subset of association rule and define the length. Thus, programming using constraint logic programming will enhance performance of processing while reduce the line of code.

Paper overview. Section 2, summarizes related work, related to the constraint association rules. In Section 3 the theory of association rules, Apriori algorithm, constraint logic programming and integrated Apriori with constraints logic programming. In Section 4 compares the speed in processing with 3 programs: Original Apriori, Apriori with function, and Apriori with constraints logic programming and discussion. In Section 5 conclude with summary.

2 Related Work

A lot of work on mining rules from data is extensive. Many researchers are interested to find some idea for enhancement the process of association rule. In post-processing, many rules are returned from the mining association rule and the process uses a lot of execution time, Ramakrishnan Srikant, Quoc Vu and Rakesh Agrawal. [9] presented mining association rule with specify the constraint by the user to receive interested rule and reduce execution time. The execution time is reduced because dropped rules is disinterested (pruning). There are several papers present improvements by using query database technique for mining association rule with constraint-based [1,2,3]. It is much more efficient to incorporate such constraints into the associations.

Tihomir Trifonov and Tsvetanka Georgieva. [11] presented an application that discovers the

constraint-based association rules in an archive for Unique Bulgarian Bells. They used query database technique for mining association rule. The application can give association rules and implement with Java and SQL. With simple and fast method, the user can just specify subset of association rules which no need of understanding SQL or Java.

In this paper, we implemented the Apriori algorithm with constraint for mining association rule. For user can specify interested rules and reduce execution time.

3 Constrained Association Mining and Its Implementation

3.1 Association Rule

Association rule will find the pattern (rule) from large databases. The pattern has format; if...then... or if => then. For example,

Sandwich, Coke => Candy

or

if Sandwich, Coke then Candy

The rule means if the customer buys Sandwich and Coke then the customer must buys Candy as well. The mining association rule has two steps for creating rules from information [5].

Step 1, Find all frequent itemset which greater than or equal minimum support value. We can find the support value of the itemsets from all transactions.

$$support(A) = \frac{A}{Transaction}$$

Step 2, Generate rules with frequent 2-itemsets or more which greater than or equal minimum confidence value. We can find the confidence of the rules from formula.

$$confidence(A -> B) = \frac{support(A \cap B)}{support(A)}$$

Data mining discovers relationships in databases. The important step is to find all frequent itemsets. There are a lot of ideas and techniques in mining association rules. In this paper, we use Apriori algorithm for finding all frequent itemsets.

3.2 Apriori Algorithm

Rakesh Agrawal and Ramakrishnan Srikant (1994) have improved algorithms AIS that it works faster and renamed this algorithm to Apriori. The Apriori algorithm uses support-based technique for pruning (cut or reduce) the number of candidate itemsets instead of enumerates all itemsets. The principle of Apriori is if an itemset is a subset of the frequent Itemset then all subset must be frequent Itemset as well. For explanation, Apriori algorithm can be used to reduce the number of Candidate itemsets. Assume a itemset is composed of three Items {A,B,C} and the subsets are {A}, {B}, {C}, {A,B}, {A,C}, {B,C}. If {A,B,C} is frequent 3-Itemset then subset of {A,B,C} must be frequent Itemset.

Frequent 1-itemsets = {A}, {B}, {C}

Frequent 2-itemsets = {A,B}, {A,C}, {B,C}

We presented this algorithm for finding frequent itemset and integrated algorithm with constraint logic programming because it has high performance. However, processing of algorithm has expensive time and generates disinterested rules.

3.3 Constraint Logic Programming (CLP)

CLP is integrated logic programming and constraint solving, use high techniques from Operations Research and finite mathematics [6]. Ordinary logic programming is used for iteration, called recursion. Besides, CLP has special predicate such as for, foreach, fromto, labeling and alldifferent which makes programming easier. CLP can also solve mathematical equations.

```
:- lib(ic).              % include library
solve(R) :-
          R = [A, B],    % define variables
          R :: 1..5,     % define constraint
          A #> B,        % # is special symbol for ic
     alldifferent([B]), % [B] value not the same.
     labeling(R).
```

Fig.1 Example of CLP

In figure 1 shows format of CLP [7] and query with command solve(R). This program will solve R. We will explain 2 steps of solving, first step defines R = A, B which assign value 1 to 5. Second step compares A, B where A must greater than B and B must different from backtrack. The result is R = [2, 1]; [3, 1]; [3, 2]; [4, 1]; [4, 2]; [4, 3]; [5, 1]; [5, 2];

[5, 3]; [5, 4]. According to the example CLP can give all possible answer, coding shortly, more flexible and efficient.

3.4 Designed Apriori with Constraint Logic Programming

This research aims to develop data mining for association rule by the algorithm apriori with CLP.

In designing the algorithm Apriori with CLP, we use the main concept and additional part of constraint of the algorithm as well. The concept is shown as figure 2.

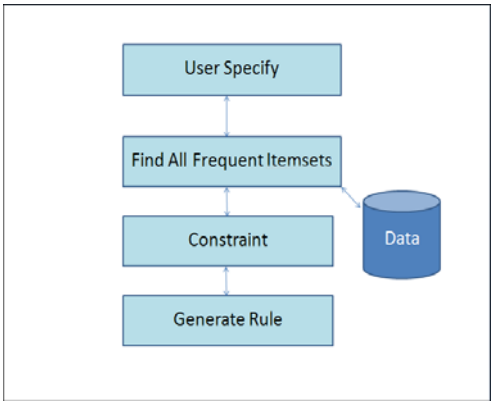


Fig.2 Framework of Apriori with constraint

- **User specify** the user must specify the minimum support, minimum Confidence and can specify subset in association rules.

- **Find all frequent itemsets** to find itemsets which greater than or equal minimum support.

- **Constraint** the conditions to be compared user-defined set with frequent itemsets. The set of user-defined is whether subset of frequent itemsets or not. Assume the user interested item {A,B} and disinterested item {D} from frequent itemsets {A,B}, {A,B,C,D}, {E,C}, {A,B,D} the result of constraint is {A,B}.

- **Generate Rule,** with frequent 2-itemsets or more which greater than or equal minimum confidence value. Suppose {A,B} is frequent 2-itemset and minimum confidence is 80%. {A,B} can generate rule A => B, confidence = 70% and B => A, confidence = 90% the result of generate rule is B => A.

3.5 Apriori with Constraint Logic Programming

The Apriori algorithm implements with CLP. CLP allows users to define items those are members of the association rules independently because condition is a mathematical.

This program is a logic programming. In preparing data must be provided in the format logic programming (Fig.3).

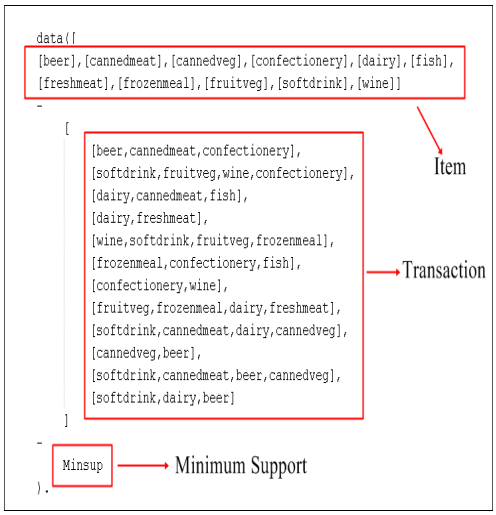


Fig.3 Preparing data in the format logic programming

- **Item** is to declare each item to the program to be tested. In Fig.3 the item consists of all 11 Items.

- **Transaction** is to declare each transaction to the program to be tested where each transaction must be in notation [ ]. In Fig.3 the transaction consists of all 12 transactions.

- **Minimum Support** is to define the minimum support or it can define at the dialog box when run the program**.**

To query the program for association rules must query in CLP format. In figure 4 shows query for finding association rules, where wine is subset, fruitveg is not subset in association rules, the length of the association rule rather than 1 item and minimum confidence is 100%.

```
association(R,ItemSet), ConS&::[wine], ConS1&::[fruitveg],
(foreach(Set,R), param(ConS,ConS1,ItemSet) do
    Set = X-_,length(X,LenItem),LenItem > 1,[Y,Y1]&::X,
    ConS&=Y,\+ConS1&=Y1 -> findRule([Set]-ItemSet-100),
    ! ; true
).
```

Fig.4 Query the program with CLP format

- association(R,ItemSet) is a predicate to find all frequent itemsets and store in variable R. ItemSet is variable to store each transaction to generate association rules.

- fineRule([Set]-ItemSet-100) is a predicate to generate association rules. [Set] is the result of constraint. 100 is minimum confidence.

- param(ConS,ConS1,ItemSet) is a variable declaration to import focused variable from outside the predicate foreach can be used inside.

- ConS&::[wine], ConS1&::[fruitveg] define variable ConS has a subset that is wine and variable ConS1 has a subset that is fruitveg.

- ConS&=Y is a constraint. Condition is true when ConS is subset of Y.

- \+ConS1&=Y1 is a constraint. Condition is true when ConS is not subset of Y.

- LenItem > 1 is the length of the association rule which more than 1 item.

- foreach(Set,R) is special predicate in CLP and take value from list R into variable Set.

- Set=X-_ is the pattern matching from logic programming. Because Set is format which consists of variable X that is frequent itemset and _ is anonymous variable.

When query the program with Fig.4 with using data from Fig.3 the program will show dialog box for define minimum support. In this query we define minimum support to 2. The result is shown in Fig 5.

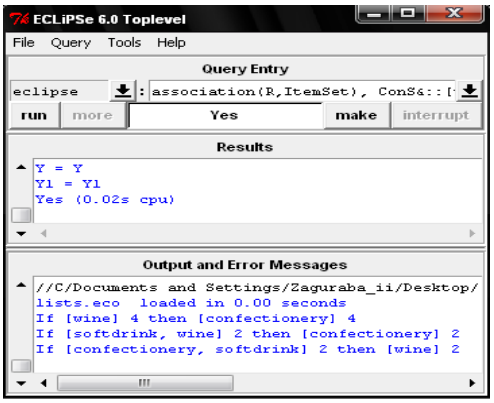


Fig.5 The result are association rules.

The result composed of three association rules. Wine is a member of the rules but fruitveg is not. From the first rule (If [wine] 4 then [confectionery] 4) means if four people buy wine then four people will buy candy together. But if run the program without the constraint there will be 24 association rules in the result.

4 Evaluation

This section provides an evaluation of Apriori with the Constraint Logic Programming that compared time of processing with the two programs. The first program is Original Apriori (working same as Apriori). The second program is Apriori with three functions.

1) The function for user to specify subset in association rules.

2) The function for user to specify not subset in association rules.

3) The function for user to specify the length of the association rules.

We use data-sets in figure 3. It consists of 12 transactions and duplicate transaction to 2,167, 32,530 and 63,890 transactions for observation the difference between them.

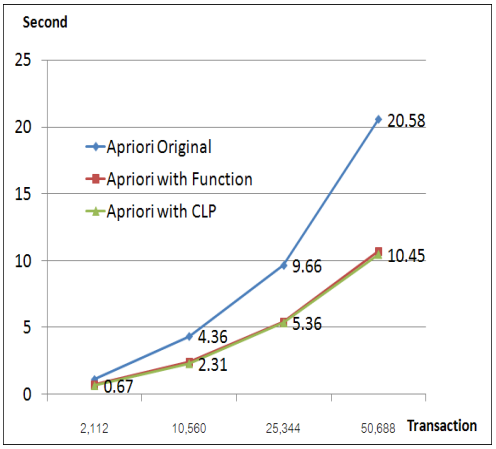


Fig.6 Execution time of three program.

In the figure 6 at 2,112 transactions execution time is not different. If the number of transaction in the both programs are over 10,560 the Original Apriori using execution time more than 50%. Because it generates 74 association rules but Apriori with Function and Apriori with CLP generate just 15 association rules. Apriori with Function and Apriori with CLP execution time are not different because both use same algorithm and pruning disinterested frequent itemsets. However, Apriori with Function is a static program and has the amount of code more than Apriori with CLP.

5 Conclusion

This paper present the algorithm Apriori with CLP. It can pruning disinterested frequent itemsets, the user can define subset and lenght in association rules that results to reduce execution time of processing. In section Evaluation, Original Apriori use execution time of processing more than Apriori with CLP and Apriori with Function. Because Apriori Original not pruning disinterested frequent itemsets. Apriori with Function is a static program because we have to edit its condition in source code when we want to but Apriori with CLP can edit at query command.

References:

[1] Agrawal, R., Imielinski, T., Swami, A. Mining association rules between sets of items in large databases, In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, 1993, pp. 207-216.

[2] Agrawal, R., Srikant, R. Fast algorithms for mining association rules, In *Proceedings of the International Conference on very Large Databases*, 1994, pp. 487-499.

[3] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I. Fast discovery of association rules, In *Proceedings of the Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996, pp. 327-328.

[4] Bayardo, R., Agrawal, R., Gunopulos, D. (2000). Constraint-based rule mining in large, dense databases, In *Proceedings of the 15th Int'l Conf. on Data Engineering*,1999, pp. 188-197.

[5] Jeudy, B., Boulicaut, J. Costraint-Based discovery and inductive query: application to association rule mining, *Pattern Detection and Discovery*, 2002, pp. 110-124.

[6] Kerdprasop, K., Kerdprasop, N. Frequent pattern discovery with declarative programming, In *Proceedings of the 1st Rajamangala University of Technology Conference*, Trang, Thailand, August 27-29, 2006.

[7] Kerdprasop, N. *Constraint Logic Programming*, [online:https://sites.google.com/site/nittayak/home/clp], access on: December 10, 2011.

[8] Ng, R.T., Lakshmanan, L.V.S., Han, J., Pang, A. Exploratory mining and pruning optimizations of constrained associations rules, In *Proceedings of 1998 ACM SIGMOD Int. Conf. Management of Data*, pp. 13–24.

[9] Srikant, R., Vu, Q., Agrawal, R. Mining association rules with item constraints, In *Proceedings of 1997 ACM KDD*. pp. 67–73, 1977.

[10] Simonis, H. *Constraint logic programming*, COSYTEC SA, 2008.

[11] Trifonov, T., Georgieva, T. Application for Discovering the Constraint-Based Association Rules in an Archive for Unique Bulgarian Bells. *European Journal of Scientific Vol.31, No.3*. 2009, pp. 366-371.

ภาคผนวก ข

รหัสต้นฉบับของโปรแกรม Apriori, ACIF, ACAF

โปรแกรม Apriori

```
% association(R,2,100).        % <-- main module- fix 6 loops.
:-lib(ordset).
:- compile("filename.txt"). % load file.

association(R,MinSup,Conf) :- data(Data),Data=_-_,
        ( count(I,2,6), fromto(Data, S0,S1,R),param(MinSup,Conf) do
          ( S0=A-B,
                findCL(A-B-MinSup,R-_-_,Conf),
                allUnion(I,R,NewItemSet),
                S1=NewItemSet-B ),!
        ).

% findCL([[a],[b],[c],[d],[e]]-[[a,c,d],[b,c,e],[a,b,c,e],[b,e]]-2 ,I).
findCL(ItemSet-Items-MinSup,R-Items-MinSup,Conf) :-
    (foreach(X,ItemSet), fromto(R,S1,S0,[]), param(Items,MinSup) do
        (supOK(X,Items,MinSup,LenItem) -> S1=[X-LenItem|S0], ! ; S1=S0)
     ),findRule(R-Items-Conf).

% supOK([a],[[a, c, d],[b, c, e],[a, b, c, e],[b, e]],2,L) .
supOK(X,Items,MinSup,LenItem) :-
            (foreach(I,Items), fromto(R,S1,S0,[]), param(X) do
                (my_subset(X,I) -> S1 = [ I | S0], ! ; S1=S0)
             ),
            length(R,LenItem),
            LenItem >= MinSup.
```

```
% findRule([[b, c, e]-2]-[[a,c,d],[b,c,e],[a,b,c,e],[b, e]]-100).
findRule(ItemSet-Items-MinConf) :- ItemSet = 0 -> ! ;
    (foreach(Set,ItemSet), param(Items,MinConf) do
          Set = X-LenItem,
          findall(Re,powerset(X,Re),PwSet),
          (foreach(ItemX,PwSet), param(X,Items,LenItem,MinConf) do
                ( ItemX = X ; ItemX = [] -> ! ;
                supOK(ItemX,Items,0,LenItemX),
                conOk(LenItem-LenItemX-MinConf) -> createRule(ItemX,X,Re) ,
                write('If '),write(ItemX),write(' '),write(LenItemX),
                write(' then '),write(Re),write(' '),writeln(LenItem),! ; !
                )
           )
     ).

% createRule([a],[a,b,c],Result).
createRule([],X,X).
createRule([H|Tr],X,Result) :- delete(H,X,Re),createRule(Tr,Re,Result).

conOk(LenItem-LenItemX-MinConf) :- Re is (LenItem/LenItemX)*100,Re >= MinConf .

% my_subset compare([1],[1,2]) return T or F
my_subset(Sub,S) :- toSet(Sub,OrdSub),  toSet(S,OrdS), ord_subset(OrdSub,OrdS).

% allUnion(2,[[a]-2,[b]-3,[c]-3,[e]-3],R).
allUnion(I,ItemSet,NewItemSet) :-  combi(ItemSet,R_), flatten(R_,R),
          (foreach(X,R), fromto(NewItemSet_,S1,S0,[]), param(I) do
               First-Sec=X,
```

```
            (unionN(I,First-Sec,Out) -> S1=[Out|S0], !;S1=S0)
        ),
        toSet(NewItemSet_,NewItemSet).
unionN(N,First-Sec,Out) :- toSet(First,F), toSet(Sec,S),
                           ord_union(F,S,Out),
                           length(Out,Len),Len=N.

% combi([[a]-2,[b]-2,[c]-2,[e]-2],R).  R = [a-b,a-c,a-e,...]
combi([],[]).
combi([H|T],[HR|TR]) :- (foreach(X,T), foreach(Y,HR), param(H) do
                           X = Set2-_,H = Set1-_ ,  Y=Set1-Set2 ),
                         combi(T,TR).
toSet(X,S) :- list_to_ord_set(X,S).

% powerset([a,b],X).
powerset([],[]).
powerset([_|Rest],L) :- powerset(Rest,L).
powerset([X|Rest],[X|L]) :- powerset(Rest,L).
```

โปรแกรม ACIF

```
% association(R,0,[[]],[[]],2,100).        % <-- main module- fix 6 loops.
:-lib(ordset).
:- compile("filename.txt"). % load file.

association(R,LengthI,Subset,NotSubset,MinSup,Conf) :- data(Data),Data=_-_,
  ( count(I,2,6), fromto(Data,S0,S1,R),param(MinSup,LengthI,Subset,Conf,NotSubset) do
        ( S0=A-B,
            findCL(A-B-MinSup,R-_-_,LengthI,Subset,NotSubset,Conf),
            allUnion(I,R,NewItemSet),
            S1=NewItemSet-B ),!
   ).

% findCL([[a],[b],[c],[d],[e]]-[[a,c,d],[b,c,e],[a,b,c,e],[b, e]]-2,R-[[a,c,d],
% [b,c,e],[a,b,c,e],[b, e]]-2,0,[[]],[[]],100).
findCL(ItemSet-Items-MinSup,R-Items-MinSup,LengthI,Subset,NotSubset,Conf) :-
          ItemSet = [H|_],length(H,LenItem),
          LenItem =1 -> findSubOk(ItemSet,Items,MinSup,R,_) ;
    ( findLength(LengthI,ItemSet,ItemSet1),
                          findSubset(ItemSet1,R1,Subset),
                          findSubset(ItemSet1,R1,Subset),
                          findNotSubset(R1,R2,NotSubset),
                          findSubOk(R2,Items,MinSup,R,LenItem1)
     ),findRule(R-Items-Conf,LenItem1).
```

```
% findLength(0,[[a,b],[a,c],[a,e],[b,c],[b,e],[c,e]],R).
findLength(Cons,Re,R):-
    (foreach(I,Re), fromto(R,S1,S0,[]), param(Cons) do
        length(I,LenItem),LenItem > Cons -> S1 = [ I | S0], ! ; S1=S0
    ).

% findSubOk([[a],[b],[c],[d],[e]],[[a,c,d],[b,c,e],[a,b,c,e], [b,e]],2,R,R1).
findSubOk(R2,Items,MinSup,R,R1):-
        (foreach(X,R2), fromto(R,S1,S0,[]),fromto(R1,S3,S2,[]), param(Items,MinSup) do
                supOK(X,Items,MinSup,Len) -> S1=[X|S0],S3 = [Len|S2],
                ! ; S1=S0, S3=S2
    ).

% findSubset([[a,c],[a,b]],R1,[[a],[b]]).
findSubset(X,X,[]).
findSubset(ItemSet,R1,[Subset|Tr]):-
                        Subset = [] -> R1 = ItemSet ;
        (foreach(X,ItemSet), fromto(R,S1,S0,[]), param(Subset) do
                intersection(Subset,X,ReSub),ReSub\=[] -> S1=[X|S0],  ! ; S1=S0
        ),findSubset(R,R1,Tr).

% findNotSubset([[a,c],[a,b]],R1,[[a],[b]]).
findNotSubset(X,X,[]).
findNotSubset(ItemSet,R1,[NotSubset|Tr]):-
                        NotSubset = [] -> R1 = ItemSet ;
        (foreach(X,ItemSet), fromto(R,S1,S0,[]), param(NotSubset) do
                intersection(NotSubset,X,ReSub),ReSub=[] -> S1=[X|S0],  ! ; S1=S0
        ),findNotSubset(R,R1,Tr).
```

```
% supOK([a],[[a, c, d],[b, c, e],[a, b, c, e],[b, e]],2,L) .
supOK(X,Items,MinSup,LenItem) :-
            (foreach(I,Items), fromto(R,S1,S0,[]), param(X) do
                (my_subset(X,I) -> S1 = [ I | S0], ! ; S1=S0)
             ),
            length(R,LenItem),
            LenItem >= MinSup.

% findRule([[a,c],[b,c],[b,e],[c,e]]-[[a,c,d],[b,c,e],[a,b,c,e],[b,e]]-100,[2,2,3,2]).
findRule([]-_-_,[]).
findRule([X|ItemSet]-Items-MinConf,[LenItem|LenItem1]) :- ItemSet = 0 -> ! ;
          findall(Re,powerset(X,Re),PwSet),
          (foreach(ItemX,PwSet), param(X,Items,LenItem,MinConf) do
                  ( ItemX = X ; ItemX = [] -> ! ;
                     supOK(ItemX,Items,0,LenItemX),
                     conOk(LenItem-LenItemX-MinConf) -> createRule(ItemX,X,Re) ,
                    write('If '),write(ItemX),
                    write(' '),write(LenItemX),write(' then '),write(Re),
                    write(' '),writeln(LenItem),! ; !
                  )
                          ),findRule(ItemSet-Items-MinConf,LenItem1).

% createRule([a],[a,b,c],Result).
createRule([],X,X).
createRule([H|Tr],X,Result) :- delete(H,X,Re),createRule(Tr,Re,Result).

conOk(LenItem-LenItemX-MinConf) :- Re is (LenItem/LenItemX)*100,Re >= MinConf .
```

```
% my_subset compare([1],[1,2]) return T or F
my_subset(Sub,S) :- toSet(Sub,OrdSub), toSet(S,OrdS),
                              ord_subset(OrdSub,OrdS).

 % allUnion(2,[[1],[2],[3]],[[1,2],[1,3],[2,3]]).
allUnion(I,ItemSet,NewItemSet) :-  combi(ItemSet,R_), flatten(R_,R),
          (foreach(X,R), fromto(NewItemSet_,S1,S0,[]), param(I) do
               First-Sec=X,
               (unionN(I,First-Sec,Out) -> S1=[Out|S0], !;S1=S0)
           ),
           toSet(NewItemSet_,NewItemSet).
unionN(N,First-Sec,Out) :- toSet(First,F), toSet(Sec,S),
                                  ord_union(F,S,Out),
                                  length(Out,Len),Len=N.


%combi([[a],[b],[c],[e]],P)).  P = [a-b,a-c,a-e,...]
combi([],[]).
combi([H|T],[HR|TR]) :- (foreach(X,T), foreach(Y,HR), param(H) do
                                    Y=H-X ),
                               combi(T,TR).
toSet(X,S) :- list_to_ord_set(X,S).

% powerset([a,b],X).
powerset([],[]).
powerset([_|Rest],L) :- powerset(Rest,L).
powerset([X|Rest],[X|L]) :- powerset(Rest,L).
```

โปรแกรม ACAF

```
% association(R,0,2,100)  % <-- main module- fix 6 loops.
:- compile("filename.txt"). % load file.
:- lib(sd).
:- lib(ic).
:- lib(ordset).
:- writeln('This is the Mining Association Rule.'),
  writeln('You can queries by ?- association(R,Length,MinSup,Conf).').

association(R,LengthI,MinSup,Conf) :-
        writeln('Please specify member in [[_]] :'),read(Subset),
        writeln('Please specify member do not need in [[_]] :'),read(NotSubset),
        writeln('Please specify member in [[_]] :'),read(Goal),
        data(Data),Data=_-_,
( count(I,2,6), fromto(Data, S0,S1,R),param(LengthI,Subset,NotSubset,Conf,MinSup,Goal)
        Do  ( S0=A-B,
              findCL(A-B-MinSup,R-_-_,LengthI,Subset,NotSubset,Conf,Goal),
              allUnion(I,R,NewItemSet),
              S1=NewItemSet-B ),!       ).
    ).

% findCL([[a],[b],[c],[d],[e]]-[[a, c, d],[b, c, e],[a, b, c, e],[b, e]]- 2,
% R-[[a, c, d],[b, c, e],[a, b, c, e],[b, e]]- 2,0,[[]],[[]],100,[[]]).
findCL(ItemSet-Items-MinSup,R-Items-MinSup,LengthI,Subset,NotSubset,Conf,Goal) :-
        (foreach(X,ItemSet), fromto(R,S1,S0,[]), param(Items,MinSup)    do
        (supOK(X,Items,MinSup,LenItem) -> S1=[X-LenItem|S0], ! ; S1=S0)
         ),        not1Item(R,Re1),
```

```
                        findLength(LengthI,Re1,Re),
                        findSubset(Subset,Re,R1),
                        findNotSubset(NotSubset,R1,R2),
                        findRule(R2-Items-Conf,Goal).

% Check Item set length > 1
not1Item(R,Re):- R=[H-_|_],length(H,LenItem),LenItem =1 -> Re = [] ; Re = R .

% findLength(0,[[a,b]-2,[a,c]-2],R).
findLength(Cons,Re,R):-
        (foreach(I,Re), fromto(R,S1,S0,[]), param(Cons) do
            I = X-_, length(X,LenItem),LenItem > Cons -> S1 = [ I | S0], ! ; S1=S0
         ).

% findSubset([[b],[c]],[[a,b]-2,[b,c]-2,[b,c,d]-2],R1).
findSubset([],X, X).
findSubset([Subset|Tr],Re,R1):-
        Subset = [] -> R1 = Re ;
        (foreach(I,Re), fromto(R,S1,S0,[]), param(Subset)    do
                I = X-_,
                Subset1&::Subset, Y&::X, Subset1&=Y -> S1 = [ I | S0], ! ; S1=S0
         ),     findSubset(Tr,R,R1).

% findNotSubset([[b],[c]],[[a,b]-2,[b,c]-2,[b,c,d]-2],R1).
findNotSubset([],X, X).
findNotSubset([NotSubset|Tr],Re,R1):-
        NotSubset = [] -> R1 = Re ;
        (foreach(I,Re), fromto(R,S1,S0,[]), param(NotSubset) do
```

```
                I = X-_,
        NotSubset1&::NotSubset, Y&::X, \+NotSubset1&=Y -> S1 = [ I | S0], ! ; S1=S0
            ),      findNotSubset(Tr,R,R1).

% findGoal([[a]],[a,b],R).
findGoal([],X, X).
findGoal([Subset|Tr],Re,R1):-
        Subset = [] -> R1 = Re ;
        Subset1&::Subset, Y&::Re, Subset1&=Y -> R1 = Re, findGoal(Tr,Re,R1).


% supOK([a],[[a,c,d],[b,c,e],[a,b,c,e],[b,e]],2,L) .
supOK(X,Items,MinSup,LenItem) :-
          (foreach(I,Items), fromto(R,S1,S0,[]), param(X) do
                (my_subset(X,I) -> S1 = [ I | S0], ! ; S1=S0)
           ),
                 length(R,LenItem),
                 LenItem >= MinSup.

% findRule([[b,c,e] - 2]-[[a,c,d],[b,c,e],[a,b,c,e],[b,e]]-100,[[b]]).
findRule(ItemSet-Items-MinConf,Goal) :-
            (foreach(Set,ItemSet), param(Items,MinConf,Goal) do
                        Set = X-LenItem,
                        findall(Re,powerset(X,Re),PwSet),
        (foreach(ItemX,PwSet), param(X,Items,LenItem,MinConf,Goal) do
                ( ItemX = X ; ItemX = [] -> ! ;
                createRule(ItemX,X,Re),findGoal(Goal,Re,R1) ->
                supOK(ItemX,Items,0,LenItemX),
                conOk(LenItem-LenItemX-MinConf,ItemX,R1),! ; !
                        )
```

```
                    )
    ).

% createRule([a],[a,b,c],Result).
createRule([],X,X).
createRule([H|Tr],X,Result) :- delete(H,X,Re),createRule(Tr,Re,Result).

% Check Confident
conOk(LenItem-LenItemX-MinConf,ItemX,Re) :-
          Re11 is (LenItem/LenItemX)*100,
          Re11 >= MinConf -> write('If '),
          write(Re),write(' '),writeln(LenItem) ;! .
          write(ItemX),write(' '),write(LenItemX),write(' then '),
          write(Re),write(' '),writeln(LenItem) ;! .

% my_subset compare([1],[1,2]) return T or F
my_subset(Sub,S) :- toSet(Sub,OrdSub), toSet(S,OrdS),
                        ord_subset(OrdSub,OrdS).

%allUnion(2,[[1],[2],[3]],[[1,2],[1,3],[2,3]]).
allUnion(I,ItemSet,NewItemSet) :-  combi(ItemSet,R_), flatten(R_,R),
          (foreach(X,R), fromto(NewItemSet_,S1,S0,[]), param(I) do
            First-Sec=X,
          (unionN(I,First-Sec,Out) -> S1=[Out|S0], !;S1=S0)
          ),
         toSet(NewItemSet_,NewItemSet).
unionN(N,First-Sec,Out) :- toSet(First,F), toSet(Sec,S),
          ord_union(F,S,Out),
          length(Out,Len),Len=N.
```

```
% combi([[a]-2,[b]-2,[c]-2,[e]-2],R).  P = [a-b,a-c,a-e,...]
combi([],[]).
combi([H|T],[HR|TR]) :- (foreach(X,T), foreach(Y,HR), param(H) do
        X = Set2-_,H = Set1-_ ,  Y=Set1-Set2 ),
         combi(T,TR).
toSet(X,S) :- list_to_ord_set(X,S).

% powerset([a,b],X).
powerset([],[]).
powerset([_|Rest],L) :- powerset(Rest,L).
powerset([X|Rest],[X|L]) :- powerset(Rest,L).
```

ประวัติผู้เขียน

นายไพชยนต์ คงไชย เกิดเมื่อวันที่ 27 มิถุนายน พ.ศ. 2531 ที่ อำเภอเมือง จังหวัดอำนาจเจริญ เริ่มเข้าศึกษาระดับชั้นอนุบาล 1 ที่โรงเรียนอนุบาลนพเก้า อำเภอเมือง จังหวัดอำนาจเจริญ หลังจากนั้นได้ย้ายไปศึกษาต่อในระดับชั้นอนุบาล 2 ถึงชั้นประถมปีที่ 6 ที่โรงเรียนอนุบาลอำนาจเจริญ อำเภอเมือง จังหวัดอำนาจเจริญ จากนั้นได้เข้าศึกษาต่อในระดับมัธยมศึกษาตอนต้นและตอนปลาย ที่โรงเรียนอำนาจเจริญ อำเภอเมือง จังหวัดอำนาจเจริญ ปีการศึกษา 2553 ได้เข้าศึกษาต่อระดับปริญญาตรีในสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี และสำเร็จการศึกษาเมื่อปี พ.ศ. 2553 ภายหลังสำเร็จการศึกษาในระดับปริญญาตรี ได้เข้าศึกษาในระดับปริญญาโท สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปี 2554

ในระหว่างการศึกษาได้รับความอนุเคราะห์อย่างดียิ่งจากจากอาจารย์ประจำวิชา Database System ได้รับความไว้วางใจให้เป็นผู้ช่วยสอนปฏิบัติการ ได้รับความอนุเคราะห์จากสำนักวิชาวิศวกรรมศาสตร์ให้เป็นครูสอนพิเศษ (Tutor) ในรายวิชา Computer Programming และได้รับการตีพิมพ์เผยแพร่บทความวิชาการซึ่งรายละเอียดสามารถดูได้ที่ภาคผนวก ก