รหัสโครงการ SUT7-705-54-36-32

รายงานการวิจัย

การประมวลผลข้อคำถามเพื่อค้นหารูปแบบที่ปรากฏบ่อย k อันดับแรกด้วยการโปรแกรมแบบมีเงื่อนไข

(Query Evaluation for Top-k Frequent Pattern Mining with Constraint Programming)

ได้รับทุนอุดหนุนการวิจัยจาก
มหาวิทยาลัยเทคโนโลยีสุรนารี

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

รหัสโครงการ SUT7-705-54-36-32

รายงานการวิจัย

การประมวลผลข้อคำถามเพื่อค้นหารูปแบบที่ปรากฏบ่อย k อันดับแรกด้วยการโปรแกรมแบบมีเงื่อนไข

(Query Evaluation for Top-k Frequent Pattern Mining with Constraint Programming)

ผู้วิจัย

รองศาสตราจารย์ ดร.นิตยา เกิดประสพ **หัวหน้าโครงการ**

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ **ผู้วิจัยร่วม**

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2554 2555 และ 2556

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

สิงหาคม 2558

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณมหาวิทยาลัยเทคโนโลยีสุรนารีและสำนักงานคณะกรรมการวิจัยแห่งชาติ ที่ได้จัดสรรงบประมาณให้ในปีงบประมาณ 2554 2555 และ 2556 เพื่อสนับสนุนโครงการวิจัยนี้ ขอขอบคุณผู้ทรงคุณวุฒิทั้งภายนอกและภายในมหาวิทยาลัย ที่ได้เสียสละเวลาทำหน้าที่ตรวจข้อเสนอโครงการวิจัยและร่างรายงานการวิจัยฉบับสมบูรณ์ ข้อเสนอแนะจากผู้ทรงคุณวุฒิทุกท่านเป็นประโยชน์อย่างมากต่อการปรับปรุงโครงการวิจัย งานวิจัยนี้สำเร็จได้อย่างดีด้วยการมีส่วนร่วมจากนักศึกษาทั้งในระดับปริญญามหาบัณฑิตและปริญญาดุษฎีบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ได้ทำหน้าที่เป็นผู้ช่วยวิจัยในโครงการวิจัยนี้ โดยเฉพาะนายไพชยนต์ คงไชย นางสาวฝนทิพย์ คูณแก้ว และนายกีระชาติ สุขสุทธิ์ นักศึกษาบัณฑิตสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ได้นำแนวคิดของโครงการวิจัยนี้ไปประยุกต์เป็นหัวข้อวิทยานิพนธ์

บทคัดย่อภาษาไทย

การค้นพบรูปแบบที่ปรากฏบ่อยเป็นกระบวนการค้นหาแพทเทิร์นที่ปรากฏซ้ำ ๆ ในข้อมูล แพทเทิร์นเหล่านี้มีประโยชน์ในการระบุความสัมพันธ์ที่ซ่อนอยู่ในกลุ่มข้อมูล อัลกอริทึมค้นหารูปแบบที่ปรากฏบ่อยมักจะได้รับการพัฒนาเป็นโปรแกรมในลักษณะของการโปรแกรมเชิงกระบวนคำสั่ง ซึ่งถ้ารูปแบบข้อมูลหรือแพทเทิร์นมีความซับซ้อนหรือมีขนาดที่ยาวมาก วิธีการโปรแกรมเชิงกระบวนคำสั่งจะมีประสิทธิภาพด้อยลง ผู้วิจัยจึงได้เสนอแนวทางของการโปรแกรมเชิงประกาศแบบมีเงื่อนไขบังคับซึ่งจะมีประสิทธิภาพสูงกว่า โดยได้ใช้ภาษาโปรล็อกเป็นพื้นฐานในการพัฒนาโปรแกรมและประมวลผลด้วยซอฟต์แวร์ ECLiPSe ซึ่งมีระบบประมวลผลเงื่อนไขบังคับ ทำให้การค้นหารูปแบบที่ปรากฏบ่อยสามารถทำได้ง่ายขึ้นกว่าวิธีการโปรแกรมแบบอื่น นอกจากนี้การตั้งข้อคำถามเพื่อสอบถามรูปแบบที่ปรากฏบ่อยในฐานข้อมูล ยังได้รับการพัฒนาให้ผู้ใช้สอบถามด้วยเงื่อนไขหลายลักษณะ ทั้งการสอบถามรูปแบบที่มีค่าความเชื่อมั่นและค่าสนับสนุนสูงสุด เค อันดับแรก และการระบุผลลัพธ์ที่ต้องการให้ปรากฏหรือไม่ปรากฏบางไอเท็มได้ ความสะดวกในการตั้งข้อคำถามที่ตรงกับความสนใจของผู้ใช้จะช่วยให้การค้นหาความสัมพันธ์ที่ปรากฏบ่อยเป็นประโยชน์ต่อผู้ใช้มากขึ้น

บทคัดย่อภาษาอังกฤษ

The problem of frequent pattern discovery is defined as the process of searching for patterns such as sets of features or items that appear in data frequently. Finding such frequent patterns has become an important data mining task because it reveals associations, correlations, and many other interesting relationships hidden in a database. Most of the proposed frequent pattern mining algorithms have been implemented with imperative programming languages. Such paradigm is inefficient when set of patterns is large and the frequent pattern is long. We suggest a high-level declarative style of programming apply to the problem of frequent pattern discovery. We consider the constraint logic programming language: ECLiPSe. Our intuitive idea is that the problem of finding frequent patterns should be efficiently and concisely implemented via a declarative paradigm with constraint processing facility since pattern matching is a fundamental feature supported by most logic programming languages. Our frequent pattern mining implementation using the Prolog language with ECLiPSe system confirms our hypothesis about conciseness of the program. Moreover, in this research querying to extract frequent patterns has been designed to facilitate ease of use. Users can pose query with several styles of constraints including finding the top-k relationships, when association rules are ranked in descending order in terms of their confidence and support scores, and specifying the items that users wish to include/exclude. These facilities of constraint based querying are expected to help users extracting the patterns most related to their interest.

สารบัญ

	หน้า
กิตติกรรมประกาศ	ก
บทคัดย่อภาษาไทย	ข
บทคัดย่อภาษาอังกฤษ	ค
สารบัญ	ง
สารบัญตาราง	ฉ
สารบัญภาพ	ช
บทที่ 1 บทนำ	
1.1 ความสำคัญและที่มาของปัญหาการวิจัย	1
1.2 วัตถุประสงค์ของโครงการวิจัย	6
1.3 ขอบเขตของการวิจัย	6
1.4 ประโยชน์ที่ได้รับ	6
บทที่ 2 งานวิจัยและสารสนเทศที่เกี่ยวข้อง	
2.1 งานวิจัยที่เกี่ยวข้อง	7
2.2 วิธีการโปรแกรมแบบมีเงื่อนไข	9
2.2.1 แนวคิดและหลักการโปรแกรมแบบมีเงื่อนไข	10
2.2.2 ตัวอย่างการโปรแกรมแบบมีเงื่อนไข	12
บทที่ 3 การออกแบบและพัฒนาโปรแกรม Top-K-patterns	
3.1 กรอบของงานวิจัย	16
3.2 การออกแบบและพัฒนาโปรแกรม	17
3.3 ตัวอย่างการใช้งานโปรแกรม	18
บทที่ 4 ผลการทดสอบโปรแกรม Top-K-patterns	
4.1 ข้อมูลที่ใช้ในการทดสอบ	22
4.2 ผลการทดสอบ	24
บทที่ 5 บทสรุป	
5.1 สรุปผลการวิจัย	32
5.2 ข้อจำกัดของโปรแกรมและข้อเสนอแนะ	34
บรรณานุกรม	35

หน้า

ภาคผนวก ก บทความวิจัยตีพิมพ์ในวารสารวิชาการ 41

1. N. Kerdprasop and K. Kerdprasop (2011). Frequent pattern discovery with constraint logic programming. *International Journal of Mathematical Models and Methods in Applied Sciences*, Vol.5, Issue 8, pp.1345-1353. (indexed in Scopus) 42

2. N. Kerdprasop, P. Kongchai, and K. Kerdprasop (2013). Constraint mining in business intelligence: A case study of customer churn prediction. *International Journal of Multimedia and Ubiquitous Engineering*, Vol.8, No.3, May, pp.11-20. (indexed in Scopus) 51

3. N. Kerdprasop, F. Koongaew, and K. Kerdprasop (2013). Building and querying a decision tree model with constraint logic programming. *International Journal of Software Engineering and Its Applications*, Vol.7, No.5, September, pp.269-282. (indexed in Scopus) 61

4. ไพชยนต์ คงไชย นิตยา เกิดประสพ และ กิตติศักดิ์ เกิดประสพ (2015). การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (The discovery of association rules using constraint logic programming). *วิศวกรรมสารเกษมบัณฑิต*, ปีที่ 4, ฉบับที่ 1, มกราคม-มิถุนายน 2558, หน้า 1-15. (indexed in TCI) 75

ภาคผนวก ข รหัสต้นฉบับของโปรแกรม Top-K-patterns 90

ประวัติผู้วิจัย 94

สารบัญตาราง

หน้า

ตารางที่ 2.1 ไลบรารีของระบบประมวลผลเงื่อนไขในซอฟต์แวร์ ECLiPSe 12
ตารางที่ 2.2 ผลประโยชน์ที่จะได้รับจากการจัดสรรคนรับผิดชอบแต่ละโครงการ 15
ตารางที่ 4.1 รายละเอียดข้อมูลลูกค้าบริษัทผู้ให้บริการโทรศัพท์ 23

สารบัญภาพ

หน้า

รูปที่ 1.1 ข้อมูลตัวอย่างรายการสินค้าที่ลูกค้าซื้อจากร้านค้า 2
รูปที่ 1.2 โครงข่ายของไอเท็มเซตทั้งหมดที่ต้องพิจารณาเพื่อค้นหาไอเท็มเซตที่ปรากฏบ่อย 3
รูปที่ 1.3 โครงข่ายของไอเท็มเซตที่ขนาดเล็กลงจากการไม่ต้องพิจารณาไอเท็ม E 4
รูปที่ 2.1 ซอฟต์แวร์สำหรับการพัฒนาโปรแกรมแบบมีเงื่อนไข 10
รูปที่ 2.2 ตัวอย่างการใช้เงื่อนไขบังคับในรูปแบบของ ECLiPSe 11
รูปที่ 2.3 โครงสร้างของโปรแกรมแบบมีเงื่อนไขในลักษณะของการโปรแกรมเชิงตรรกะ 11
รูปที่ 2.4 ตัวอย่างการโปรแกรมแบบมีเงื่อนไข 13
รูปที่ 2.5 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้โจทย์สมการหลายตัวแปร 13
รูปที่ 2.6 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้โจทย์ SEND + MORE = MONEY 14
รูปที่ 2.7 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้ปัญหาการจัดตารางทำงาน 15
รูปที่ 3.1 โครงสร้างของส่วนประกอบในโปรแกรม Top-K-patterns 16
รูปที่ 3.2 ขั้นตอนวิธีของโปรแกรม Top-K-patterns 17
รูปที่ 3.3 รูปแบบข้อมูลเข้าของโปรแกรม Top-K-patterns 18
รูปที่ 3.4 จอภาพแสดงการคอมไพล์เพื่อเริ่มต้นใช้งานโปรแกรม Top-K-patterns 19
รูปที่ 3.5 การสอบถามข้อมูลเพื่อแสดงผลลัพธ์เป็นกฎความสัมพันธ์ 3 อันดับแรก 19
รูปที่ 3.6 การสอบถามข้อมูลเพื่อแสดงกฎความสัมพันธ์ 5 อันดับแรกที่มีเบียร์เป็นเป้าหมาย ของกฎ 20
รูปที่ 3.7 การสอบถามเพื่อแสดงกฎความสัมพันธ์ 5 อันดับแรกโดยไม่ระบุเป้าหมายของกฎ 20
รูปที่ 4.1 โครงสร้างข้อมูลส่วนระบุรายละเอียดของไอเท็ม 23
รูปที่ 4.2 ตัวอย่างข้อมูลสามเรคคอร์ด 24
รูปที่ 4.3 ผลการทดสอบความถูกต้องของโปรแกรม Top-K-patterns เปรียบเทียบกับโปรแกรม Apriori 25
รูปที่ 4.4 การทดสอบข้อคำถามที่ 1 ด้วยโปรแกรม Apriori และโปรแกรม Top-K-patterns 26
รูปที่ 4.5 การทดสอบข้อคำถามที่ 2 ด้วยโปรแกรม Top-K-patterns 27
รูปที่ 4.6 การทดสอบข้อคำถามที่ 3 ด้วยโปรแกรม Top-K-patterns 28
รูปที่ 4.7 การทดสอบข้อคำถามที่ 4 ด้วยโปรแกรม Top-K-patterns 28
รูปที่ 4.8 การทดสอบข้อคำถามที่ 5 ด้วยโปรแกรม Top-K-patterns 29
รูปที่ 4.9 การทดสอบข้อคำถามที่ 6 ด้วยโปรแกรม Top-K-patterns 29
รูปที่ 4.10 การทดสอบข้อคำถามที่ 7 ด้วยโปรแกรม Top-K-patterns 30
รูปที่ 4.11 การทดสอบข้อคำถามที่ 8 ด้วยโปรแกรม Top-K-patterns 30
รูปที่ 4.12 การเปรียบเทียบเวลาที่ใช้ในการประมวลผลและจำนวนกฎที่ได้เมื่อเพิ่มเงื่อนไขบังคับ 31

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

การค้นหารูปแบบที่ปรากฏบ่อย (frequent pattern mining) เป็นการค้นหารูปแบบที่เกิดขึ้นซ้ำ ๆ ในข้อมูลขนาดใหญ่ เช่น ฐานข้อมูลทรานแซคชันของห้างสรรพสินค้า ล็อกไฟล์บันทึกการเข้าใช้เว็บของผู้ใช้อินเทอร์เน็ต หรือฐานข้อมูลโครงสร้างโมเลกุลในสารประกอบโปรตีน การตรวจพบรูปแบบที่ปรากฏบ่อยจะเป็นความรู้พื้นฐานที่สำคัญ นำไปสู่ความเข้าใจในการเชื่อมโยงด้านต่าง ๆ ของข้อมูล เช่น พฤติกรรมผู้บริโภค รูปแบบการใช้งานเว็บเพจ แนวโน้มการเปลี่ยนแปลงทางโครงสร้าง สายรหัสพันธุกรรม เป็นต้น การค้นหารูปแบบที่ปรากฏบ่อยนี้เป็นขั้นตอนพื้นฐานที่สำคัญของการทำเหมืองข้อมูลประเภท การค้นหาความสัมพันธ์ (association mining)

การทำเหมืองข้อมูลจากฐานข้อมูลขนาดใหญ่เพื่อค้นหาความสัมพันธ์หรือความเกี่ยวข้องของสินค้าที่ถูกซื้อร่วมกันบ่อย ได้รับการเสนอแนวคิดครั้งแรกโดยทีมนักวิจัยของศูนย์วิจัยบริษัทไอบีเอ็มที่อัลมาเด็น (IBM Almaden Research Center) ที่ประกอบด้วย Rakesh Agrawal, Tomasz Imielinski และ Arun Swami (1993) โดยนักวิจัยกลุ่มนี้ได้เสนออัลกอริทึมที่ต่อมาภายหลังนิยมเรียกว่า อัลกอริทึม AIS (Agrawal-Imielinski-Swami) เพื่อค้นหาความสัมพันธ์และความเชื่อมโยงของสินค้าที่มักจะถูกซื้อในคราวเดียวกัน แล้วแสดงความสัมพันธ์นั้นในลักษณะของกฎ ซึ่งก็คือข้อความเชิงตรรกะที่แสดงการเกิดเหตุการณ์ในรูปแบบ "ถ้า...แล้ว" หรือ IF...THEN เรียกข้อความหรือกฎประเภทนี้ว่า *กฎความสัมพันธ์* (association rule)

กฎความสัมพันธ์มักจะถูกเขียนอยู่ในรูปแบบที่ใช้ตัวแปร X และ Y แทนเซตหรือกลุ่มของสินค้า (เรียกว่าไอเท็มเซต โดยแต่ละไอเท็มจะหมายถึงสินค้าแต่ละชิ้น) และใช้สัญลักษณ์ลูกศรแทนความเชื่อมโยงระหว่างเซตของสินค้า เช่น $X \rightarrow Y$ (s, c) แทนความหมายว่าเมื่อลูกค้าซื้อสินค้า X แล้วลูกค้าจะซื้อสินค้า Y ร่วมด้วย โดย X และ Y คือเซตของสินค้าที่ไม่ใช่เซตว่างและเป็นเซตที่สมาชิกไม่ซ้ำกัน สัญลักษณ์ s แทนค่า support หรือค่าสนับสนุน ใช้เป็นมาตรวัดความถี่ของการซื้อสินค้า X และ Y ซึ่งคำนวณได้จากสัดส่วนของจำนวนเรคคอร์ดที่ปรากฏทั้งสินค้าในเซต X และ Y ต่อจำนวนเรคคอร์ดทั้งหมดในฐานข้อมูลทรานแซคชัน ส่วนสัญลักษณ์ c แทนค่า confidence หรือค่าความเชื่อมั่นของกฎ ค่านี้จะใช้แทนความถูกต้องหรือความน่าเชื่อถือของกฎโดยคำนวณได้จากสัดส่วนของจำนวนเรคคอร์ดที่ปรากฏทั้ง X และ Y ต่อจำนวนเรคคอร์ดที่ปรากฏ X (เป็นการให้ความสนใจเฉพาะเซต X โดยไม่สนใจว่า Y จะปรากฏร่วมด้วยหรือไม่)

เมื่อมีการกำหนด support และ confidence เป็นเกณฑ์หรือมาตรวัดความถี่และความน่าเชื่อถือของกฎ วิธีการค้นหากฎความสัมพันธ์จึงประกอบด้วยสองขั้นตอนหลัก คือ

1. ค้นหาไอเท็มเซตที่ปรากฏบ่อยทั้งหมด โดยไอเท็มเซตที่จัดเป็นเซตที่ปรากฏบ่อยจะต้องมีค่าสนับสนุนไม่ต่ำกว่าค่าสนับสนุนที่กำหนดไว้เป็นเกณฑ์ขั้นต่ำ (เรียกเกณฑ์ขั้นต่ำนี้ว่า minimum support หรือ min_sup)
2. สร้างกฎความสัมพันธ์จากไอเท็มเซตที่ปรากฏบ่อย โดยกฎนี้จะต้องมีค่าความเชื่อมั่นไม่ต่ำกว่าค่าความเชื่อมั่นที่กำหนดไว้เป็นเกณฑ์ขั้นต่ำ (เรียกว่า minimum confidence หรือ min_conf)

ตัวอย่างเช่น ถ้าฐานข้อมูลทรานแซคชันประกอบด้วย ข้อมูลการซื้อสินค้าของลูกค้า 5 ราย (รายละเอียดดังรูปที่ 1.1) การซื้อสินค้าของลูกค้าแต่ละรายจะบันทึกเป็นแต่ละทรานแซคชันหรือเรคคอร์ด ที่ระบุด้วยหมายเลขทรานแซคชัน หรือ TID (transaction identifier)

TID	Items
1	{Cereal, Milk}
2	{Beer, Cereal, Diaper, Egg}
3	{Beer, Diaper, Milk}
4	{Beer, Cereal, Diaper, Milk}
5	{Diaper, Milk}

รูปที่ 1.1 ข้อมูลตัวอย่างรายการสินค้าที่ลูกค้าซื้อจากร้านค้า

ในฐานข้อมูลตัวอย่างมีสินค้า 5 ชนิดได้แก่ Beer, Cereal, Diaper, Egg, Milk สินค้าเหล่านี้เรียกว่าไอเท็ม กำหนดให้ในการค้นหาสินค้าที่ถูกซื้อร่วมกันบ่อย มีค่า min_sup เป็น 3/5 หรือ 60% และค่า min_conf เป็น 90%

ขั้นตอนแรกของกระบวนการค้นหากฎความสัมพันธ์ จะเริ่มต้นด้วยการค้นหาไอเท็มเซตที่ปรากฏบ่อยทั้งหมด โดยจะต้องค้นหาจากไอเท็มเซตที่ไม่ใช่เซตว่างทั้งหมด 31 เซต (คำนวณได้จาก 2^5-1) แสดงไอเท็มเซตทั้งหมดได้ดังรูปที่ 1.2 (แต่ละไอเท็มในรูปใช้อักษรย่อ B, C, D, E, M แทน Beer, Cereal, Diaper, Egg, Milk ตามลำดับและละเว้นการใช้สัญลักษณ์เซตหรือเครื่องหมายวงเล็บปีกกาเพื่อให้อ่านง่าย)

เมื่อนับค่า support ของแต่ละไอเท็มเซตทั้ง 31 เซตเพื่อคัดเลือกเฉพาะไอเท็มเซตที่ผ่านเกณฑ์ min_sup 60% จะประกอบด้วยไอเท็มเซต 6 เซต ต่อไปนี้

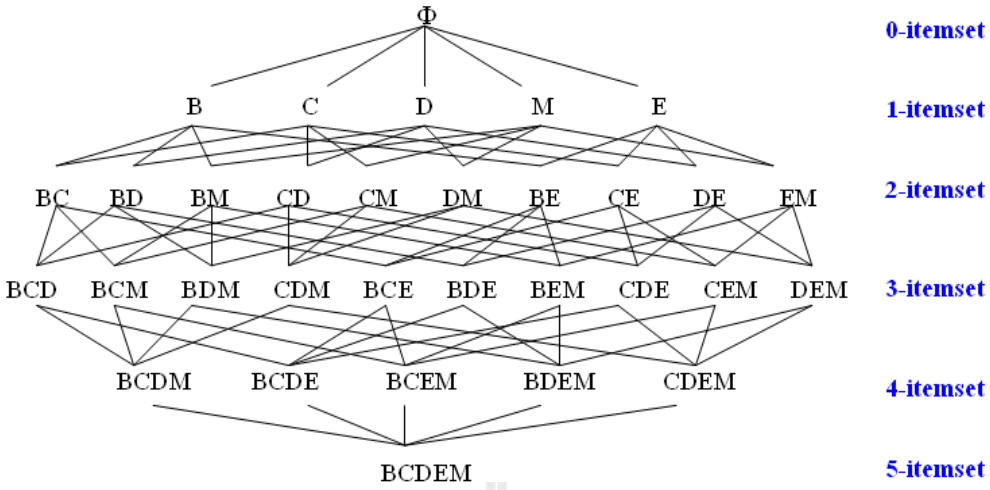
1-itemset: {Beer}, {Cereal}, {Diaper}, {Milk}

2-itemset: {Beer and Diaper}, {Diaper and Milk}

3-itemset: {}

4-itemset: {}

5-itemset: {}



รูปที่ 1.2 โครงข่ายของไอเท็มเซตทั้งหมดที่ต้องพิจารณาเพื่อค้นหาไอเท็มเซตที่ปรากฏบ่อย

ขั้นตอนที่สองของการสร้างกฎความสัมพันธ์ จะเป็นการนำข้อมูลในไอเท็มเซตที่ปรากฏบ่อยมาสร้างเป็นกฎ โดยเริ่มจาก 2-itemset, 3-itemset, ... ไปตามลำดับ แต่เนื่องจากในตัวอย่างนี้ไอเท็มเซตที่ปรากฏบ่อยตั้งแต่ 3-itemset เป็นต้นไปเป็นเซตว่าง จึงพิจารณาเฉพาะข้อมูลใน 2-itemset ที่ประกอบด้วยสองเซตคือ {Beer and Diaper} และ {Diaper and Milk} นำไอเท็มใน 2-itemset มาสร้างกฎความสัมพันธ์ในเบื้องต้นได้ 4 กฎ ดังนี้

Beer ⟶ Diaper (confidence = 3/3 = 100%)

Diaper ⟶ Beer (confidence = 3/4 = 75%)

Diaper ⟶ Milk (confidence = 3/4 = 75%)

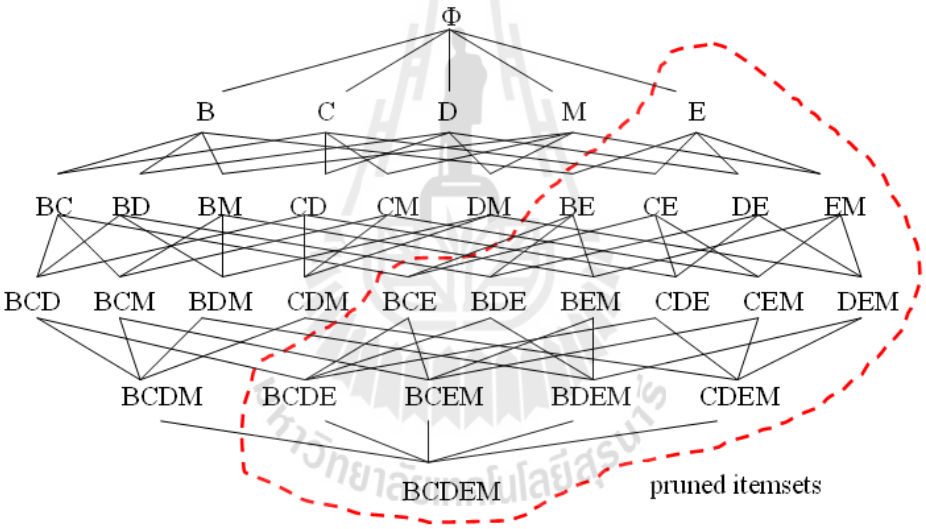
Milk ⟶ Diaper (confidence = 3/4 = 75%)

แต่จากเกณฑ์ min_conf ที่กำหนดไว้ 90% ทำให้กฎที่ 2-4 ไม่ผ่านเกณฑ์ ในผลลัพธ์สุดท้ายได้กฎความสัมพันธ์เพียงข้อเดียวคือ Beer ⟶ Diaper (confidence = 3/3 = 100%) จากค่าสนับสนุนหรือ support ของ Beer and Diaper = 3/5 ทำให้แปลความหมายของกฎความสัมพันธ์นี้ได้ว่า *ลูกค้าของร้านค้านี้จำนวนมากถึง 60% ที่ตั้งใจมาซื้อ Beer แล้วจะซื้อ Diaper ด้วย ความสัมพันธ์นี้ระบุด้วยความเชื่อมั่น 100%*

จากจุดเริ่มต้นของการเสนออัลกอริทึม AIS (Agrawal et al., 1993) ในปีค.ศ. 1993 ทำให้แนวคิดของการค้นหาความรู้จากฐานข้อมูล ในลักษณะของการทำเหมืองข้อมูลประเภทการค้นหาความสัมพันธ์ได้รับความสนใจอย่างมากจากนักวิจัยในสาขาการทำเหมืองข้อมูลและการวิเคราะห์ข้อมูล

ในปีต่อมา Rakesh Agrawal และ Ramakrishnan Srikant (1994) ได้ปรับปรุงอัลกอริทึม AIS ให้ทำงานได้เร็วขึ้นโดยอาศัยคุณสมบัติที่เรียกว่า Apriori property ที่ระบุว่า “เซตย่อยของไอเท็มเซตที่ปรากฏบ่อย จะต้องเป็นเซตที่ปรากฏบ่อยด้วยเช่นกัน” และเรียกอัลกอริทึมที่พัฒนาขึ้นใหม่นี้ว่า อัลกอริทึม APRIORI

จุดเด่นของอัลกอริทึมนี้อยู่ที่ความสามารถในการพัฒนาความเร็วในการค้นหาไอเท็มเซตที่ปรากฏบ่อย ด้วยการละเว้นการพิจารณาไอเท็มเซตที่ปรากฏซ้ำด้วยความถี่ต่ำกว่าเกณฑ์ min_sup ดังตัวอย่างข้อมูลทรานแซคชันในรูปที่ 1.1 ถ้ากำหนด min_sup = 2/5 หรือ 40% จะพบว่าไอเท็ม Egg (หรือ E) ปรากฏในทรานแซคชันที่ 2 เพียงทรานแซคชันเดียว จึงไม่จัดเป็นไอเท็มเซตที่ปรากฏบ่อย ด้วยคุณสมบัติ Apriori ทำให้เราสามารถตัดการพิจารณาไอเท็มเซตทุกเซตที่มี E เป็นสมาชิก จึงลดจำนวนไอเท็มเซตที่ต้องพิจารณาจาก 31 เซต (ดังรูปที่ 1.2) ลงเหลือเพียง 15 เซต ดังแสดงในรูปที่ 1.3 (ส่วนที่ล้อมรอบด้วยเส้นประแสดงขอบเขตของไอเท็มเซตที่ถูกตัดทิ้ง เนื่องจากปรากฏบ่อยไม่ถึงเกณฑ์ min_sup จึงไม่ต้องพิจารณา)



รูปที่ 1.3 โครงข่ายของไอเท็มเซตที่ขนาดเล็กลงจากการไม่ต้องพิจารณาไอเท็ม E

จากตัวอย่างข้างต้นจะเห็นได้ว่าในขั้นตอนการค้นหารูปแบบที่ปรากฏบ่อย ต้องใช้เกณฑ์ minimum support ที่ระบุโดยผู้ใช้เพื่อจำกัดขอบเขตของการค้นหา (search space) ซึ่งในทางปฏิบัติกับข้อมูลจริงที่มีจำนวนทรานแซคชันมาก การะบุค่า minimum support ที่ต่ำเกินไปจะทำให้ได้คำตอบเป็นไอเท็มเซตปริมาณมหาศาล ซึ่งยากต่อการอ่านและแปลผล นอกจากนี้ในกรณีที่เครื่องคอมพิวเตอร์ที่ใช้ประมวลผลมีความจุของหน่วยความจำน้อย ปริมาณไอเท็มเซตที่มากเกินไปสามารถทำให้เกิดเหตุการณ์หน่วยความจำไม่เพียงพอต่อการใช้งานได้ แต่ในทางตรงกันข้ามถ้าผู้ใช้ระบุค่า minimum support สูงเกินไป อาจได้คำตอบเป็นเซตว่าง นั่นคือไม่มีไอเท็มเซตใดเลยที่ไอเท็มปรากฏ

ซ้ำด้วยความถี่ถึงเกณฑ์ขั้นต่ำที่กำหนด ทำให้ผู้ใช้ต้องปรับลดค่า minimum support และทดลองซ้ำในลักษณะลองผิดลองถูก

การปรับเปลี่ยนค่า minimum support และทดลองค้นหาไอเท็มเซตที่ปรากฏบ่อยซ้ำหลายครั้งเช่นนี้เป็นภาระหนักสำหรับผู้ใช้ จึงมีการเสนอแนวคิดของการยกเลิกเกณฑ์ minimum support แล้วเปลี่ยนไปใช้เกณฑ์ top-k frequent itemsets (Wang et al., 2005; Chuang et al., 2008) ซึ่งใช้งานได้ง่ายกว่า โดยให้ผู้ใช้ระบุจำนวนรูปแบบที่ปรากฏบ่อยที่สุด k อันดับแรก เช่น จากข้อมูลทรานแซคชันในรูปที่ 1 ถ้าผู้ใช้ระบุว่าต้องการทราบไอเท็มเซตที่ปรากฏซ้ำบ่อยที่สุด 5 อันดับแรก (top-5 frequent itemsets) จะได้คำตอบเป็นดังนี้ (คำตอบที่ได้ปรากฏไอเท็มเซตหกอันดับเนื่องจากในอันดับที่สามถึงหกมีค่า support เท่ากัน)

{Diaper}	support = 4
{Milk}	support = 4
{Beer}	support = 3
{Cereal}	support = 3
{Beer and Diaper}	support = 3
{Diaper and Milk}	support = 3

นอกจากการค้นหารูปแบบที่ปรากฏบ่อยด้วยวิธีการระบุค่า top-k แทนค่า minimum support แล้ว การทำให้การทำเหมืองข้อมูลชนิดการค้นหาความสัมพันธ์สามารถใช้งานได้จริงกับข้อมูลขนาดใหญ่ หรือข้อมูลที่มีปริมาณไม่จำกัด เช่นข้อมูลสตรีม ควรจะมีส่วนอำนวยความสะดวกให้ผู้ใช้ระบุเงื่อนไขในความสัมพันธ์ที่ผู้ใช้สนใจ เช่นไอเท็มเซตที่ปรากฏบ่อย 5 อันดับแรกจะต้องมีไอเท็ม Milk อยู่ในเซตนั้นด้วย หรืออาจจะระบุเงื่อนไขว่าไอเท็มเซตที่ปรากฏบ่อย 10 อันดับแรกจะต้องมีราคาขายรวมมากกว่าห้าร้อยบาท การกำหนดเงื่อนไขโดยผู้ใช้เช่นนี้จะทำให้ได้ผลลัพธ์เป็นรูปแบบที่ปรากฏบ่อยที่ตรงกับความสนใจของผู้ใช้ และยังช่วยให้กระบวนการค้นหารูปแบบทำได้รวดเร็วขึ้นเนื่องจากสามารถลดขนาดของ search space ลงได้มาก

โครงการวิจัยนี้จึงมีแนวคิดที่จะออกแบบอัลกอริทึมและพัฒนาโปรแกรมต้นแบบเพื่อการค้นหารูปแบบที่ปรากฏบ่อย k อันดับแรก โดยผู้ใช้สามารถระบุเงื่อนไขของการค้นหาได้ ทั้งนี้เพื่อให้ได้รูปแบบที่ปรากฏบ่อยที่มีลักษณะใกล้เคียงกับความต้องการของผู้ใช้มากที่สุด

1.2 วัตถุประสงค์ของโครงการวิจัย

- ออกแบบกรอบงาน (framework) และอัลกอริทึมสำหรับการค้นหารูปแบบที่ปรากฏบ่อย k อันดับแรกในฐานข้อมูล ด้วยวิธีการกำหนดเงื่อนไข (constraints) ในข้อคำถามของผู้ใช้
- พัฒนาโปรแกรมต้นแบบด้วยวิธีการโปรแกรมเชิงตรรกะแบบมีเงื่อนไขบังคับ

1.3 ขอบเขตของการวิจัย

โครงการวิจัยนี้เป็นการออกแบบและพัฒนาโปรแกรมการทำเหมืองข้อมูล ประเภทการค้นหาความสัมพันธ์ โดยเน้นที่การประมวลผลข้อคำถามเพื่อค้นหารูปแบบที่ปรากฏบ่อย K อันดับแรก ผลลัพธ์ของการประมวลผลข้อคำถาม จะเป็นรูปแบบความสัมพันธ์ที่ตรงกับความต้องการของผู้ใช้มากที่สุด โดยยังไม่ได้คำนึงถึงการปรับปรุงประสิทธิภาพการ access ข้อมูล และการทำงานกับฐานข้อมูลจะยังไม่คำนึงถึงสถานการณ์ของการ update ข้อมูล

การพัฒนาโปรแกรมจะใช้ซอฟต์แวร์ ECLiPSe (http://eclipseclp.org/) และใช้ภาษาเชิงตรรกะในการเขียนรหัสคำสั่ง

1.4 ประโยชน์ที่ได้รับ

งานวิจัยนี้เป็นการพัฒนาองค์ความรู้ใหม่ในด้านการทำเหมืองข้อมูลประเภทการค้นหารูปแบบที่ปรากฏบ่อยและการสร้างกฎความสัมพันธ์ ประโยชน์ที่ได้รับจากงานวิจัยนี้คือ

- สามารถตีพิมพ์ผลงานวิจัยเกี่ยวกับเทคนิคและอัลกอริทึมใหม่ที่ออกแบบขึ้นในส่วนของการค้นหารูปแบบที่ปรากฏบ่อย ในวารสารวิชาการทั้งระดับชาติและระดับนานาชาติได้ 4 บทความ (ปรากฏในภาคผนวก)
- ผู้ช่วยวิจัยที่เป็นนักศึกษาในระดับบัณฑิตศึกษา ได้พัฒนาความสามารถในการทำงานวิจัย และสามารถปรับใช้แนวคิดจากโครงการวิจัยนี้เป็นวิทยานิพนธ์ได้ 3 เรื่อง
- การออกแบบและพัฒนาระบบขึ้นในลักษณะ prototype ทำให้ได้โปรแกรมต้นแบบที่สามารถจดลิขสิทธิ์ได้ 1 โปรแกรม

บทที่ 2
งานวิจัยและสารสนเทศที่เกี่ยวข้อง

2.1 งานวิจัยที่เกี่ยวข้อง

นับจากความสำเร็จของการคิดค้นอัลกอริทึม APRIORI (Agrawal and Srikant, 1994) ได้มีนักวิจัยจำนวนมากใช้แนวคิดของอัลกอริทึมนี้เป็นพื้นฐาน และปรับปรุงประสิทธิภาพให้ดีขึ้นด้วยวิธีต่าง ๆ กัน งานวิจัยเด่นในกลุ่มนี้ประกอบด้วยงานวิจัยของ J.S. Park, M.S. Chen และ P.S. Yu (1995) ที่เสนอแนวทางของการใช้เทคนิคแฮชชิงเพื่อเพิ่มความเร็วของการค้นหาไอเท็มเซตที่ปรากฏบ่อย J. Han และ Y. Fu (1995) ได้เสนอให้สร้างกฎความสัมพันธ์ที่มีโครงสร้างของไอเท็มเซตเป็นหลายระดับชั้นเพื่อให้มีความละเอียดมากขึ้น เช่น จากการค้นพบกฎความสัมพันธ์ Beer $\rightarrow$ Diaper สามารถสืบค้นให้ละเอียดขึ้นได้ว่าเบียร์ที่ลูกค้านิยมซื้อร่วมกับผ้าอ้อมเด็กนั้นเป็นเบียร์ประเภทใด

ในกรณีของการค้นหาความสัมพันธ์จากฐานข้อมูลทรานแซคชันที่มีทั้งจำนวนข้อมูลและขนาดของเรคคอร์ดใหญ่มากจนกระทั่งไม่สามารถบรรจุข้อมูลทั้งหมดในหน่วยความจำหลักได้ A. Savasere, E. Omiecinski และ S. Navathe (1995) เสนอให้ใช้เทคนิคการแบ่งข้อมูลเป็นส่วนย่อยแล้วทยอยค้นหาความสัมพันธ์ที่ปรากฏในแต่ละส่วนย่อยนั้น H. Toivonen (1996) ได้ใช้แนวทางที่ต่างออกไปโดยเสนอการใช้เทคนิคการสุ่มเพื่อค้นหาความสัมพันธ์จากข้อมูลตัวแทน D.W. Cheung และคณะ (Cheung et al., 1996) ได้เสนอให้ใช้เทคนิค incremental หรือการทำงานกับข้อมูลครั้งละไม่มาก และเมื่ออ่านข้อมูลใหม่เพิ่มเติมจะต้องสามารถปรับกฎความสัมพันธ์ให้ถูกต้องสอดคล้องกับข้อมูลใหม่ได้ นอกจากนี้ยังมีนักวิจัยจำนวนมาก (Park et al.,1995; Agrawal & Shafer, 1996; Cheung et al., 1996; Zaki et al., 1997) ได้เสนอแนวทางการพัฒนาความเร็วในการค้นหาไอเท็มเซตที่ปรากฏบ่อยด้วยการประมวลผลแบบขนาน

งานวิจัยที่กล่าวถึงข้างต้นล้วนแต่ใช้แนวทาง APRIORI เป็นพื้นฐานแต่เสริมประสิทธิภาพความเร็วด้วยเทคนิคต่าง ๆ กัน แต่งานวิจัยของ Jiawei Han, Jian Pei และ Yiwen Yin (2000) ได้เสนอแนวทางที่แตกต่างออกไป ด้วยการอ่านข้อมูลทรานแซคชันแล้วสร้างโครงสร้างต้นไม้เรียกว่า frequent pattern tree หรือ FP-tree เพื่อบันทึกไอเท็มเซตที่ปรากฏในทรานแซคชัน จากนั้นใช้อัลกอริทึม FP-growth เพื่อค้นหาไอเท็มเซตที่ปรากฏบ่อยโดยไม่ต้องสร้างไอเท็มเซตชั่วคราวแล้วตัดทิ้งภายหลังเมื่อค่าสนับสนุนของเซตต่ำกว่าเกณฑ์ค่าสนับสนุนขั้นต่ำ วิธีนี้เมื่อเปรียบเทียบกับวิธี ARIORI แล้วสามารถเพิ่มความเร็วในการค้นหาไอเท็มเซตที่ปรากฏบ่อยและใช้หน่วยความจำน้อยกว่า แนวทางการค้นหาไอท็มเซตที่ปรากฏบ่อยด้วยโครงสร้าง FP-tree จึงได้รับความสนใจจากนักวิจัยและ

พัฒนาต่อเนื่องมาโดยลำดับตั้งแต่ช่วงปีค.ศ. 2000 ถึงปัจจุบัน (Agrawal et al., 2001; Pei et al., 2001; Liu et al., 2002; Grahne & Zhu, 2003)

การใช้ค่าสนับสนุนขั้นต่ำเพื่อลดขนาดของ search space และเพิ่มความเร็วในการค้นหาไอเท็มเซตที่ปรากฏบ่อย จัดเป็นเงื่อนไข หรือ constraint อย่างง่ายชนิดหนึ่ง เริ่มใช้โดย R. Agrawal และ R. Srikant (1994) ต่อมาได้มีการขยายการศึกษาเกี่ยวกับการใช้เงื่อนไขให้สามารถระบุไอเท็มที่สนใจได้ (Srikant et al., 1997) J. Pei และคณะ (Pei and Han, 2000; Pei et al., 2004) ได้ศึกษาการกำหนดเงื่อนไขในรูปแบบเดียวกับที่ใช้ในภาษา SQL เช่นกำหนดช่วงของค่าในไอเท็มเซต I เป็นค่าที่ต่ำกว่า 15 (range(I) < 15) หรือกำหนดค่าเฉลี่ยในไอเท็มเซตให้มีค่ามากกว่า 50 (avg(I) > 50) เป็นต้น S. Bistarelli และ F. Bonchi (2007) ได้นำเสนอแนวคิดของการใช้เงื่อนไขที่ยืดหยุ่นด้วยการคำนวณความน่าจะเป็น แล้วใช้ค่าความน่าจะเป็นเทียบเคียงว่าไอเท็มเซตที่คำนวณได้สามารถใช้เป็นคำตอบที่ตรงกับเงื่อนไขที่ผู้ใช้ระบุหรือไม่

ในช่วงระยะเวลาทศวรรษแรกตั้งแต่ปีค.ศ. 1993 ถึง 2000 ของการวิจัยด้านการค้นหาไอเท็มเซตปรากฏบ่อยและกฎความสัมพันธ์ เทคนิคการค้นพบกฎความสัมพันธ์และการค้นหารูปแบบที่ปรากฏบ่อยได้รับการพัฒนาอย่างต่อเนื่องให้มีประสิทธิภาพสูง และสามารถรองรับข้อมูลขนาดใหญ่ได้ แต่เมื่อเทคโนโลยีอินเทอร์เน็ตได้รับความนิยมสูงขึ้น ลักษณะของข้อมูลเปลี่ยนจาก offline เป็น online และปริมาณของข้อมูลเพิ่มขึ้นอย่างไม่มีขีดจำกัดเกิดเป็นลักษณะข้อมูลสตรีม (data stream) ข้อมูลสตรีมเริ่มได้รับการนิยามเมื่อปีค.ศ. 2001 (Guha et al., 2001; Babcock et al., 2002; Gaber et al., 2005; Jiang & Gruenwald, 2006) ว่าหมายถึง

- ข้อมูลที่เกิดขึ้นอย่างต่อเนื่อง
- ไม่มีขีดจำกัดในเรื่องของปริมาณและจุดสิ้นสุด
- ข้อมูลถูกส่งออกจากแหล่งผลิตด้วยความเร็วสูงและอาจจะมีการกระจายของข้อมูลที่ไม่คงที่

นักวิจัยได้พยายามปรับปรุงเทคนิคการค้นพบกฎความสัมพันธ์ให้ทำงานได้กับข้อมูลสตรีม เช่น M. Halatchev และ L. Gruenwald (2005) ได้ปรับปรุงเทคนิคการค้นพบกฎความสัมพันธ์ให้สามารถประเมินข้อมูลที่หายไปในข้อมูลสตรีมที่รับมาจากเซนเซอร์ที่ส่งผ่านเครือข่าย H. Kargupta และคณะ (2004) ได้พัฒนาระบบ VEDAS ให้สามารถตรวจจับยานพาหนะในเวลาจริง นอกจากนี้ยังมีงานวิจัยจำนวนมากในช่วงระยะเวลาปี 2004 ถึงปัจจุบัน (Cai et al., 2004; Chang & Lee, 2004; Charikar et al., 2004; Chi ey al., 2004; Gaber et al., 2004; Ghoting & Parthasarathy, 2004; Li et al., 2004; Teng et al., 2004; Yu et al., 2004; Mao et al., 2005; Kerdprasop et al., 2006) ที่มุ่งพัฒนาเทคนิคในการค้นหารูปแบบความสัมพันธ์และไอเท็มเซตที่ปรากฏบ่อยในข้อมูลสตรีม อัลกอริทึมต่าง ๆ ที่ถูกเสนอเหล่านี้ บางอัลกอริทึมพัฒนาขึ้นเพื่อรองรับเฉพาะบางแอพพลิเคชัน บางอัลกอริทึมใช้ค้นหาความสัมพันธ์เฉพาะข้อมูลในบางช่วง เช่น เฉพาะข้อมูลที่เกิดขึ้นล่าสุดใน

สตรีม และบางอัลกอริทึมทำงานกับข้อมูลสตรีมในลักษณะ offline โครงการวิจัยนี้ต้องการพัฒนาเทคนิคการสอบถามและการค้นหารูปแบบที่ปรากฏบ่อย k อันดับแรกเพื่อให้ได้รูปแบบความสัมพันธ์ที่ปรากฏบ่อยในเวลาที่รวดเร็วและทันต่อความต้องการใช้งาน การค้นหาความสัมพันธ์และรูปแบบที่ปรากฏบ่อยจะมีลักษณะของ constraint-based mining โดยการค้นหารูปแบบจะใช้เงื่อนไขที่ระบุโดยผู้ใช้เป็นเกณฑ์ในการค้นหา

การใช้เงื่อนไขที่ระบุโดยผู้ใช้เป็นแนวทางวิจัยที่ R. Srikant และคณะ (1997) ได้เสนอแนวคิดโดยยังไม่มีการพัฒนาโปรแกรมขึ้นใช้จริง ต่อมา R. Bayardo และทีมงาน (2000) ได้พัฒนาเงื่อนไขพิเศษนอกจากการกำหนดค่าสนับสนุนต่ำสุดและค่าเชื่อมั่นต่ำสุด เรียกว่าเงื่อนไข minimp (minimum improvement) เป็นเงื่อนไขที่มีแนวคิดคล้ายการกำหนดค่าเชื่อมั่นต่ำสุดเพื่อลดความซับซ้อนของกฎความสัมพันธ์ที่ได้จากฐานข้อมูลขนาดใหญ่และให้ผลลัพธ์เป็นกฎความสัมพันธ์ที่เข้าใจง่ายแก่ผู้ใช้ B. Jeudy และ J. Boulicaut (2002) ได้เสนอการใช้เงื่อนไขบังคับในฐานข้อมูลอุปนัยเพื่อเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์และได้ใช้เครื่องมือ Mine Rule Operator เพื่อง่ายต่อการค้นหากฎความสัมพันธ์ ทีมของ A. Gallo และคณะ (2005) เสนอเกี่ยวกับการเพิ่มประสิทธิภาพการค้นหากฎความสัมพันธ์โดยใช้เงื่อนไขบังคับเช่นเดียวกัน ทีมวิจัยนี้ได้เสนออัลกอริทึมซึ่งช่วยในการสอบถามข้อมูล (query) ในฐานข้อมูลเพื่อให้ได้กฎความสัมพันธ์ตามที่ผู้ใช้ต้องการและช่วยลดเวลาในการค้นหา T. Trifonov และ T. Georgieva (2009) ได้นำเสนอการประยุกต์ใช้การทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับเพื่อหาความสัมพันธ์ของเสียงระฆัง โดยเขาได้พัฒนาโปรแกรมด้วยภาษาจาวาและใช้ภาษาสอบถามข้อมูล SQL เพื่อให้ใช้งานง่ายและสะดวกต่อผู้ใช้ที่ไม่จำเป็นต้องมีความรู้ภาษาสอบถามข้อมูล

โครงการวิจัยนี้ใช้แนวทางการเพิ่มประสิทธิภาพการค้นหากฎความสัมพันธ์ด้วยการใช้เงื่อนไขเช่นเดียวกับงานวิจัยอื่น แต่จะมีข้อแตกต่างจากงานอื่นตรงที่มีการใช้เงื่อนไขบังคับในลักษณะของรูปแบบที่เกิดด้วยความถี่ค่อนข้างสูงซึ่งเรียกว่า top-k patterns การพัฒนาโปรแกรมต้นแบบที่เรียกชื่อว่า Top-k frequent pattern mining จะใช้แนวทางการโปรแกรมเชิงตรรกะด้วยภาษา Prolog และพัฒนาบนสภาพแวดล้อมของซอฟต์แวร์ ECLiPSe ที่เป็น constraint logic programming system ทำให้สามารถใช้คุณสมบัติของ meta-programming ในการกำหนด rule ที่ตรงกับความสนใจของผู้ใช้และสามารถผนวกเงื่อนไขไว้ใน rule ดังนั้นกรอบแนวคิดของการออกแบบระบบและการพัฒนาระบบประมวลผลข้อคำถามจะอ้างอิงกับระบบฐานข้อมูลนิรนัยและการโปรแกรมเชิงตรรกะ

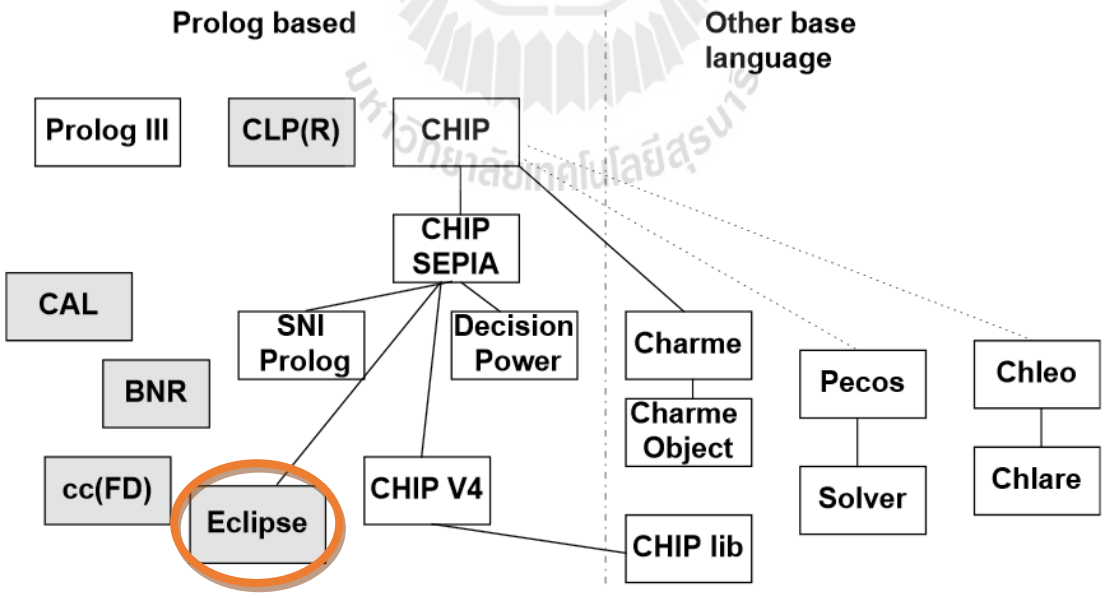
2.2 วิธีการโปรแกรมแบบมีเงื่อนไข

การโปรแกรมแบบมีเงื่อนไข (constraint programming) เป็นแนวทางการทำโปรแกรมที่มีวัตถุประสงค์หลักเพื่อเพิ่มความเร็วในการประมวลผลโปรแกรม ด้วยการลดขอบเขตค่าที่เป็นไปได้

ของตัวแปรต่าง ๆ ที่จะเป็นผลลัพธ์สุดท้ายของโปรแกรม การลดขอบเขตค่านี้จะกระทำผ่านการระบุเงื่อนไข (constraint) วิธีการโปรแกรมแบบนี้นิยมใช้ในงานด้านปัญญาประดิษฐ์และงานวางแผนที่มีลักษณะการค้นหาผลเฉลยที่ตรงตามเงื่อนไข รวมถึงงานในลักษณะการทำให้พอใจตามเงื่อนไขบังคับและการหาค่าเหมาะที่สุด (constraint satisfaction and optimization)

2.2.1 แนวคิดและหลักการโปรแกรมแบบมีเงื่อนไข

การโปรแกรมแบบมีเงื่อนไขเป็นแนวทางการพัฒนาโปรแกรมที่มีวัตถุประสงค์ให้การประมวลผลรวดเร็วขึ้นและโปรแกรมเมอร์ใช้เวลาในการพัฒนาโปรแกรมสั้นลง แนวคิดนี้จึงใช้วิธีการสร้างระบบประมวลผลเงื่อนไข (constraint system) ที่มีการเพิ่มไลบรารีที่จำเป็นในระบบประมวลผลเพื่อทำหน้าที่คำนวณขอบเขตค่าของตัวแปรต่าง ๆ ในผู้ใช้ระบุในโปรแกรม ฟังก์ชันการทำงานในส่วนนี้จะเรียกว่าส่วนแก้ปัญหาเงื่อนไข (constraint solver) การพัฒนาส่วนแก้ปัญหาเงื่อนไขสามารถใช้ภาษาระดับสูงได้หลากหลายภาษา แต่ระบบประมวลผลเงื่อนไขส่วนใหญ่จะใช้ภาษาโปรล็อกเป็นพื้นฐานในการพัฒนา (แสดงแผนภาพระบบประมวลผลเงื่อนไขดังรูปที่ 2.1) ในงานวิจัยนี้จึงเน้นไปที่การโปรแกรมแบบมีเงื่อนไขเชิงตรรกะ (constraint logic programming) ที่ใช้ซอฟต์แวร์ ECLiPSe เป็นสภาพแวดล้อมหลักในการพัฒนาเนื่องจากมีรูปแบบคำสั่งที่เข้าใจได้ง่ายและการเขียนคำสั่งสั้นกะทัดรัด



รูปที่ 2.1 ซอฟต์แวร์สำหรับการพัฒนาโปรแกรมแบบมีเงื่อนไข (Simonis, 2008)

`?- X :: 0..10, Y :: 4..8, X #> Y.`
 	
`X = X{5 .. 10}`
`Y = Y{4 .. 8}`
	`?- X :: 0..5, Y :: 4..8, X #> Y.`
	
	`X = 5`
	`Y = 4`

(a) $X \in [0..10]$ & $Y \in [4..8]$ & $X>Y$	(b) $X \in [0..5]$ & $Y \in [4..8]$ & $X>Y$

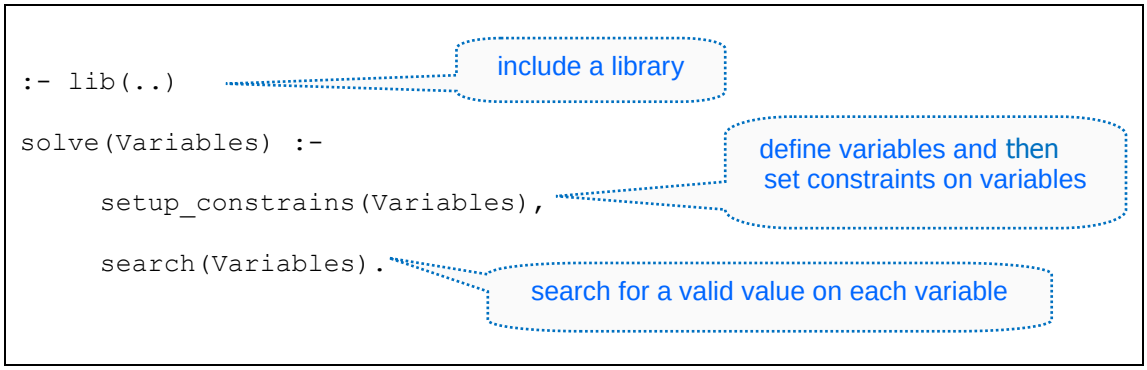
รูปที่ 2.2 ตัวอย่างการใช้เงื่อนไขบังคับในรูปแบบของ ECLiPSe

วิธีการโปรแกรมแบบมีเงื่อนไขจะใช้วิธีการกำหนดขอบเขตค่าของตัวแปร เช่น ในรูปที่ 2.2(a) เป็นการกำหนดว่าตัวแปร X เป็นเลขจำนวนเต็มมีค่าอยู่ระหว่าง 0-10 และตัวแปร Y เป็นเลขจำนวนเต็มมีค่าอยู่ระหว่าง 4-8 เงื่อนไขบังคับคือค่าของตัวแปร X จะต้องมีค่ามากกว่าค่าของตัวแปร Y ดังนั้นระบบประมวลผลเงื่อนไขของ ECLiPSe จึงระบุว่า X จะต้องมีค่าอยู่ระหว่าง 5-10 และ Y จะต้องมีค่าอยู่ระหว่าง 4-8 จึงจะทำให้เงื่อนไขดังกล่าวเป็นจริง โดยระบบประมวลผลเงื่อนไขจะแสดงขอบเขตค่าดังกล่าวในลักษณะของข้อความ X{5..10} และ Y{4..8}

ถ้ามีการเปลี่ยนขอบเขตค่าเริ่มต้นของตัวแปร X เป็น 0-5 โดยที่เงื่อนไขอื่นยังคงเดิม (ดังรูปที่ 2.2(b)) ระบบประมวลผลเงื่อนไขสามารถกำหนดค่าที่แน่นอนได้ทันทีว่า X จะต้องมีค่าเป็น 5 และ Y จะต้องมีค่าเป็น 4

จากตัวอย่างที่แสดงแนวคิดในการโปรแกรมแบบมีเงื่อนไขข้างต้น สามารถสรุปเป็นหลักการทำโปรแกรมแบบมีเงื่อนไขบังคับได้ว่า (Apt & Wallace, 2007; Niederlinski, 2014) การทำโปรแกรมจะประกอบด้วยขั้นตอน 3 ขั้นตอน (แสดงโครงสร้างโปรแกรมดังรูปที่ 2.3) คือ

1) ระบุไลบรารีของระบบประมวลผลเงื่อนไขที่ต้องใช้ในโปรแกรม
2) กำหนดชื่อตัวแปรและขอบเขตค่าในเบื้องต้นของตัวแปรเหล่านั้น
3) เรียกใช้คำสั่งค้นหาเพื่อคำนวณค่าสุดท้ายที่เป็นไปได้ของตัวแปร



รูปที่ 2.3 โครงสร้างของโปรแกรมแบบมีเงื่อนไขในลักษณะของการโปรแกรมเชิงตรรกะ

ซอฟต์แวร์ ECLiPSe มีไลบรารีต่าง ๆ ให้ผู้ใช้เลือกตามแต่ละประเภทของตัวแปร เช่น ตัวแปรที่เป็นเลขจำนวนเต็ม สามารถใช้ไลบรารี fd และ ic เป็นต้น รายละเอียดไลบรารีที่ใช้บ่อยสรุปได้ดังตารางที่ 2.1

ตารางที่ 2.1 ไลบรารีของระบบประมวลผลเงื่อนไขในซอฟต์แวร์ ECLiPSe

Solver library	Variable domains	Constraints class	Behaviour
suspend	numeric	Arbitrary arithmetic in/dis/equalities	Passive test
fd	integer, symbol	Linear in/dis/equalities and some others	Domain propagation
ic	real, integer	Arbitrary arithmetic in/dis/equalities	Bounds/domain propagation
ic_global	integer	N-ary constraints over lists of integers	Bounds/domain propagation
ic_sets	set of integer	Set operations (subset, cardinality, union, …)	Set-bounds propagation
ic_symbolic	ordered symbols	Dis/equality, ordering, element, …	Bounds/domain propagation
propia	inherited	any	various
eplex	real, integer	Linear in/equalities	Global, optimising

2.2.2 ตัวอย่างการโปรแกรมแบบมีเงื่อนไข

การโปรแกรมแบบมีเงื่อนไขใช้โครงสร้างเช่นเดียวกับการโปรแกรมเชิงตรรกะ นั่นคือโปรแกรมจะประกอบด้วยข้อความเชิงตรรกะที่เรียกว่าข้อความแบบฮอร์น (Horn clause) รูปแบบของข้อความแบบฮอร์นจะประกอบด้วยเพรดิเคตที่ส่วนหัวของข้อความและเงื่อนไขที่ส่วนบอดี้ เขียนอยู่ในลักษณะของประโยค “ถ้า-แล้ว” ดังต่อไปนี้

head(Arg) ← body1(Arg1), body2(Arg2), ..., bodyN(ArgN).

ข้อความแบบฮอร์นเป็นการประกาศข้อความที่เป็นจริงแบบมีเงื่อนไข โดยระบุว่าเพรดิเคตที่ส่วนหัวเป็นจริงถ้าเงื่อนไขทุกเงื่อนไขในส่วนบอดี้เป็นจริง เพรดิเคตในภาษาเชิงตรรกะจึงเทียบเคียงได้กับฟังก์ชันในภาษาคอมพิวเตอร์ทั่วไป

วิธีการโปรแกรมแบบมีเงื่อนไขมีลักษณะที่เพิ่มเติมจากการโปรแกรมเชิงตรรกะตรงที่มีการระบุขอบเขตของค่าตัวแปรในส่วนบอดี้ และหลังจากระบุเงื่อนไขบนค่าตัวแปรแล้วมักจะใช้การเรียกเพรดิเคต labeling ให้ทำการค้นหาค่าที่เป็นไปได้ของตัวแปร ค่าที่เป็นไปได้นี้จะเป็นคำตอบของโปรแกรม ตัวอย่างโปรแกรมในรูปที่ 2.4(a) เป็นการค้นหาค่าของชุดตัวแปรสี่ตัวคือ X1, X2, X3 และ X4 ขอบเขตค่าของตัวแปรเหล่านี้คือค่า 1 ถึง 5 โดยมีเงื่อนไขสามเงื่อนไขคือ

(๑) ค่าของตัวแปร X1 จะต้องมีค่ามากกว่าตัวแปร X2

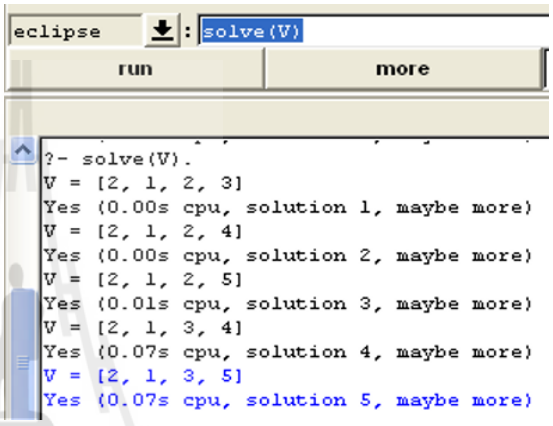
(๒) ค่าของตัวแปร X2, X3, X4 จะต้องไม่ซ้ำกัน

(๓) ค่าของตัวแปร X1 จะต้องไม่เท่ากับค่าของตัวแปร X4

การประมวลผลของโปรแกรมตัวอย่างนี้ใช้ไลบรารี ic หรือ interval constraint เพื่อค้นหาค่าของตัวแปรที่สามารถทำให้เงื่อนไขทั้งหมดข้างต้นเป็นจริง การรันโปรแกรมใช้การเรียกใช้เพรดิเคต solve(V) ผลการรันโปรแกรมด้วย ECLiPSe แสดงได้ดังรูปที่ 2.4(b) จะสังเกตได้ว่าคำตอบที่เป็นไปได้มีหลายคำตอบ คำตอบแรกคือ X1= 2, X2 = 1, X3 = 2, X4 = 3

```
% CLP: example 1
:- lib(ic).
solve(V) :-
      V = [X1, X2, X3, X4],
      V :: 1..5,
      X1 #> X2,
      alldifferent([X2, X3, X4]),
      X1 #\= X4,
      labeling(V).
```

(a) กำหนดค่าให้กับตัวแปรอย่างมีเงื่อนไข

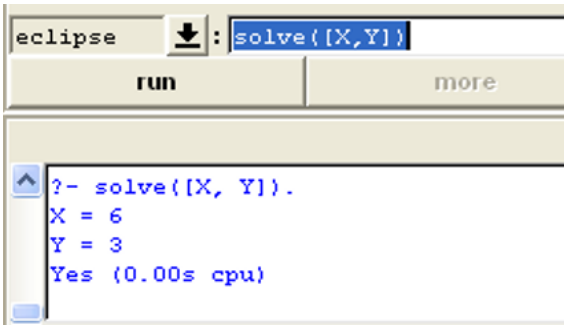


(b) ผลการรันโปรแกรมด้วย ECLiPSe

รูปที่ 2.4 ตัวอย่างการโปรแกรมแบบมีเงื่อนไข

ในกรณีของการแก้โจทย์สมการหลายตัวแปร สามารถใช้วิธีการโปรแกรมแบบมีเงื่อนไขได้เช่นกัน ดังตัวอย่างในรูปที่ 2.5 ที่แสดงการหาค่าของตัวแปร X และ Y โดยที่ทั้งสองตัวแปรมีค่าอยู่ในช่วง 0-9 และกำหนดสมการให้สองสมการคือ X+Y = 9 และ 2X + 4Y = 24 ผลลัพธ์ที่ได้คือ X=6 และ Y=3

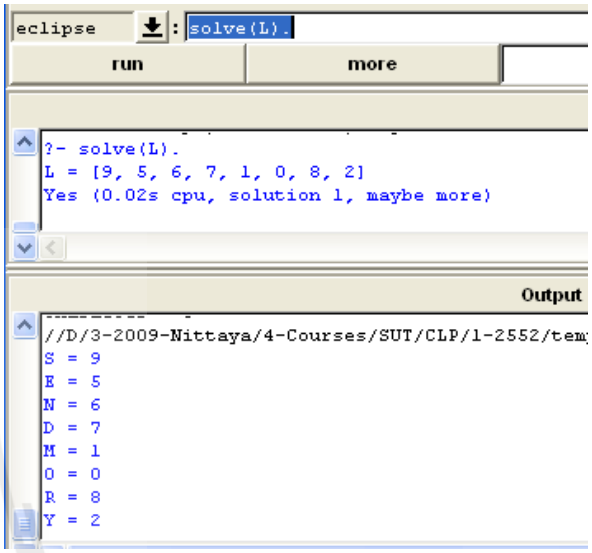
```
% CLP: example 2
%       X, Y ∈ {0, ..., 9}
%       X + Y = 9              (eq1)
%       2X + 4Y = 24           (eq2)

:- lib(ic).

solve([X,Y]) :-
         [X,Y] :: 0..9,
         X+Y #= 9,
         2*X + 4*Y #= 24,
         labeling([X,Y]).
```



รูปที่ 2.5 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้โจทย์สมการหลายตัวแปร

นอกจากการใช้งานในการแก้โจทย์ปัญหาทางคณิตศาสตร์แล้ว การโปรแกรมแบบมีเงื่อนไขยังสามารถใช้ในงานปัญญาประดิษฐ์ ในลักษณะของปัญหาการถอดรหัสตัวเลขที่แสดงด้วยข้อความ (cryptarithmetic problem) ตัวอย่างในรูปที่ 2.6 แสดงการเขียนโปรแกรมแบบมีเงื่อนไขเพื่อค้นหาค่าของแต่ละตัวอักษรที่สามารถทำให้ผลบวก SEND + MORE = MONEY เป็นจริง

```
:- lib(ic).
solve(L) :-
    L= [S, E, N, D, M, O, R, Y],
    L :: 0..9,
    S #\= 0, M #\= 0,
    [R1, R2, R3] :: 0..1,
    alldifferent(L),
    D + E #= Y + 10*R1,
    R1 + N + R #= E + 10*R2,
    R2 + E + O #= N + 10*R3,
    R3 + S + M #= O + 10*M,
    labeling(L),% Enumerate search
    C = ['S','E', 'N', 'D',
         'M','O', 'R', 'Y'],
   (foreach(Character, C),
      foreach(Value, L)
        do  write(Character),
            write(' = '),
            write(Value), nl ) .
```



รูปที่ 2.6 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้โจทย์ SEND + MORE = MONEY

ประโยชน์ที่สำคัญอีกประการหนึ่งของวิธีการโปรแกรมแบบมีเงื่อนไขคือใช้ในการแก้ปัญหาการจัดตาราง (scheduling problem) รูปที่ 2.7 แสดงตัวอย่างการเขียนโปรแกรมแบบมีเงื่อนไขเพื่อจัดสรรคนเข้าทำงานในโครงการต่าง ๆ จำนวนคนที่จะรับการจัดสรรในตัวอย่างนี้มีสี่คน จำนวนโครงการมีสี่โครงการเช่นเดียวกัน แต่ผลประโยชน์ที่หน่วยงานจะได้รับจากการจัดสรรคนให้รับผิดชอบงานในแต่ละโครงการไม่เท่ากัน ตารางที่ 2.2 สรุปมูลค่าผลประโยชน์ที่หน่วยงานจะได้รับจากการจัดสรรคนสี่คน (W1, W2, W3, W4) เข้ารับผิดชอบในโครงการสี่โครงการ (T1, T2, T3, T4) เงื่อนไขของการจัดคนเข้าทำงานคือ คนทำงานหนึ่งคนจะรับผิดชอบเพียงหนึ่งโครงการเท่านั้น และผลประโยชน์โดยรวมที่หน่วยงานได้รับจะต้องไม่ต่ำกว่า 19 ผลลัพธ์ของการรันโปรแกรม (ด้านขวาของรูปที่ 2.7) แสดงว่าโจทย์ปัญหานี้มีได้มากกว่าหนึ่งคำตอบ โดยคำตอบแรกที่แสดงคือ กำหนดให้ W1 รับผิดชอบโครงการ T1, W2 รับผิดชอบโครงการ T2, W3 รับผิดชอบโครงการ T3, W4 รับผิดชอบโครงการ T4 จะทำให้ได้รับผลประโยชน์ = 7+2+7+3 = 19

ตารางที่ 2.2 ผลประโยชน์ที่จะได้รับจากการจัดสรรคนรับผิดชอบแต่ละโครงการ

	T1	T2	T3	T4
W1	7	1	3	4
W2	8	2	5	1
W3	4	3	7	2
W4	3	1	6	3

```
assign(Workers) :-
     Workers = [W1, W2, W3, W4],
     Workers::1..4,
     alldifferent(Workers),

        % extract profit of each
        % worker on each project
     element(W1, [7, 1, 3, 4], WP1),
     element(W2, [8, 2, 5, 1], WP2),
     element(W3, [4, 3, 7, 2], WP3),
     element(W4, [3, 1, 6, 3], WP4),

     P #= WP1 + WP2 + WP3 + WP4,
     P #>= 19,

     labeling(Workers),
     write('Assigning scheme = '),
     writeln(Workers),
     write('   Profit = '),
     writeln(P).
```

```
?- assign(L).
L = [1, 2, 3, 4]
Yes (0.00s cpu, solution 1, maybe more)
```

```
Assigning scheme = [1, 2, 3, 4]
        Profit = 19
```

	T1	T2	T3	T4
W1	7	1	3	4
W2	8	2	5	1
W3	4	3	7	2
W4	3	1	6	3

รูปที่ 2.7 การโปรแกรมแบบมีเงื่อนไขเพื่อแก้ปัญหาการจัดตารางทำงาน

บทที่ 3
การออกแบบและพัฒนาโปรแกรม Top-K-patterns

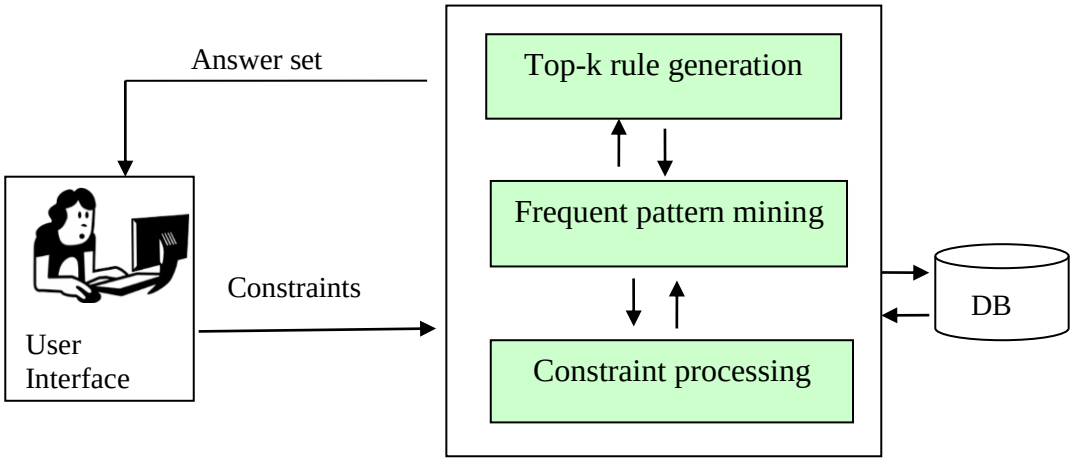
3.1 กรอบของงานวิจัย

โครงสร้างของโปรแกรมค้นหารูปแบบที่ปรากฏบ่อย เค อันดับแรก (หรือเรียกชื่อย่อว่าโปรแกรม Top-K-patterns) ประกอบด้วยส่วนประกอบหลัก 3 ส่วน (ดังแสดงในรูปที่ 3.1) คือ

- ส่วนประมวลผลเงื่อนไขบังคับที่ระบุโดยผู้ใช้ (constraint processing)
- ส่วนค้นหารูปแบบที่ปรากฏบ่อยจากข้อมูลในฐานข้อมูล (frequent pattern mining) โดยรูปแบบนั้นจะต้องตรงตามเงื่อนไขที่ผู้ใช้ระบุ
- ส่วนที่ทำหน้าที่สร้างกฎความสัมพันธ์จากรูปแบบที่ปรากฏบ่อยด้วยความถี่สูงสุด เค อันดับแรก (top-k rule generation)

ในส่วนติดต่อกับผู้ใช้ (user interface) โปรแกรม Top-K-patterns จะใช้กรอบหน้าต่าง query entry ซึ่งเป็นกรอบหน้าต่างพื้นฐานของซอฟต์แวร์ ECLiPSe ร่วมกับการสร้างกรอบหน้าต่างโต้ตอบเพื่อให้ผู้ใช้ระบุเงื่อนไขต่าง ๆ ที่ต้องการให้ปรากฏในผลลัพธ์

เงื่อนไขที่ผู้ใช้ระบุจะมีความยืดหยุ่น โดยผู้ใช้อาจระบุเพียงค่าพารามิเตอร์ K ที่หมายถึงจำนวนรูปแบบความสัมพันธ์ในลักษณะกฎถ้า-แล้ว ที่มีค่าความเชื่อมั่นและค่าสนับสนุนเรียงลำดับจากค่ามากมาค่าน้อย จำนวนสูงสุด เค อันดับแรก หรือผู้ใช้อาจระบุพารามิเตอร์อื่น ๆ เพิ่มเติม ได้แก่ชื่อไอเท็มที่สนใจและต้องการให้ปรากฏในผลลัพธ์ ค่าสนับสนุนขั้นต่ำและค่าความเชื่อมั่นขั้นต่ำของกฎ



รูปที่ 3.1 โครงสร้างของส่วนประกอบในโปรแกรม Top-K-patterns

3.2 การออกแบบและพัฒนาโปรแกรม

การออกแบบโปรแกรม Top-K-patterns ใช้แนวทางการค้นหารูปแบบที่ปรากฏบ่อยของอัลกอริทึม Apriori (Agrawal and Srikant, 1994) เป็นพื้นฐานในการพัฒนา ขั้นตอนวิธีของโปรแกรม Top-K-patterns แสดงได้ดังรูปที่ 3.2

Algorithm Top-K-patterns

Input: Database D

Output: Association rules set S

Step:

(1) Read the user-specified main constraint: K
and other possible constraints: RuleLength, Minsup, Minconf, Inclusive_item, Exclusive_item, Target_item

(2) Scan D to generate frequent itemset L

(3) Pruning L according to the Inclusive_item, Exclusive_item, Target_item, and Minsup constraints

(4) Generate association rule set S from the pruned frequent itemset L

(5) Rank rules in S in descending order according to the confidence_support score

(6) Output the first k rules of S

รูปที่ 3.2 ขั้นตอนวิธีของโปรแกรม Top-K-patterns

ขั้นตอนแรกของโปรแกรม Top-K-patterns จะเป็นการอ่านเงื่อนไขบังคับที่ผู้ใช้ระบุ เงื่อนไขนี้จำแนกเป็นเงื่อนไขที่จำเป็นต้องระบุ ได้แก่ ค่า K ที่หมายถึงจำนวนกฎที่มีค่าความเชื่อมั่นและค่าสนับสนุนสูงสุด เค อันดับแรก และเงื่อนไขที่ผู้ใช้อาจจะระบุเพิ่มเติม ได้แก่ ขนาดของกฎ (หรือ RuleLength ซึ่งวัดจากจำนวนไอเท็มที่ปรากฏในกฎ) ค่าสนับสนุนขั้นต่ำ (Minsup) ค่าความเชื่อมั่นขั้นต่ำ (Minconf) ไอเท็มที่ต้องการให้ปรากฏในกฎ (Inclusive_item) โดยอาจจะปรากฏในส่วนใดของกฎก็ได้ ไอเท็มที่ไม่ต้องการให้ปรากฏในกฎ (Exclusive_item) และไอเท็มที่ต้องการให้ปรากฏในส่วนผลสรุปของกฎ (Target_item) เงื่อนไขเพิ่มเติมเหล่านี้ถ้าผู้ใช้ไม่ระบุโปรแกรมจะใช้ค่าที่เตรียมไว้ก่อนล่วงหน้า

ในขั้นตอนที่สองของโปรแกรม เป็นการอ่านฐานข้อมูลเพื่อสร้างเซตของไอเท็มที่ปรากฏบ่อย ขั้นตอนนี้จะใช้การทำงานเช่นเดียวกับอัลกอริทึม Apriori (Agrawal and Srikant, 1994) หลังจากได้เซตของไอเท็มที่ปรากฏบ่อย ในขั้นตอนที่สามจะเป็นการตัดทิ้งไอเท็มเซตตามเงื่อนไขเพิ่มเติมที่ผู้ใช้ระบุ

ขั้นตอนที่สี่เป็นการนำไอเท็มเซตแต่ละเซตมาสร้างกฎในลักษณะของข้อความเชิงตรรกะ “ถ้า X แล้ว Y” เพื่อแสดงความสัมพันธ์ของการเกิดขึ้นบ่อยร่วมกันของไอเท็ม X และ Y จากนั้นในขั้นตอนที่ห้าจะเป็นการเรียงลำดับกฎถ้า-แล้ว ตามค่าความเชื่อมั่นและค่าสนับสนุน โดยเรียงลำดับจากค่ามากไปน้อย ในขั้นตอนสุดท้ายจะเป็นการคัดเลือกกฎที่อยู่ในลำดับสูงสุด เค ลำดับแรก แสดงเป็นผลลัพธ์ให้ผู้ใช้ทราบ

3.3 ตัวอย่างการใช้งานโปรแกรม

การนำโปรแกรม Top-K-patterns ไปใช้งาน ผู้ใช้จะต้องเตรียมข้อมูลเข้าให้อยู่ในรูปแบบของข้อความเชิงตรรกะ โดยข้อมูลทั้งฐานข้อมูลจะเป็นอาร์กิวเมนต์หนึ่งของเพรดิเคต data และระบุชื่อไอเท็มทั้งหมดที่เป็นไปได้ไว้ภายในอาร์กิวเมนต์แรกของเพรดิเคต รูปที่ 3.3 แสดงตัวอย่างข้อมูลในรูปแบบของข้อความเชิงตรรกะ ข้อมูลนี้แปลงมาจากฐานข้อมูลทรานแซคชันที่แสดงประกอบไว้ด้านซ้ายมือของภาพ

โครงสร้างที่ใช้บันทึกฐานข้อมูล จะอยู่ในรูปแบบของ

data([list of items] — [list of each transaction]).

โดยแต่ละไอเท็มจะอยู่ภายในลิสต์และแต่ละทรานแซคชันจะอยู่ภายในลิสต์เช่นเดียวกัน ทำให้ลักษณะข้อมูลที่ปรากฏเป็นลิสต์ซ้อนอยู่ภายในลิสต์ เช่น [[beer]] ลิสต์ของไอเท็มและลิสต์ของทรานแซคชันจะแยกออกจากกันด้วยเครื่องหมาย “—” ทั้งนี้เพื่อความสะดวกในการประมวลผลแบบ pattern matching ซึ่งเป็นจุดเด่นของภาษาเชิงตรรกะ

TID	Items
1	{Cereal, Milk}
2	{Beer, Cereal, Diaper, Egg}
3	{Beer, Diaper, Milk}
4	{Beer, Cereal, Diaper, Milk}
5	{Diaper, Milk}

```
data(
[ [beer], [cereal], [diaper], [milk], [egg] ]
—
[ [cereal, milk],
  [beer, cereal, diaper, egg],
  [beer, diaper, milk],
  [beer, cereal, diaper, milk],
  [diaper, milk] ] ).
```

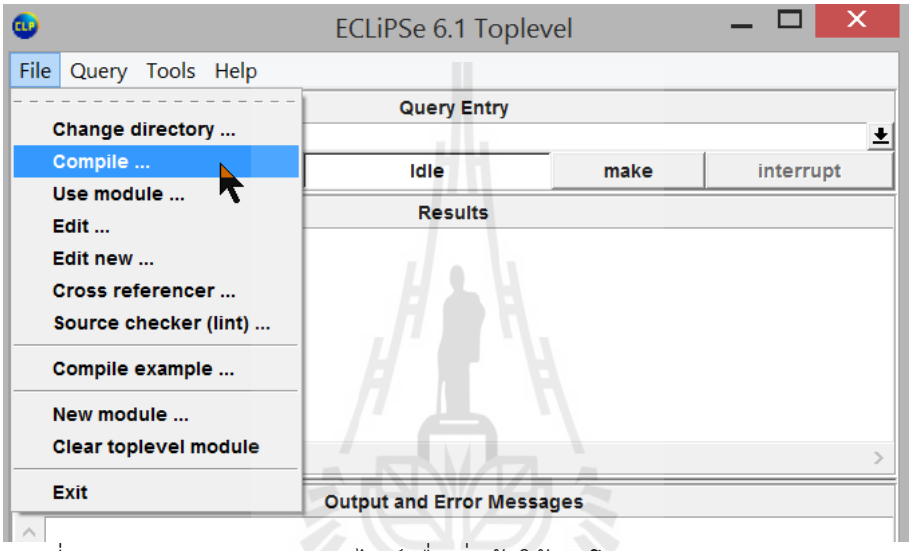
รูปที่ 3.3 รูปแบบข้อมูลเข้าของโปรแกรม Top-K-patterns

เมื่อแปลงฐานข้อมูลทรานแซคชันเป็นข้อความเชิงตรรกะแล้ว บันทึกเป็นไฟล์ข้อความ (เช่น data.text) จากนั้นเปิดไฟล์และคอมไพล์โปรแกรม Top_K_patterns ด้วยซอฟต์แวร์ ECLiPSe (ดังรูปที่ 3.4) การสอบถามข้อมูลจะใช้การป้อนข้อคำถามที่กรอบข้อความ Query Entry ดังแสดงใน รูปที่ 3.5 ซึ่งเป็นการสอบถามกฎความสัมพันธ์ 3 อันดับแรก ผลลัพธ์ที่ได้คือกฎความสัมพันธ์ต่อไปนี้

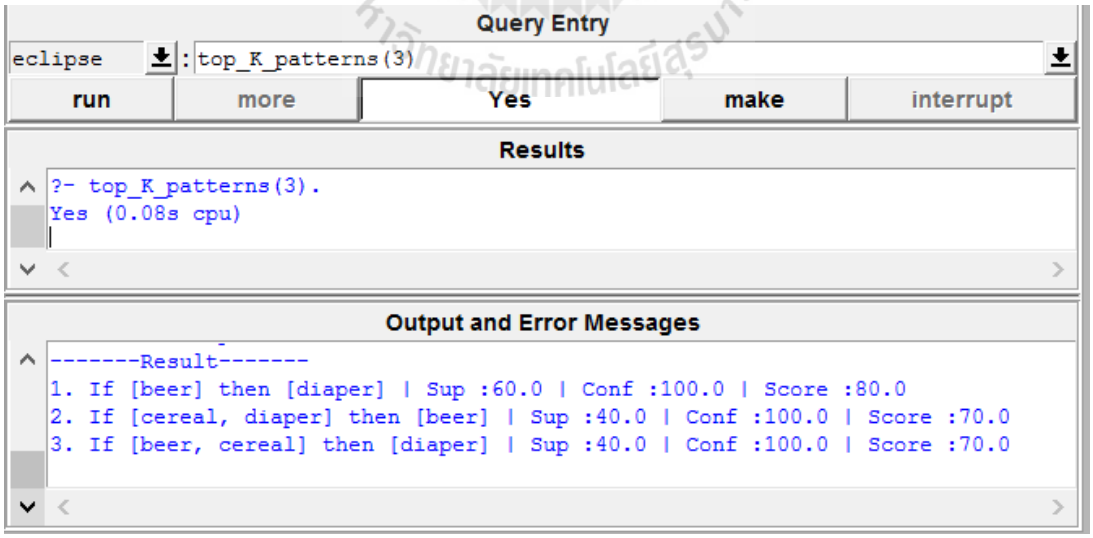
(๑) ถ้าลูกค้าซื้อเบียร์ แล้วจะซื้อผ้าอ้อมเด็ก (ค่าสนับสนุน 60% ความเชื่อมั่น 100% เฉลี่ย 80%)

(๒) ถ้าลูกค้าซื้อซีเรียลและผ้าอ้อมเด็ก แล้วจะซื้อเบียร์ (ค่าสนับสนุน 40% ความเชื่อมั่น 100% เฉลี่ย 70%)

(๓) ถ้าลูกค้าซื้อเบียร์และซีเรียล แล้วจะซื้อผ้าอ้อมเด็ก (ค่าสนับสนุน 40% ความเชื่อมั่น 100% เฉลี่ย 70%)

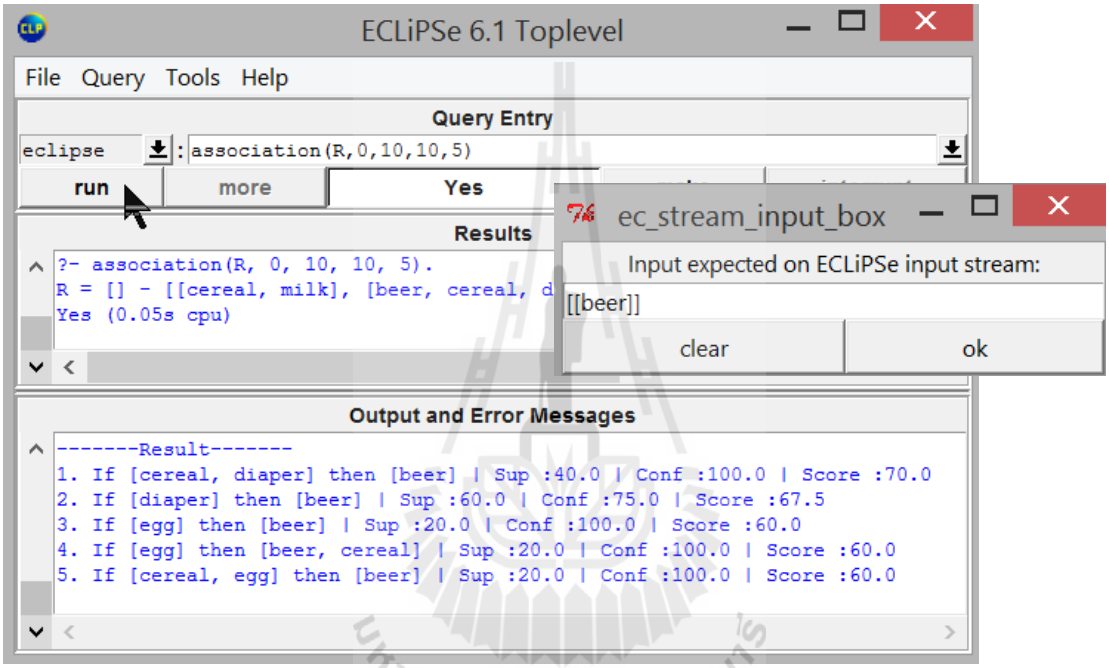


รูปที่ 3.4 จอภาพแสดงการคอมไพล์เพื่อเริ่มต้นใช้งานโปรแกรม Top-K-patterns

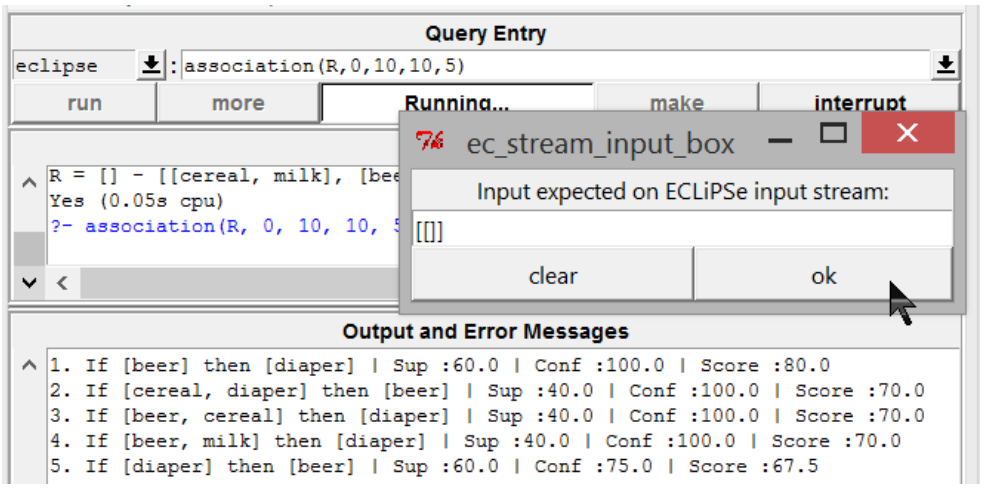


รูปที่ 3.5 การสอบถามข้อมูลเพื่อแสดงผลลัพธ์เป็นกฎความสัมพันธ์ 3 อันดับแรก

ในกรณีที่ผู้ใช้ต้องการระบุเงื่อนไขบังคับเพิ่มเติมนอกเหนือจากการระบุค่า เค สามารถทำได้ด้วยการเรียกใช้เพรดิเคต association(OutputRuleSet, RuleLength, Minsup, Minconf, K) จากนั้นคลิกที่ปุ่มคำสั่ง run โดยในรูปที่ 3.6 แสดงตัวอย่างการค้นหากฎความสัมพันธ์ห้าอันดับแรกที่มีเบียร์เป็นเป้าหมายของกฎ และกำหนดเงื่อนไขขนาดของกฎต้องมากกว่า 0 ค่าสนับสนุนขั้นต่ำและค่าความเชื่อมั่นขั้นต่ำเป็น 10 การกำหนดไอเท็มที่ต้องการให้ปรากฏในกฎใช้วิธีระบุผ่านกล่องโต้ตอบ จากนั้นกด ok ในรูปที่ 3.7 แสดงผลลัพธ์เพื่อเป็นการเปรียบเทียบว่าถ้าผู้ใช้ไม่ระบุเป้าหมายของกฎ (ด้วยการใช้สัญลักษณ์ลิสต์ว่างซึ่งแทนด้วย [[]] ในกล่องข้อความและกด ok) แต่เงื่อนไขอื่น ๆ เป็นเช่นเดียวกันจะได้ผลลัพธ์ที่แตกต่างออกไป



รูปที่ 3.6 การสอบถามข้อมูลเพื่อแสดงกฎความสัมพันธ์ 5 อันดับแรกที่มีเบียร์เป็นเป้าหมายของกฎ



รูปที่ 3.7 การสอบถามเพื่อแสดงกฎความสัมพันธ์ 5 อันดับแรกโดยไม่ระบุเป้าหมายของกฎ

การระบุเงื่อนไขเกี่ยวกับไอเท็มที่ต้องการหรือไม่ต้องการให้ปรากฏในผลลัพธ์ โปรแกรม Top-K-patterns ใช้วิธีสร้างกล่องข้อความโต้ตอบ 3 ครั้ง เพื่อให้ผู้ใช้สามารถระบุเงื่อนไขได้ 3 กรณี คือ

(๑) การระบุว่าต้องมีไอเท็มที่ผู้ใช้สนใจ จะเป็นกล่องข้อความแรกที่ปรากฏขึ้นที่หน้าจอ ถ้าผู้ใช้ไม่มีความสนใจในไอเท็มใดเป็นพิเศษ ให้พิมพ์สัญลักษณ์ [[]] แล้วกด ok แต่ถ้าต้องการระบุว่าในกฎความสัมพันธ์ต้องมีไอเท็มเบียร์และซีเรียล ให้พิมพ์ [[beer], [cereal]] ในกรณีที่ต้องการระบุว่าในกฎความสัมพันธ์ต้องปรากฏเบียร์หรือซีเรียล ให้พิมพ์ [[beer, cereal]]

(๒) การระบุไอเท็มที่ไม่ต้องการให้ปรากฏในกฎ จะระบุในกล่องข้อความที่สองที่จะปรากฏขึ้นที่หน้าจอ รูปแบบการระบุจะเหมือนกับในกรณีแรก

(๓) การระบุเฉพาะส่วนเป้าหมายของกฎ (ส่วนที่ปรากฏหลัง then ในกฎความสัมพันธ์) จะระบุในกล่องข้อความที่สาม

บทที่ 4
ผลการทดสอบโปรแกรม Top-K-patterns

การทดสอบความถูกต้องและประสิทธิภาพของโปรแกรม Top-K-patterns ที่พัฒนาขึ้นใหม่ในงานวิจัยนี้ จะใช้วิธีเปรียบเทียบกับผลลัพธ์ที่ได้จากโปรแกรม Apriori (Agrawal and Srikant, 1994) ซึ่งเป็นโปรแกรมมาตรฐานที่มักจะใช้ในการค้นหารูปแบบที่ปรากฏบ่อยและแสดงผลรูปแบบนั้นในลักษณะของกฎความสัมพันธ์ ลักษณะการเปรียบเทียบจะใช้การเทียบเคียงจำนวนกฎและไอเท็มที่ปรากฏในกฎที่ได้จากโปรแกรม Top-K-patterns และโปรแกรม Apriori โดยในงานวิจัยนี้จะเขียนโปรแกรม Apriori ในลักษณะของการโปรแกรมเชิงตรรกะที่ใช้เงื่อนไขบังคับและประมวลผลบนซอฟต์แวร์ ECLiPSe ทั้งนี้เพื่อให้โปรแกรม Top-K-patterns และ Apriori มีสภาพแวดล้อมที่ใกล้เคียงกันมากที่สุดและลดความลำเอียงในการเปรียบเทียบ

4.1 ข้อมูลที่ใช้ในการทดสอบ

ในการทดสอบความถูกต้องและประสิทธิภาพของโปรแกรม Top-K-patterns งานวิจัยนี้ใช้ข้อมูล Churn (ดาวน์โหลดได้จาก https://www.sgi.com/tech/mlc/db/ และเลือกข้อมูล churn.data) ข้อมูลนี้บันทึกรายละเอียดลูกค้าของบริษัทโทรศัพท์แห่งหนึ่งในประเทศสหรัฐอเมริกาเพื่อใช้วิเคราะห์ลักษณะของลูกค้าที่มีแนวโน้มจะยกเลิกการใช้บริการของบริษัท (churn = true)

ข้อมูล Churn ประกอบด้วย 3333 เรคคอร์ด ข้อมูลในแต่ละเรคคอร์ดบันทึกรายละเอียดพฤติกรรมการใช้โทรศัพท์ของลูกค้าแต่ละราย (คำอธิบายข้อมูลสรุปในตารางที่ 4.1) พฤติกรรมการใช้โทรศัพท์บันทึกในแนวคอลัมน์ (feature, variable, or attribute) ประกอบด้วย 21 คอลัมน์ แต่ในงานวิจัยนี้เลือกใช้เพียง 12 คอลัมน์ เนื่องจากการวิเคราะห์ข้อมูลเบื้องต้นพบว่ารายละเอียดใน 9 คอลัมน์มีความสำคัญน้อย และคอลัมน์เหล่านี้ไม่ส่งผลต่อการวิเคราะห์ว่าลูกค้าจะยกเลิกการใช้บริการของบริษัทหรือไม่ (นั่นคือ churn = true หมายถึงยกเลิกบริการ หรือ churn = false หมายถึงไม่ยกเลิกบริการ)

การเตรียมข้อมูลเพื่อใช้ในงานวิจัยนี้จะต้องมีการแปลงรูปแบบข้อมูล และจะต้องเปลี่ยนข้อมูลที่เป็นค่าตัวเลขให้เป็นค่าเชิงกลุ่ม (categorical or nominal) ทั้งนี้เนื่องจากการวิเคราะห์เพื่อหากฎความสัมพันธ์ใช้วิธีการนับเป็นพื้นฐานในการวิเคราะห์ จึงไม่สามารถทำงานกับข้อมูลที่เป็นค่าตัวเลขต่อเนื่องได้ ลักษณะของข้อมูลที่แปลงรูปแบบแล้วแสดงตัวอย่างบางส่วนได้ดังรูปที่ 4.1 ที่แสดงส่วนของการบรรยายไอเท็ม (หรือค่าที่เป็นไปได้ในแต่ละคอลัมน์ โดยแสดงรายละเอียดเฉพาะคอลัมน์ vMailPlan และ vMailMessage) และรูปที่ 4.2 ที่แสดงส่วนข้อมูลโดยแสดงเพียงสามเรคคอร์ด

ตารางที่ 4.1 รายละเอียดข้อมูลลูกค้าบริษัทผู้ให้บริการโทรศัพท์

ชื่อตัวแปร	ชนิดข้อมูล	คำอธิบาย
state	discrete	Name of 50 states and District of Columbia
accountLength	continuous	How long account has been active
areaCode	continuous	
intlPlan	discrete	Dichotomous categorical, yes or no
vMailPlan	discrete	Dichotomous categorical, yes or no
vMailMessage	continuous	Number of voice mail messages
dayCalls	continuous	Number of times customer used service during day time
eveCalls	continuous	Number of times customer used service during evening
nightCalls	continuous	Number of times customer used service during the night
intlCalls	continuous	Number of times customer used service for international calls
custServCalls	continuous	Number of times customer called the customer service
churn	discrete	Dichotomous categorical, true or false

```
data([
[vMailPlan_no], [vMailPlan_yes],
[vMailMessage_25],[vMailMessage_26],[vMailMessage_0],[vMailMessage_24],
[vMailMessage_37],[vMailMessage_27],[vMailMessage_33],[vMailMessage_39],
[vMailMessage_30],[vMailMessage_41],[vMailMessage_28],[vMailMessage_34],
[vMailMessage_46],[vMailMessage_29],[vMailMessage_35],[vMailMessage_21],
[vMailMessage_32],[vMailMessage_42],[vMailMessage_36],[vMailMessage_22],
[vMailMessage_23],[vMailMessage_43],[vMailMessage_31],[vMailMessage_38],
[vMailMessage_40],[vMailMessage_48],[vMailMessage_18],[vMailMessage_17],
[vMailMessage_45],[vMailMessage_16],[vMailMessage_20],[vMailMessage_14],
[vMailMessage_19],[vMailMessage_51],[vMailMessage_15],[vMailMessage_11],
[vMailMessage_12],[vMailMessage_47],[vMailMessage_8],[vMailMessage_44],
[vMailMessage_49],[vMailMessage_4],[vMailMessage_10],[vMailMessage_13],
[vMailMessage_50],[vMailMessage_9], ... ] – [ // data part //] ).
```

รูปที่ 4.1 โครงสร้างข้อมูลส่วนระบุรายละเอียดของไอเท็ม

```
data([ // item part // ]
– [
[state_KS,accountLength_128,areaCode_415,intlPlan_no,vMailPlan_yes,vMailMessage_25,
        dayCalls_110,eveCalls_99,nightCalls_91,intlCalls_3,custServCalls_1,churn_False],
[state_OH,accountLength_107,areaCode_415,intlPlan_no,vMailPlan_yes,vMailMessage_26,
        dayCalls_123,eveCalls_103,nightCalls_103,intlCalls_3,custServCalls_1,churn_False],
[state_IN,accountLength_65,areaCode_415,intlPlan_no,vMailPlan_no,vMailMessage_0,
        dayCalls_137,eveCalls_83,nightCalls_111,intlCalls_6,custServCalls_4,churn_True],
... ] ).
```

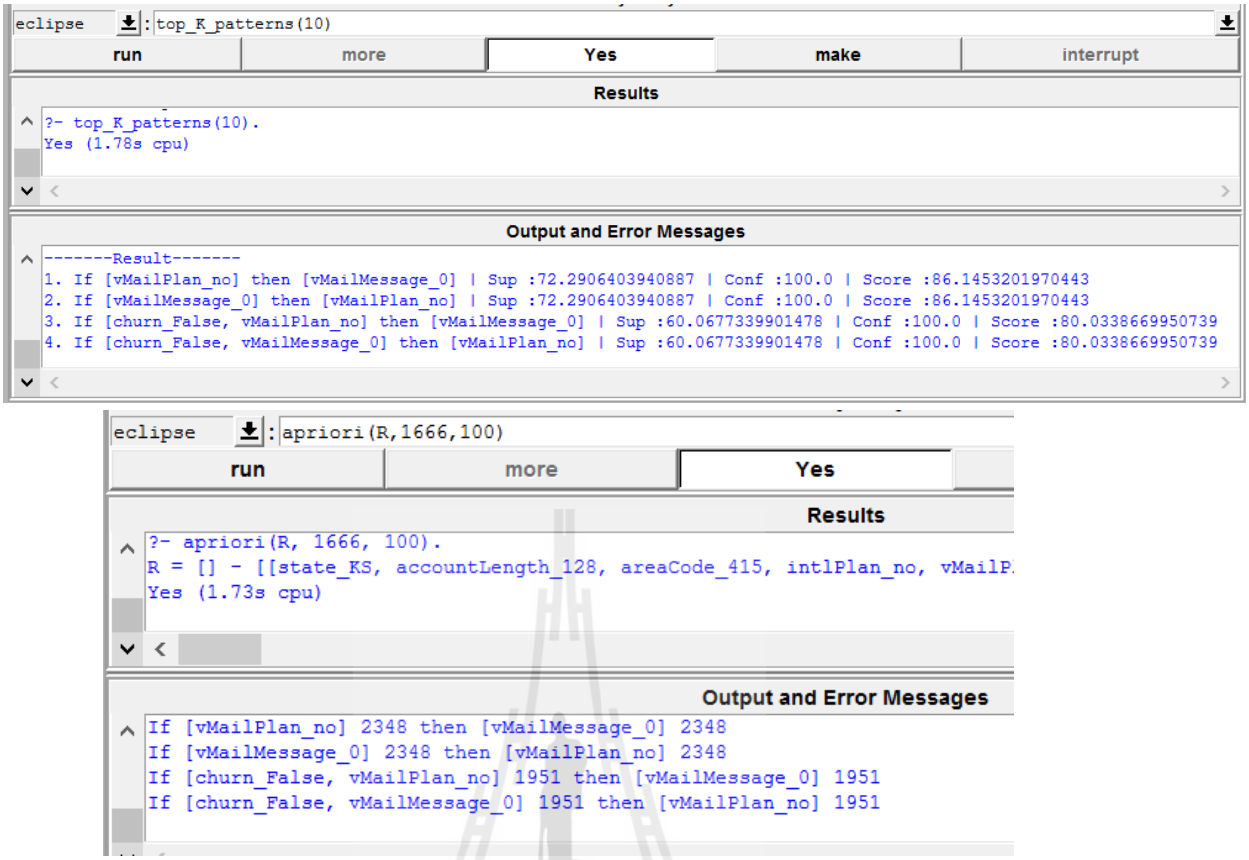
รูปที่ 4.2 ตัวอย่างข้อมูลสามเรคคอร์ด

4.2 ผลการทดสอบ

การทดสอบโปรแกรมแบ่งเป็นการทดสอบความถูกต้องและการทดสอบประสิทธิภาพ ในส่วนของการทดสอบความถูกต้องใช้วิธีการเปรียบเทียบจำนวนกฎและชื่อไอเท็มที่ปรากฏในกฎที่ได้รับจากโปรแกรม Top-K-patterns เปรียบเทียบว่าตรงกันกับกฎที่ได้รับจากโปรแกรม Apriori หรือไม่ โดยกำหนดค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำของทั้งสองโปรแกรมให้ตรงกัน ผลลัพธ์ที่ได้แสดงดังรูปที่ 4.3 ซึ่งได้จากการกำหนด Minsup = 50% และ Minconf = 100% (ในโปรแกรม Apriori ค่าสนับสนุน 50% จะหมายถึงจำนวนข้อมูลขั้นต่ำ 1666 เรคคอร์ด) ผลที่ได้มีจำนวนกฎและรายละเอียดไอเท็มตรงกัน จึงยืนยันได้ว่าโปรแกรม Top-K-patterns ทำงานได้ถูกต้อง จำนวนกฎที่ได้มี 4 กฎ คือ

IF (vMailPlan=no) THEN (vMailMessage=0)
IF (vMailMessage=0) THEN (vMailPlan=no)
IF (churn=False & vMailPlan=no) THEN (vMailMessage=0)
IF (churn=False & vMailMessage=0) THEN (vMailPlan=no)

การทำงานของโปรแกรม Top-K-patterns จะใช้เวลาในการประมวลผล 1.78 วินาที ในขณะที่โปรแกรม Apriori ใช้เวลา 1.73 วินาที เวลาที่โปรแกรม Top-K-patterns ต้องใช้มากกว่าเนื่องจากเมื่อค้นหาไอเท็มเซตที่ปรากฏบ่อยแล้วสร้างเป็นกฎความสัมพันธ์ จะต้องเรียงลำดับกฎตามคะแนนเฉลี่ยของค่าความเชื่อมั่นและค่าสนับสนุน จากนั้นคัดเลือกกฎที่มีคะแนนสูงสุด เค อันดับแรก ซึ่งโปรแกรม Apriori จะไม่มีขั้นตอนการคัดเลือกกฎ เค อันดับแรก จึงทำให้โปรแกรม Apriori ประมวลผลได้เร็วกว่าเล็กน้อย



รูปที่ 4.3 ผลการทดสอบความถูกต้องของโปรแกรม Top-K-patterns (ภาพบน) เปรียบเทียบกับโปรแกรม Apriori (ภาพล่าง)

ในขั้นตอนการทดสอบประสิทธิภาพของกฎความสัมพันธ์ที่ได้ เมื่อผู้ใช้ระบุเงื่อนไขบังคับต่าง ๆ ได้ทดสอบกับการใช้ข้อคำถามจำนวน 8 ข้อคำถาม (ข้อคำถามที่ 2-8 ใช้ค่า Minsup และ Minconf เช่นเดียวกับข้อคำถามแรก) ดังนี้

Query 1: Rules are to be induced with the thresholds:

- minimum support (Minsup) = 1.5% (or 50 records from the total of 3333)
- minimum confidence (Minconf) = 80%.

Query 2: Rules must contain the feature churn_False (or non-churner).

Query 3: Rules must have at least three items.

Query 4: Rules must NOT contain the feature ‘churn_False’.

Query 5: Rules must contain the feature ‘churn_False’ as the target of the rule.

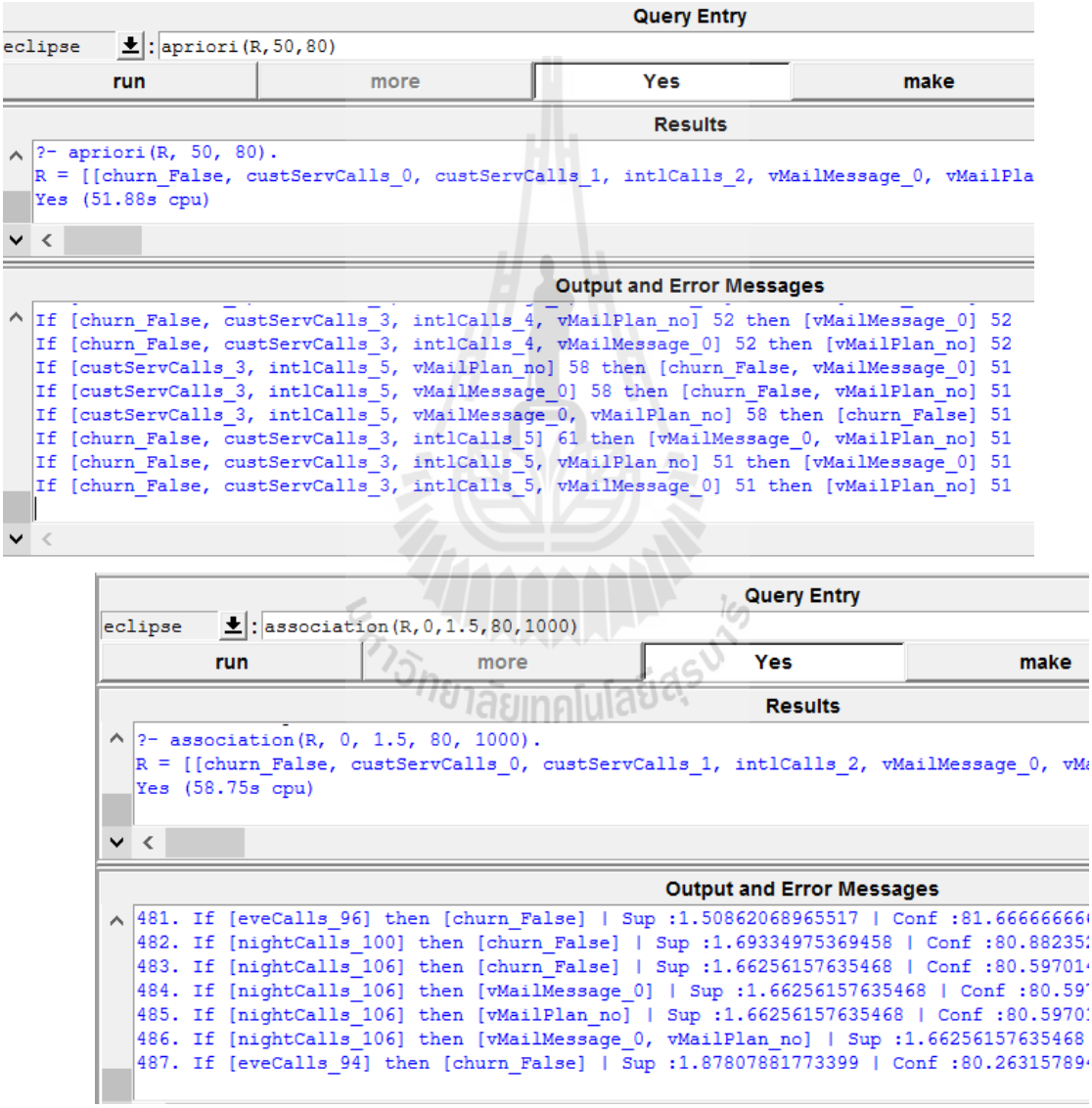
Query 6: Rules must contain either the feature ‘churn_False’, OR ‘churn_True’.

Query 7: Rules must contain both the features ‘churn_True’ AND ‘vMailPlan_no’.

Query 8: Rules must satisfy these constraints:

- has at least three items,
- must contain both ‘churn_False’ AND ‘vMailPlan_no’,
- must NOT contain either the feature ‘vMailMessage_0’, OR ‘intlCalls_2’,
- the target clause of the rules must be ‘churn_False’.

ผลการทดสอบข้อคำถามที่ 1 ด้วยโปรแกรม Apriori และโปรแกรม Top-K-patterns แสดงภาพเปรียบเทียบได้ดังรูปที่ 4.4

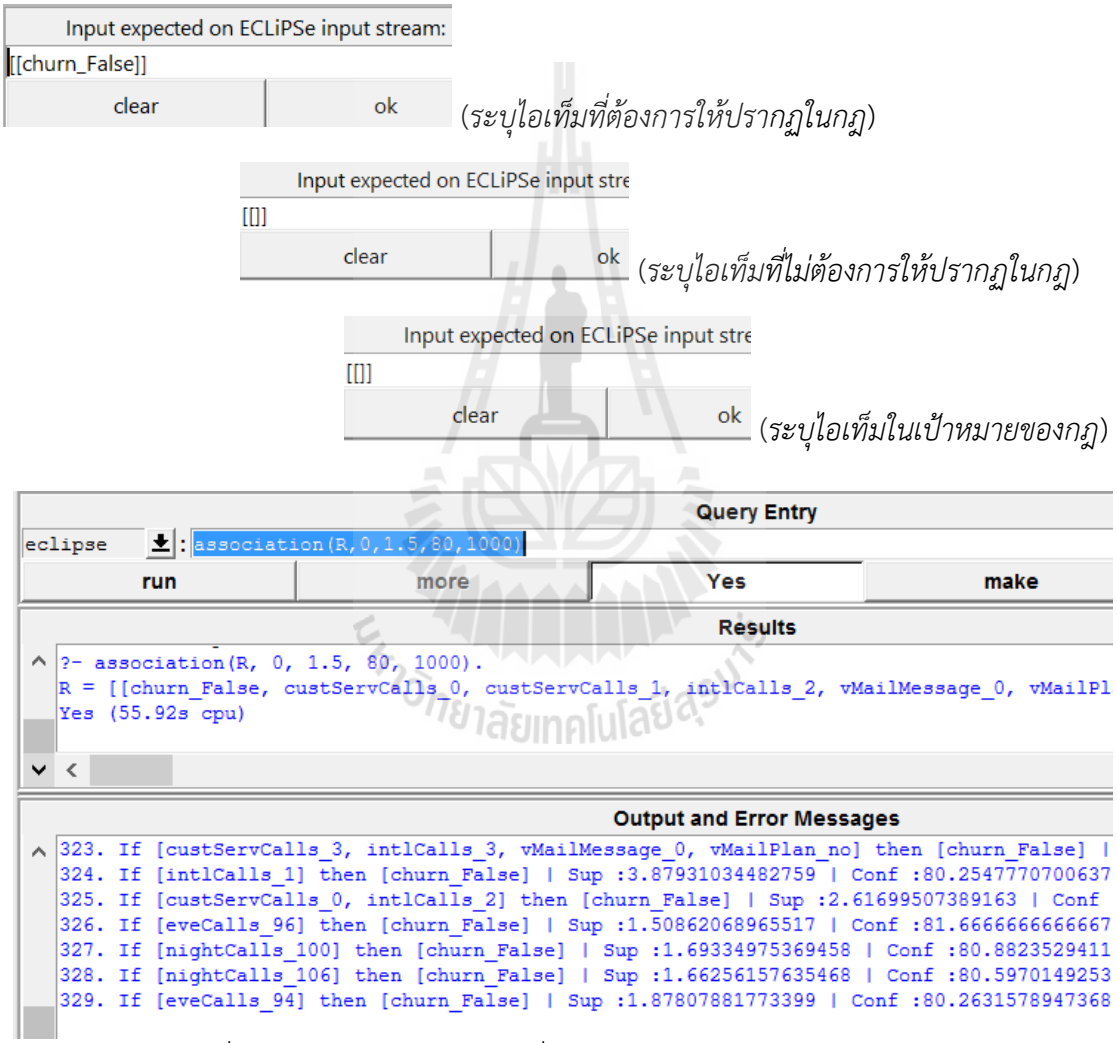


รูปที่ 4.4 การทดสอบข้อคำถามที่ 1 ด้วยโปรแกรม Apriori และโปรแกรม Top-K-patterns

(Query 1: Minsup=1.5% or 50 records, Minconf=80%)

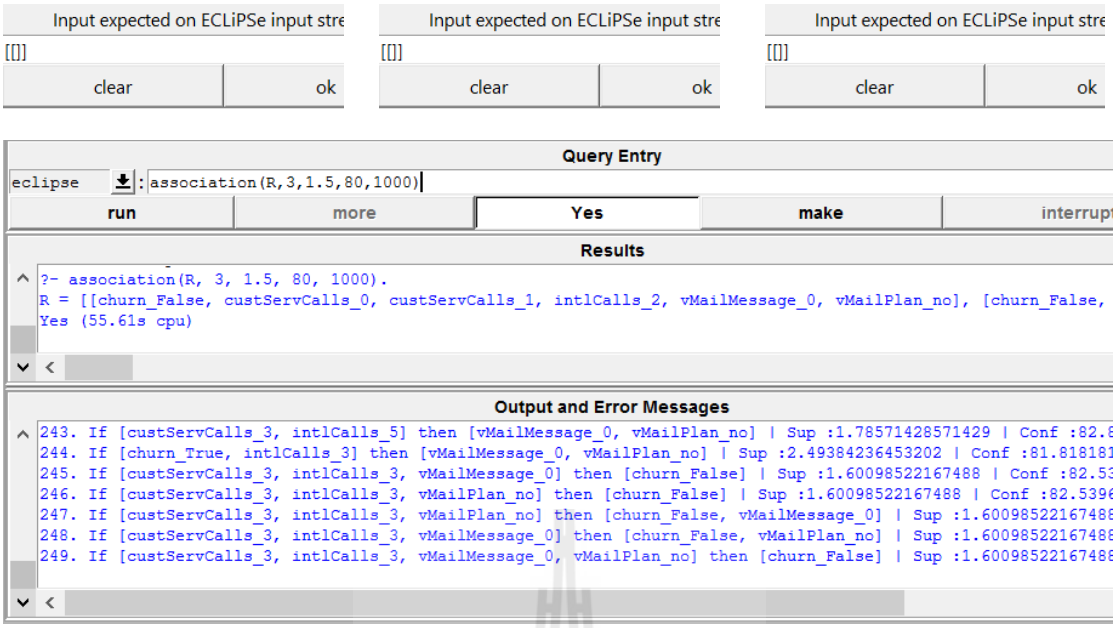
ในกรณีของข้อคำถามที่ 2-8 โปรแกรม Apriori ไม่สามารถประมวลผลได้เนื่องจากโปรแกรม Apriori ไม่ได้ถูกออกแบบให้ระบุเงื่อนไขบังคับอื่น ๆ นอกเหนือจากค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำ ดังนั้นการทดสอบข้อคำถามที่ 2-8 จึงทำได้เฉพาะกับโปรแกรม Top-K-patterns แสดงผลที่ได้ดังรูปที่ 4.5-4.11 โดยในแต่ละภาพจะแสดงกล่องข้อความที่ระบุไอเท็มที่ต้องการ ไอเท็มที่ไม่ต้องการ และไอเท็มเป้าหมาย ไว้เหนือจอภาพที่เป็นผลลัพธ์ของการค้นหากฎความสัมพันธ์

ส่วนสุดท้ายของบทนี้ได้สรุปเวลาที่ใช้ในการประมวลผลแต่ละข้อคำถาม จำนวนกฎที่ได้เมื่อจำนวนเงื่อนไขบังคับเพิ่มมากขึ้น แสดงในรูปที่ 4.12



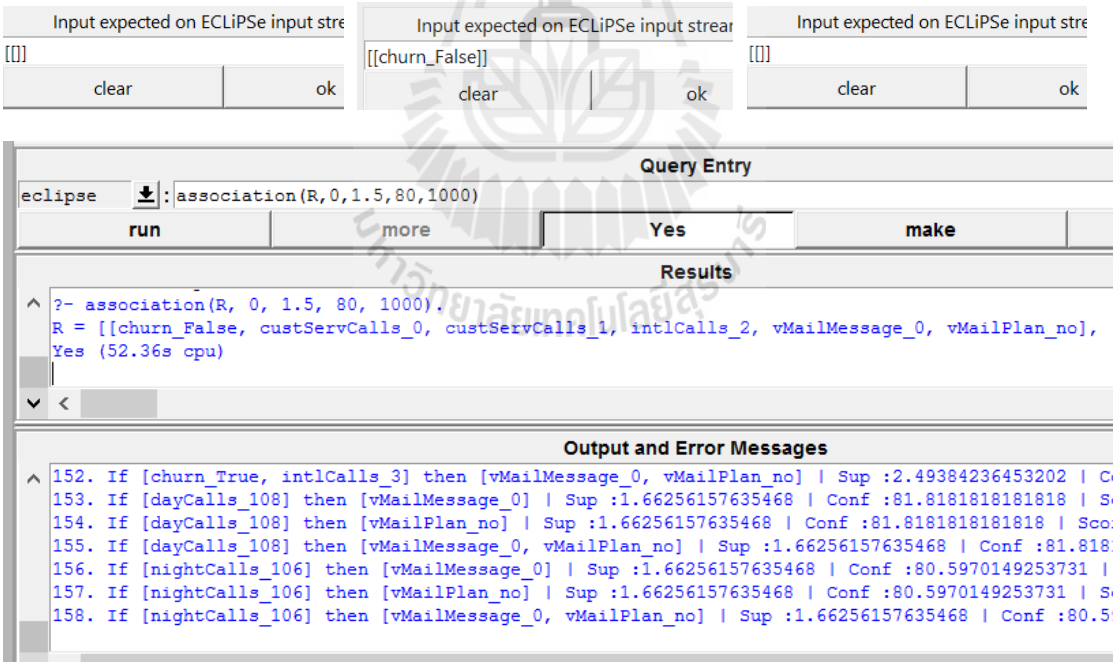
รูปที่ 4.5 การทดสอบข้อคำถามที่ 2 ด้วยโปรแกรม Top-K-patterns

(Query 2: Minsup=1.5%, Minconf=80%, must contain 'churn_False')



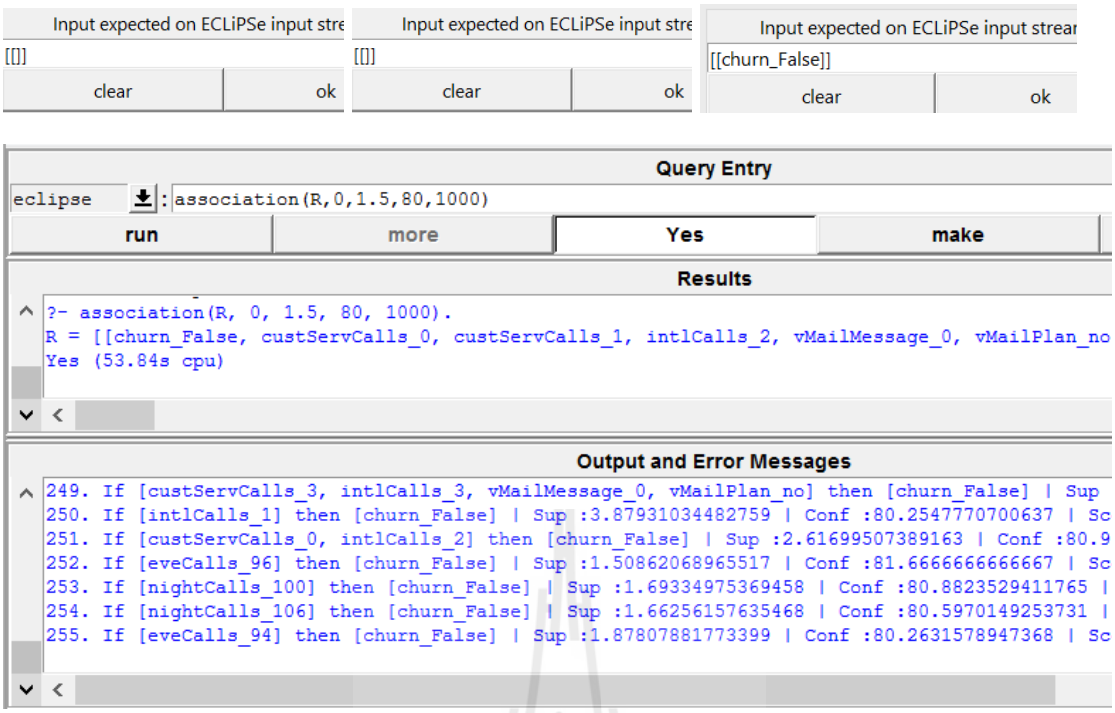
รูปที่ 4.6 การทดสอบข้อคำถามที่ 3 ด้วยโปรแกรม Top-K-patterns

(Query 3: Minsup=1.5%, Minconf=80%, must contain at least 3 items)



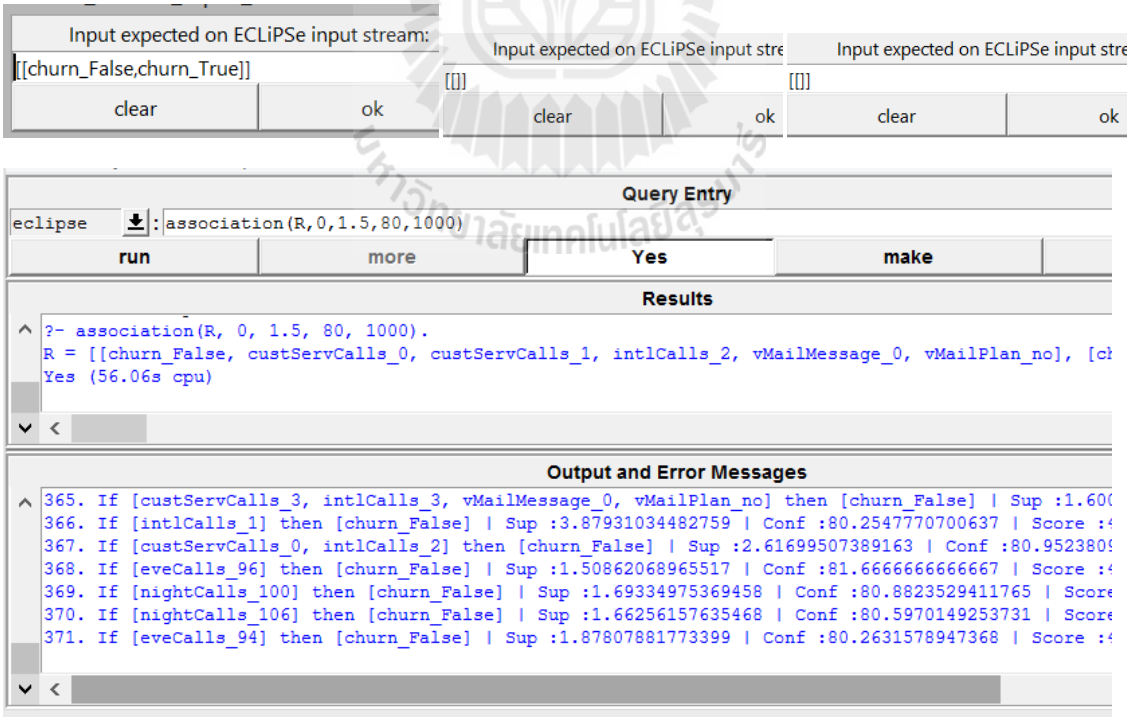
รูปที่ 4.7 การทดสอบข้อคำถามที่ 4 ด้วยโปรแกรม Top-K-patterns

(Query 4: Minsup=1.5%, Minconf=80%, must NOT contain ‘churn_False’)



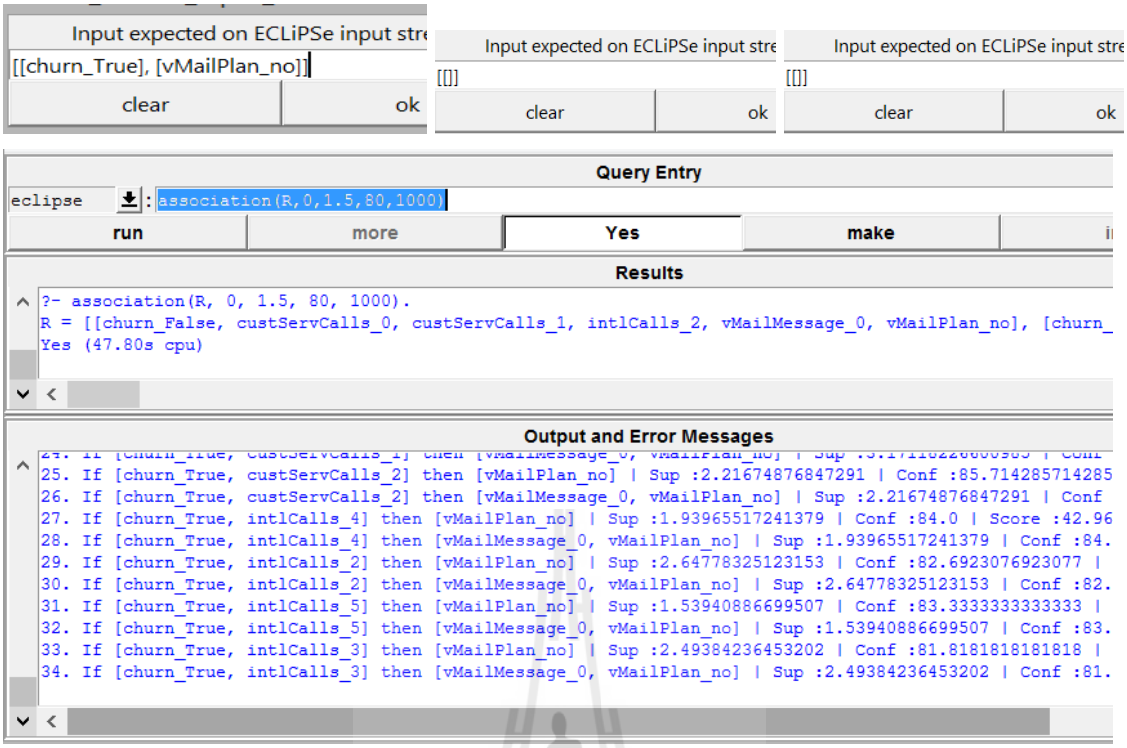
รูปที่ 4.8 การทดสอบข้อคำถามที่ 5 ด้วยโปรแกรม Top-K-patterns

(Query 5: Minsup=1.5%, Minconf=80%, at target of rule must contain ‘churn_False’)



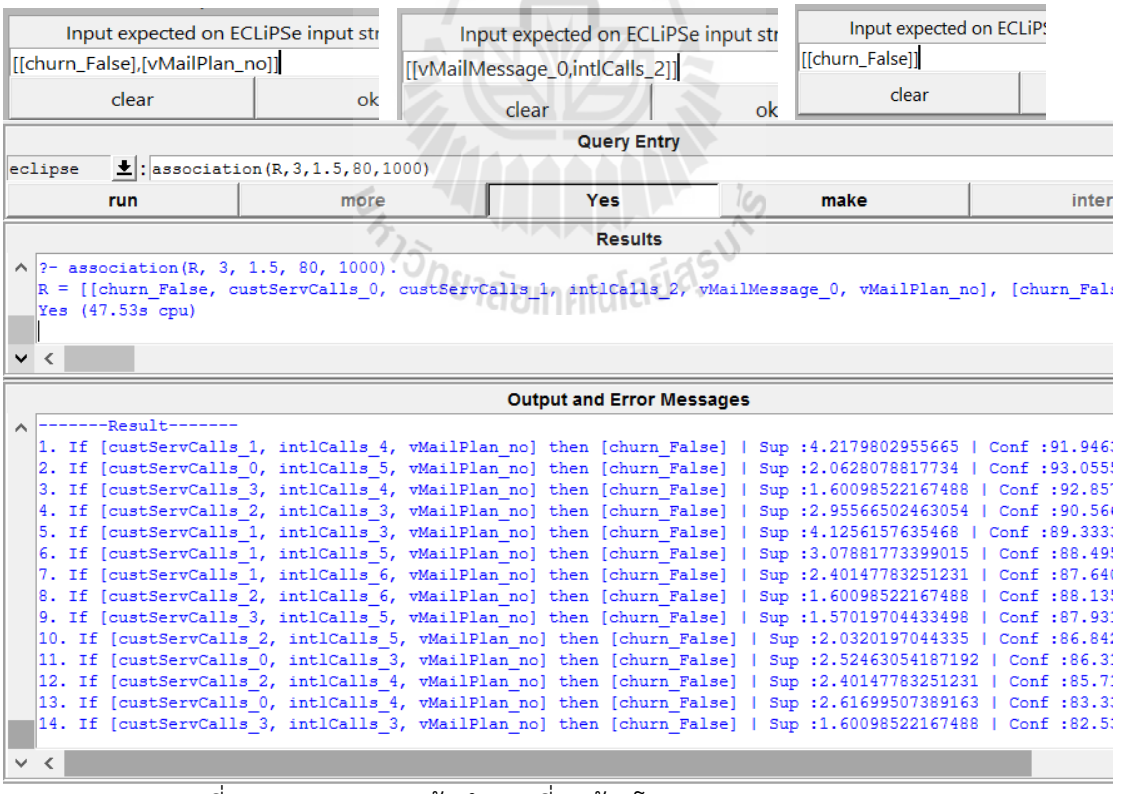
รูปที่ 4.9 การทดสอบข้อคำถามที่ 6 ด้วยโปรแกรม Top-K-patterns

(Query 6: Minsup=1.5%, Minconf=80%, must contain ‘churn_False’ OR ‘churn_True’)



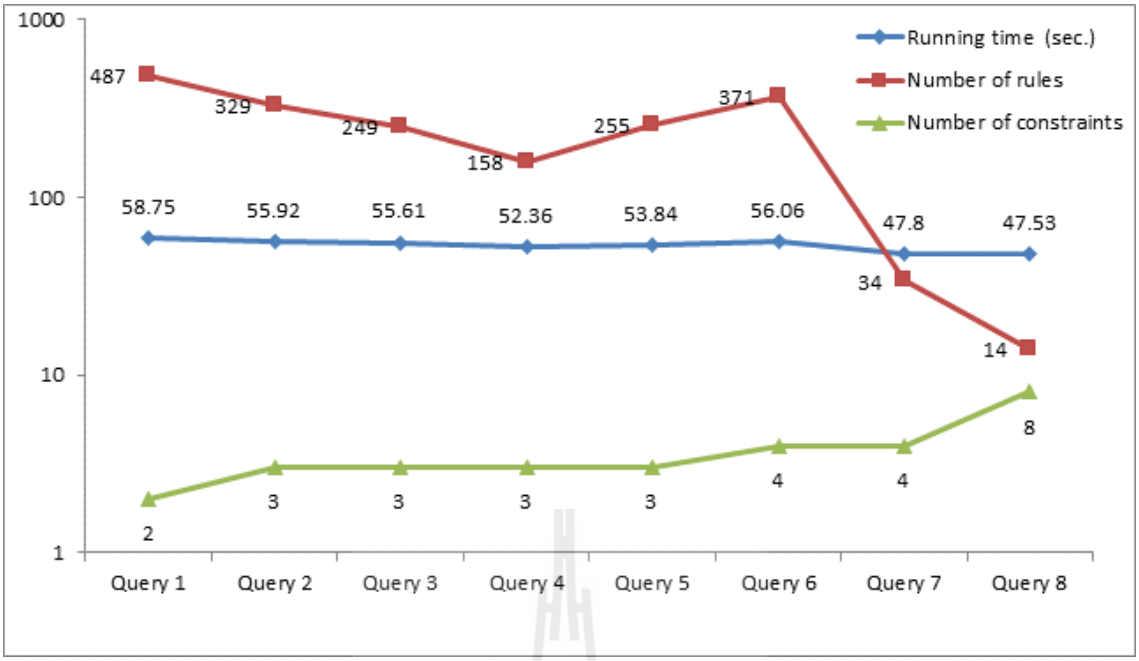
รูปที่ 4.10 การทดสอบข้อคำถามที่ 7 ด้วยโปรแกรม Top-K-patterns

(Query 7: Minsup=1.5%, Minconf=80%, must contain ‘churn_True’ AND ‘vMailPlan_no’)



รูปที่ 4.11 การทดสอบข้อคำถามที่ 8 ด้วยโปรแกรม Top-K-patterns

(Query 8: Minsup=1.5%, Minconf=80%, has at least 3 items, must contain both ‘churn_Flase’ and ‘vMailPlan_no’, must NOT contain ‘vMailMessage_0’ or ‘intlCalls_2’, target must be ‘churn_False’)



รูปที่ 4.12 การเปรียบเทียบเวลาที่ใช้ในการประมวลผลและจำนวนกฎที่ได้เมื่อเพิ่มเงื่อนไขบังคับ

การแสดงกราฟในรูปที่ 4.12 ใช้สเกลในแนวแกนตั้งเป็นลอการิทึมเนื่องจากข้อมูลเวลา จำนวนกฎ และจำนวนเงื่อนไขบังคับ มีขนาดที่แตกต่างกันมาก จากการเปรียบเทียบจำนวนเงื่อนไขบังคับกับจำนวนกฎความสัมพันธ์ที่ได้รับ (เส้นกราฟล่างสุดและบนสุด ในรูปที่ 4.12) จะเห็นได้ว่าเมื่อเพิ่มจำนวนเงื่อนไขจาก 2 เป็น 3 ในข้อคำถามที่ 2 และ 3 และจาก 4 เป็น 8 ในข้อคำถามที่ 7 และ 8 จะเห็นการลดลงของจำนวนกฎได้อย่างชัดเจน ส่วนในกรณีข้อคำถามที่ 3-6 จำนวนกฎที่ได้รับมีทั้งลดลงและเพิ่มขึ้นทั้งนี้ขึ้นอยู่กับรายละเอียดในเงื่อนไข ถ้าผู้ใช้ระบุชื่อและค่าของไอเท็มในเงื่อนไขต่างออกไปจากตัวอย่างนี้ ผลลัพธ์ที่ได้ย่อมต่างไปเช่นเดียวกัน

ข้อสังเกตที่เห็นได้ชัดเจนอีกประการหนึ่งคือในข้อคำถามที่ 6 ใช้เงื่อนไข OR ในขณะที่ข้อคำถามที่ 7 ใช้เงื่อนไข AND ในการระบุไอเท็มที่ต้องการให้ปรากฏในผลลัพธ์ ข้อคำถามที่ 6 ได้กฎความสัมพันธ์จำนวน 356 กฎ ส่วนข้อคำถามที่ 7 ได้กฎความสัมพันธ์เพียง 34 กฎ

ในการเปรียบเทียบเวลาที่ใช้ในการประมวลผล ข้อคำถามแรกใช้เวลาในการประมวลผลมากที่สุด เนื่องจากการใช้เงื่อนไขบังคับจำกัดเพียงเงื่อนไขค่าสนับสนุนและค่าความเชื่อมั่น (เหมือนโปรแกรมค้นหากฎความสัมพันธ์โดยทั่วไป) เมื่อเพิ่มเงื่อนไขบังคับลักษณะต่าง ๆ เพิ่มมากขึ้น เวลาที่ใช้ในการประมวลผลจะลดลง แต่สัดส่วนการลดลงของเวลาไม่มีนัยสำคัญมากนักในข้อมูลตัวอย่างชุดนี้ (ขนาดของข้อมูล 526 KB) แต่คาดว่าถ้าข้อมูลมีขนาดใหญ่และการระบุเงื่อนไขมีมากขึ้น เวลาที่ใช้จะแตกต่างกันอย่างมีนัยสำคัญ

บทที่ 5
บทสรุป

5.1 สรุปผลการวิจัย

การค้นหารูปแบบที่ปรากฏบ่อย (frequent pattern discovery) และแสดงผลลัพธ์ของรูปแบบที่ค้นพบในลักษณะของข้อความเชิงตรรกะ “ถ้า *เกิดเหตุการณ์ (หรือข้อมูล) X* แล้ว *จะเกิดเหตุการณ์ (หรือข้อมูล) Y ร่วมด้วย*” หรือ IF *X* THEN *Y* (อาจจะเขียนรูปแบบย่อเป็น *X => Y*) เรียกว่าการทำเหมืองข้อมูลชนิดการค้นหาความสัมพันธ์ (association mining) การวิเคราะห์ข้อมูลในลักษณะการค้นหาความสัมพันธ์นี้ เป็นกระบวนการอัตโนมัติที่ต้องใช้การอ่านข้อมูลจากฐานข้อมูลหลายรอบ ในแต่ละรอบโปรแกรมจะนับไอเท็ม (item, attribute, feature, or variable) ที่ปรากฏร่วมกันในแต่ละทรานแซคชัน (หรือเรคคอร์ด) เพื่อนับความถี่ของการปรากฏร่วมกันของแต่ละกลุ่มไอเท็ม กลุ่มเหล่านี้เรียกว่าเซตของไอเท็ม หรือ ไอเท็มเซต (itemset)

การทำเหมืองข้อมูลประเภทค้นหาความสัมพันธ์มุ่งเน้นที่การค้นหาเซตที่มีไอเท็มปรากฏบ่อย (frequent itemset) ขนาดตั้งแต่หนึ่งไอเท็ม ไปจนถึงหลายสิบหรือหลายร้อยไอเท็ม การอ่านฐานข้อมูลหลายรอบเพื่อนับจำนวนครั้งของการเกิดร่วมกันของกลุ่มไอเท็ม จึงเป็นงานที่ใช้เวลาและหน่วยความจำมาก ในกรณีที่ข้อมูลมีจำนวนไอเท็มและจำนวนทรานแซคชันสูงมาก เช่นข้อมูลภาพ ข้อมูลบันทึกการเข้าใช้เว็บ เครื่องคอมพิวเตอร์ส่วนบุคคลที่ใช้งานกันทั่วไปอาจจะมีประสิทธิภาพไม่เพียงพอที่จะประมวลผลได้สำเร็จ

การใช้เงื่อนไขบังคับ (constraint) เพื่อจำกัดขอบเขตการค้นหาไอเท็มปรากฏบ่อย เป็นแนวทางหลักที่นิยมใช้เพื่อเพิ่มความเร็วในการประมวลผล และลดเนื้อที่หน่วยความจำที่จำเป็นต้องใช้สำหรับบันทึกความถี่ของการเกิดร่วมกันของไอเท็มในเซตที่สนใจ แนวทางแก้ปัญหาแบบอื่นที่เป็นไปได้ เช่น ใช้การประมวลผลแบบขนานเพื่อเพิ่มความเร็วในการประมวลผล ใช้โครงสร้างข้อมูลต้นไม้เพื่อช่วยลดพื้นที่หน่วยความจำในการบันทึกผลลัพธ์ชั่วคราว แต่การใช้เงื่อนไขบังคับเป็นการลดขอบเขตของปริภูมิค้นหา (search space) จึงเป็นปัจจัยโดยตรงที่ช่วยลดการใช้เนื้อที่หน่วยความจำและลดเวลาการประมวลผล การใช้เงื่อนไขบังคับประกอบกับการค้นหาความสัมพันธ์ที่ปรากฏบ่อยจึงเป็นแนวทางหลักของงานวิจัยนี้

เงื่อนไขบังคับที่ใช้อยู่ในโปรแกรมค้นหาความสัมพันธ์ส่วนใหญ่จะเป็นเงื่อนไขค่าสนับสนุนขั้นต่ำ (minimum support) และค่าความเชื่อมั่นขั้นต่ำ (minimum confidence) ค่าสนับสนุนขั้นต่ำจะหมายถึงค่าความถี่ขั้นต่ำของไอเท็มเซตปรากฏบ่อยที่ผู้ใช้คาดว่าจะให้ผลลัพธ์ที่น่าสนใจ โดยค่าสนับสนุนของไอเท็มเซต X และ Y จะคำนวณได้จากจำนวนทรานแซคชันที่ปรากฏทั้งไอเท็ม X และ Y

หารด้วยจำนวนทรานแซคชันทั้งหมดในฐานข้อมูล (support(X∧Y) = number of transactions that contain (X∧Y) / number of all transactions) ไอเท็มเซตที่ปรากฏบ่อยไม่ถึงเกณฑ์ขั้นต่ำจะถูกละทิ้ง เนื่องจากผู้ใช้คาดว่าจะไม่ให้ผลลัพธ์ที่เป็นประโยชน์ต่อการนำไปใช้งาน และในขั้นตอนช่วงที่สองที่เป็นการสร้างกฎ “ถ้า *X* แล้ว *Y*” จากกลุ่มของไอเท็มที่ปรากฏบ่อยถึงเกณฑ์ขั้นต่ำ จะใช้เงื่อนไขบังคับที่เรียกว่า ค่าความเชื่อมั่นขั้นต่ำของกฎความสัมพันธ์ ซึ่งเป็นสัดส่วนของค่าสนับสนุนของไอเท็มทั้งเซตต่อค่าสนับสนุนของเฉพาะไอเท็มที่ปรากฏในข้อความส่วน “ถ้า” ของกฎความสัมพันธ์ นั่นคือ confidence(X=>Y) = support(X∧Y) / support(X) กฎความสัมพันธ์ใดที่มีค่าความเชื่อมั่นไม่ถึงเกณฑ์ขั้นต่ำที่ผู้ใช้ระบุ จะไม่ถูกแสดงเป็นผลลัพธ์

โปรแกรม Apriori (Agrawal and Srikant, 1994) เป็นโปรแกรมแรกที่ใช้ค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำเป็นเงื่อนไขบังคับ เพื่อเพิ่มความเร็วของการค้นหารูปแบบของไอเท็มเซตที่ปรากฏบ่อยและแสดงผลลัพธ์ในลักษณะกฎความสัมพันธ์ โปรแกรมนี้ได้รับความสำเร็จสูงมากในการประยุกต์ใช้กับข้อมูลในงานด้านต่าง ๆ แต่เนื่องจากข้อจำกัดของโปรแกรม Apriori ที่ผู้ใช้จะต้องทดลองประมวลผลข้อมูลหลายครั้ง แล้วสังเกตจากผลลัพธ์ที่ได้เพื่อปรับค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำให้เหมาะสมกับข้อมูลของตน นอกจากนี้กฎที่ได้จะปรากฏไอเท็มที่อาจจะมีความน่าสนใจกระจายไปในแต่ละกฎตามความถี่ที่ไอเท็มนั้น ๆ เกิดร่วมกับไอเท็มอื่น ๆ ในฐานข้อมูลทรานแซคชัน ทำให้เป็นปัญหาต่อผู้ใช้ในการค้นหากฎที่มีการเกิดขึ้นของไอเท็มที่น่าสนใจเพื่อนำไปใช้ประโยชน์ต่อไป

ปัญหาของข้อจำกัดดังกล่าวข้างต้นจึงเป็นที่มาของงานวิจัยนี้ ที่ผู้วิจัยมีวัตถุประสงค์หลักต้องการจะออกแบบโปรแกรมทำเหมืองข้อมูลประเภทการค้นหากฎความสัมพันธ์ ให้ผู้ใช้สามารถใช้งานโปรแกรมได้สะดวกขึ้น ผู้ใช้ที่ไม่ทราบรายละเอียดขั้นตอนการค้นหากฎความสัมพันธ์สามารถใช้คำสั่งที่ไม่ต้องระบุเงื่อนไขบังคับมาก โดยเรียกใช้คำสั่ง top_K_patterns(K) และระบุเพียงจำนวนกฎที่ต้องการให้แสดงในผลลัพธ์ โปรแกรมจะกำหนดค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำให้กับผู้ใช้ จากนั้นจะแสดงกฎความสัมพันธ์ที่มีค่าความเชื่อมั่นและค่าสนับสนุนสูงสุด เค อันดับแรกให้ผู้ใช้ทราบ

สำหรับผู้ใช้ที่มีความเชี่ยวชาญในการใช้งานโปรแกรมทำเหมืองข้อมูล ประเภทการค้นหากฎความสัมพันธ์มาก่อนแล้ว และเข้าใจวิธีการรวมถึงเทคนิคกำหนดค่าสนับสนุนและค่าความเชื่อมั่นขั้นต่ำ โปรแกรม Top-K-patterns ที่พัฒนาขึ้นนี้จะช่วยอำนวยความสะดวกให้กับผู้ใช้มากขึ้น โดยผู้ใช้สามารถระบุเงื่อนไขบังคับอื่น ๆ เพิ่มขึ้น ได้แก่

- ระบุจำนวนกฎที่ต้องการให้แสดงในผลลัพธ์
- ระบุจำนวนไอเท็มขั้นต่ำที่จะปรากฏในกฎความสัมพันธ์
- ระบุชื่อไอเท็มที่ต้องการให้ปรากฏในกฎความสัมพันธ์ (หมายถึง ไอเท็มที่ระบุปรากฏในส่วนใดของกฎก็ได้)

- ระบุชื่อไอเท็มที่ไม่ต้องการให้ปรากฏในกฎ (ใช้ในกรณีที่ผู้ใช้ไม่สนใจไอเท็มนั้น ถึงแม้จะเป็นไอเท็มที่ปรากฏบ่อยก็ตาม)
- ระบุรายการไอเท็มที่ต้องการให้ปรากฏ (หรือไม่ปรากฏ) ในกฎความสัมพันธ์ได้หลายรายการในลักษณะของการดำเนินการเชิงตรรกะ AND และ OR
- ระบุชื่อไอเท็มที่ต้องการให้ปรากฏในส่วนผลของกฎ (นั่นคือ ปรากฏในส่วนหลังข้อความ THEN ในกฎ IF-THEN)

การใช้เงื่อนไขบังคับระบุชื่อไอเท็มในส่วน THEN จะช่วยให้ผู้ใช้สามารถใช้งานโปรแกรมทำเหมืองข้อมูลแบบค้นหากฎความสัมพันธ์ (association mining) ในลักษณะเดียวกับการจำแนกประเภทข้อมูล (classification) ได้

5.2 ข้อจำกัดของโปรแกรมและข้อเสนอแนะ

โปรแกรม Top-K-patterns ที่พัฒนาขึ้นนี้ใช้วิธีการโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (constraint logic programming) ประสบผลสำเร็จตามวัตถุประสงค์ นั่นคือสามารถทำงานได้ถูกต้องเทียบเท่ากับโปรแกรม Apriori ที่นิยมใช้กันแพร่หลายในปัจจุบัน และโปรแกรม Top-K-patterns สามารถระบุเงื่อนไขบังคับได้หลากหลายมากกว่าโปรแกรม Apriori แต่การโต้ตอบกับผู้ใช้ยังต้องได้รับการปรับปรุงให้มีจอภาพโต้ตอบที่สวยงามและน่าใช้มากขึ้นกว่าในเวอร์ชันปัจจุบัน

การพัฒนาโปรแกรมด้วยซอฟต์แวร์ ECLiPSe ช่วยให้ซอร์สโค้ดของโปรแกรมสั้นและพัฒนาโปรแกรมได้เร็ว แต่นักพัฒนาโปรแกรมที่ไม่คุ้นเคยกับวิธีการโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับอาจจะทำความเข้าใจโปรแกรมได้ยาก ผู้วิจัยจึงมีแนวคิดที่จะปรับเปลี่ยนซอฟต์แวร์เป็น SWI Prolog ที่ใช้พื้นฐานการโปรแกรมเชิงตรรกะเป็นแนวทางหลัก และใช้ constraint handler เป็นส่วนเสริมในการประมวลผลเงื่อนไขบังคับ

แนวทางการพัฒนาในอนาคตที่สำคัญคือการปรับปรุงวิธีการประมวลผลของโปรแกรม Top-K-patterns ให้สามารถรองรับข้อมูลขนาดใหญ่มากได้ วิธีการหนึ่งที่เป็นไปได้คือใช้วิธีการแบ่งข้อมูลเป็นส่วนย่อยและประมวลผลในแบบเพิ่มพูน (incremental)

บรรณานุกรม

R. Agrawal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing*, 61:350-371, 2001.

R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)*, pp.207-216, Washington, DC, May 1993.

R. Agrawal and R. Srikant. Fast algorithm for mining association rules in large databases. In *Research Report RJ 9839*, IBM Almaden Research Center, San Jose, CA, June 1994.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pp.487-499, Santiago, Chile, Sept. 1994.

R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. *IEEE Trans. Knowledge and Data Engineering*, 8:962-969, 1996.

K.R. Apt and M. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, U.K., 2007.

B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Model and issues in data stream systems. In *Proc. ACM Symp. Principles of Database Systems (PODS'02)*, 2002.

R. Bayardo, R. Agrawal, D. Gunopulos. Constraint-based rule mining in large, dense database. In *Proc. of the Data Mining and Knowledge Discovery*, 2000.

S. Bistarelli and F. Bonchi. Soft constraint based pattern mining. *Data and Knowledge Engineering*, 62:118-137, 2007.

Y. Cai, G.Pape, J. Han, M. Welge, and L. Auvil. MAIDS: Mining alarming incidents from data streams. In *Proc. Int. Conf. on Management of Data*, June 2004.

J. Chang and W. Lee. A sliding window method for finding recently frequent itemsets over online data streams. *Journal of Information Science and Engineering*, July 2004.

M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, Jan. 2004.

D. W. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems*, pp.31-44, Miami Beach, Florida, Dec. 1996.

D. W. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 Int. Conf. Data Engineering (ICDE'96)*, pp.106-114, New Orleans, Louisiana, Feb. 1996.

Y. Chi, H. Wang, P. Yu, and R. R. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proc. IEEE Int. Conf. on Data Mining*, Nov. 2004.

K. Chuang, J. Huang, and M. Chen. Mining top-k frequent patterns in the presence of the memory constraint. *The VLDB Journal*, 17:1321-1344, 2008.

M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Resource-aware knowledge discovery in data streams. In *Proc. Int. Workshop on Knowledge Discovery in Data Streams*, Sept. 2004.

M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: A review. *ACM SIGMOD Record*, 34(2), June 2005.

A. Gallo, R. Esposito, R. Meo, M. Botta. Optimization of association rules extraction through exploitation of context dependent constraints. In *Proc. 9th Conference on Advances in Artificial Intelligence (AI*AI'05)*, 2005.

A. Ghoting and S. Parthasarathy. Facilitating interactive distributed data stream processing and mining. In *Proc. IEEE Int. Symposium on Parallel and Distributed Processing Systems*, Apr. 2004.

G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In Proc. *ICDM'03 Int. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, FL, Nov. 2003.

S. Guha, N. Koudas, and K. Shim. Data streams and histograms. In *Proc. ACM Symposium on Theory of Computing*, 2001.

M. Halatchev and L. Gruenwald. Estimating missing values in related sensor data streams. In *Proc. Int. Conf. on Management of Data*, Jan. 2005.

J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pp.420-431, Zurich, Switzerland, Sept. 1995.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pp.1-12, Dallas, TX, May 2000.

B. Jeudy, J. Boulicaut. Costraint-based discovery and inductive query: application to association rule mining. In *Proc. ESF Exploratory Workshop on Pattern Detection and Discovery*, 2002.

M. Jiang and L. Gruenwald. Research issues in data stream association mining. *ACM SIGMOD Record*, 35(1):14-19, 2006.

H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring. In *Proc. SIAM Int. Conf. on Data Mining*, 2004.

K. Kerdprasop, N. Kerdprasop, and P. Sattayatham. A Monte Carlo method to data stream analysis. *Enformatika Transactions on Engineering, Computing and Technology*, 14:240-245, Aug. 2006.

H.-F. Li, S.-Y. Lee, and M.-K. Shan. An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proc. Int. Workshop on Knowledge Discovery in Data Streams*, Sept. 2004.

C.-H. Lin, D.-Y. Chiu, Y.-H. Wu, and A. Chen. Mining frequent itemsets from data streams with a time-sensitive sliding window. In *Proc. SIAM Int. Conf. on Data Mining*, April 2005.

J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by opportunistic projection. In *Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02)*, pp.239-248, Edmonton, Canada, July 2002.

G. Mao, X. Wu, C. Liu, X. Zhu, G. Chen, Y. Sun, and X. Liu. Online mining of maximal frequent item sequences from data streams. *Technical Report CS-05-07*, University of Vermont, Computer Science, June 2005.

A. Niederlinski. *A Quick and Gentle Guide to Constraint Logic Programming via ECLiPSe*. Jacek Skalmierski Computer Studio, Poland, 2014.

J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'95)*, pp.175-186, San Jose, CA, May 1995.

J. S. Park, M. S. Chen, and P. S. Yu. Efficient parallel mining for association rules. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pp.31-36, Baltimore, Maryland, Nov. 1995.

J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery in Databases*, pp.350-354, 2000.

J. Pei, J. Han, and L. Lakshmanan. Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8:227-252, 2004.

J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pp.215-224, Heidelberg, Germany, April 2001.

A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pp.432-443, Zurich, Switzerland, Sept. 1995.

H. Simonis. *Constraint Logic Programming*. 2008. Online available at http://4c.ucc.ie/~hsimonis/ConstraintLogicProgramming.pdf

R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. Int. Conf. Knowledge Discovery and Data Mining*, pp.67-73, 1997.

W.-G. Teng, M.-S. Chen, and P. Yu. Resource-aware mining with variable granularities in data streams. In *Proc. SIAM Int. Conf. on Data Mining*, 2004.

H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pp.134-145, Bombay, India, Sept. 1996.

T. Trifonov, T. Georgieva, T. (2009). Application for discovering the constraint-based association rules in an archive for unique Bulgarian bells. *European Journal of Scientific*, 31(3), 2009.

J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):652-664, 2005.

J. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *Proc. Int. Conf. on Very Large Databases*, 2004.

M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithm for discovery of association rules. *Data Mining and Knowledge Discovery*, 1:343-374, 1997.

ภาคผนวก

ภาคผนวก ก

ผลผลิตของงานวิจัย: บทความวิจัยตีพิมพ์ในวารสารวิชาการ

1. N. Kerdprasop and K. Kerdprasop (2011). Frequent pattern discovery with constraint logic programming. *International Journal of Mathematical Models and Methods in Applied Sciences*, Vol.5, Issue 8, pp.1345-1353. (indexed in Scopus)
2. N. Kerdprasop, P. Kongchai, and K. Kerdprasop (2013). Constraint mining in business intelligence: A case study of customer churn prediction. *International Journal of Multimedia and Ubiquitous Engineering*, Vol.8, No.3, May, pp.11-20. (indexed in Scopus)
3. N. Kerdprasop, F. Koongaew, and K. Kerdprasop (2013). Building and querying a decision tree model with constraint logic programming. *International Journal of Software Engineering and Its Applications*, Vol.7, No.5, September, pp.269-282. (indexed in Scopus)
4. ไพชยนต์ คงไชย, นิตยา เกิดประสพ และ กิตติศักดิ์ เกิดประสพ (2015). การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (The discovery of association rules using constraint logic programming). *วิศวกรรมสารเกษมบัณฑิต*, ปีที่ 4, ฉบับที่ 1, มกราคม-มิถุนายน 2558, หน้า 1-15. (indexed in TCI)

Frequent Pattern Discovery with Constraint Logic Programming

Nittaya Kerdprasop and Kittisak Kerdprasop

***Abstract*—**Constraint logic programming is a declarative programming style combining the features of logic programming and constraint propagation to solve combinatorial and optimization problems such as resource allocation, scheduling, and routing. We consider the problem of mining frequent patterns within a setting of constraint logic programming approach. Frequent patterns are patterns such as sets of features or items in transactions that appear frequently. Such patterns can reveal associations, correlations, and many other interesting relationships hidden in a dataset. Constraints can play an important role in improving the performance of mining algorithms. The problem of constraint-based pattern mining can be formulated as the discovery of all patterns in a given dataset that satisfy the specified constraints. We present implementation of problem modeling and solving with respect to pattern mining in knowledge discovery in databases.

***Keywords*—** Frequent pattern discovery, Itemset mining, Logic-based programming, Constraint programming, ECLiPSe constraint system.

I. Introduction

THE main goal of data mining is to extract hidden knowledge from data [8]. Knowledge is a valuable asset to most organizations as a substantial source to enhance understanding of data relationships and support better decisions to increase organizational competency. Automatic knowledge acquisition can be achieved through the availability of the knowledge induction component. The induced knowledge can facilitate various knowledge-related activities ranging from expert decision support, data exploration and explanation, estimation of future trends, and prediction of future outcomes based on present data. The methodology of knowledge induction is known as knowledge discovery in databases, or data mining.

Data mining methods are broadly defined depending on the specific research objective and involve different classes of mining tasks including regression, classification, clustering, identifying meaningful associations between data attributes. The later mining task refers to association mining, or market basket analysis [9] in the retail business domain, which is the main focus of our research**.**

Association mining is a popular method for discovering relations between features or variables in large databases [11], [12], [14], [19] and then presenting the discovered results as a set of if-then rules, such as {milk, bread} => {butter} to indicate that if a customer buys milk and bread, he or she is more like to buy butter as well. Association rule generation process is composed of two major phases: frequent itemset mining and rule generation. Frequent itemset mining is to find all items or features that are frequently occurred together [13], [21]. It is an import phase of association mining because it is a difficult task to search all possible itemsets. We thus pay attention to the design and implementation of frequent itemset discovery by applying the methodology of constraint logic programming. Our implementation is based on the ECLiPSe constraint system.

The organization of this paper is as follows. The problem of frequent pattern discovery is defined in Section 2. Then the logic-based and constraint programming implementation of frequent pattern discovery is explained and demonstrated in Section 3. Experimentation and results are presented in Section 4. Finally, Section 5 concludes the paper with discussion of our future research direction.

II. Concepts and Techniques of Frequent Pattern Discovery

Frequent patterns are patterns such as sets of features or items that appear in data frequently. Frequent pattern mining focuses on the discovery of relationships or correlations between items in a dataset. A set of market basket transactions [1], [2] is a common dataset used in frequent pattern analysis. A dataset is typically in a table format. Each row is a transaction, identified by a transaction identifier or a *TID*. A transaction contains a set of items bought by a customer. A set of transactions might be organized in either an enumerated (dense), or a sparse binary vector format [6]. In either format a dataset can be processed horizontally or vertically. Figure 1 illustrates the data organization formats of a simple market basket dataset.

Manuscript received June 5, 2011: Revised version received August 2, 2011. This work was supported by grants from the National Research Council of Thailand (NRCT) and Suranaree University of Technology through the funding of Data Engineering Research Unit.

N. Kerdprasop is an associate professor and the director of Data Engineering Research Unit, School of Computer Engineering, Suranaree University of Technology, 111 University Avenue, Muang District, Nakhon Ratchasima 30000, Thailand (phone: +66-44-224-432; fax: +66-44-224-602; e-mail: nittaya@sut.ac.th).

K. Kerdprasop is with the School of Computer Engineering and Data Engineering Research Unit, Suranaree University of Technology, 111 University Avenue, Muang District, Nakhon Ratchasima 30000, Thailand (e-mail: KittisakThailand@gmail.com).

TID	Items
1	{Cereal, Milk}
2	{Beer, Cereal, Diaper, Egg}
3	{Beer, Diaper, Milk}
4	{Beer, Cereal, Diaper, Milk}
5	{Diaper, Milk}

(a) Horizontally enumerated format

TID	Item IDs				
	B	C	D	E	M
	2	1	2	2	1
	3	2	3		3
	4	4	4		4
			5		5

(b) Vertically enumerated format

TID	Item IDs				
	B	C	D	E	M
1	0	1	0	0	1
2	1	1	1	1	0
3	1	0	1	0	1
4	1	1	1	0	1
5	0	0	1	0	1

(c) Horizontal binary vector

TID	Item IDs				
	B	C	D	E	M
1	0	1	0	0	1
2	1	1	1	1	0
3	1	0	1	0	1
4	1	1	1	0	1
5	0	0	1	0	1

(d) Vertical binary vector

Fig. 1 horizontal and vertical organization schemes of transaction database

In a horizontally enumerated data organization (Figure 1(a)), each transaction contains only items positively associated with a customer purchase. It is a simplistic representation of market basket data because it ignores other information such as the quantity of purchased items or the profit of item sold. A horizontally enumerated format is sometimes referred to as a *TidLists* dataset organization.

In a vertical organization of items bought enumeration (Figure 1(b)), each column stores an ordered list of *TIDs* of the transactions that contain an item. This format of a dataset occupies that same space as the horizontally enumerated format.

Figures 1(c) and 1(d) represent a binary vector format. A value in each vector cell is 1 if the item is present in a transaction and 0 otherwise. A binary vector format is referred to as a *TidSets* dataset organization.

Recent attention has been given to the influence of data organization on the performance of the process of frequent pattern discovery. A vertical vector organization has been proven an efficient layout for the problem of frequent pattern discovery, but it suffers from the memory usage. In this paper, we study the performance of frequent pattern discovery based on the horizontal item organization.

In frequent pattern mining, we are interested in analyzing connections among items. A collection of zero or more items is called an itemset. For example, the first transaction in Figure 1 contains the itemset {Cereal, Milk}. Since this set contains two items, it is called a 2-itemset. An itemset can be an empty set, a 1-itemset, a 2-itemset, and so on. Figure 2 shows all combinations of distinct itemsets from the set of items {B, C, D, E, M}, where B = Beer, C = Cereal, D = Diaper, E = Egg, and M = Milk.

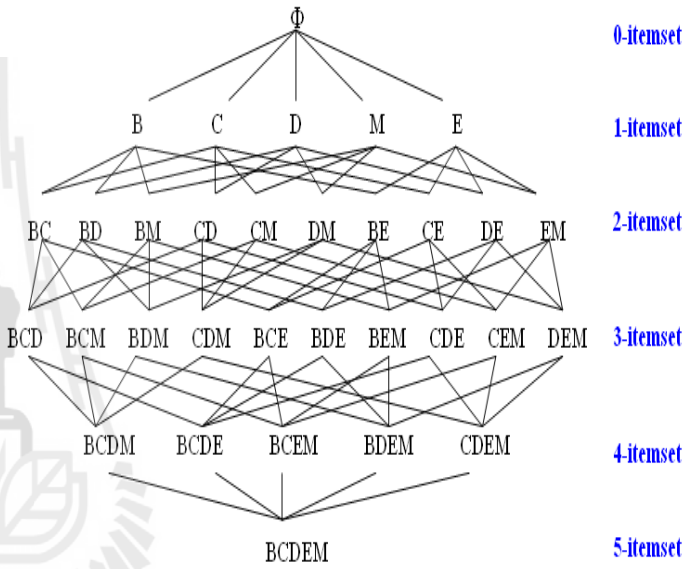


Fig. 2 a lattice of possible frequent patterns from the five distinct items

The discovery of interesting relationships hidden in large datasets is the objective of frequent pattern mining. The uncovered relationships can be represented in the form of association rules. An association rule is an inference of the form $X \rightarrow Y$, where X and Y are non-empty disjoint itemsets. To form association rules, we consider only valid itemsets. An itemset is valid if it really occurs in a transaction. For instance, from a dataset shown in Figure 1 an itemset {Egg, Milk} is invalid because none of the customers buy both eggs and milk.

The identification of all valid itemsets is computational expensive. It can be seen from Figure 2 that a dataset of I items has 2^I distinct itemsets. To reduce the search space, the measurements of *support* and *confidence* are used to constrain the mining process.

The constraint *support* forces the mining process to discover only relationships that occur frequently, while *confidence* constrains the reliability of the inference made by a rule.

The support count for an itemset *Z*, denoted as σ(*Z*), is the number of transactions that contain a particular itemset *Z*. As an example, consider a dataset in Figure 1, there are three transactions (*TID* 2, 3, 4) that contain the item Beer, thus σ(Beer) = 3. Given the definition of support count, the metrics support and confidence of the association rule $X \rightarrow Y$ can be defined as follows [9].

$$Support,\ s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N},$$

where N is the number of all transactions.

$$Confidence,\ c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}.$$

Given a dataset as shown in Figure 1, an example of association rule is the statement that "customers who buy beer also buy diaper, with 60% supporting transactions and 100% confidence." An itemset is called a *frequent itemset* if its support is greater than or equal user-specified support threshold (called *minSup*).

An association rule generated from frequent itemset with the confidence greater than or equal a confidence threshold (called *minConf*) is considered a valid association rule. With the pre-specified *minSup* and *minConf* metrics, the problem of association rule discovery can be stated as follows: Given a set of transactions, find all the rules having support ≥ *minSup* and confidence ≥ *minConf*. This problem can be decomposed into two subtasks:

(1) Frequent itemset generation: find all itemsets that satisfy the *minSup* threshold.

(2) Rule generation: generate from frequent itemsets all high confidence rules.

It is the *minSup* constraint that helps reducing the computational complexity of frequent itemset generation. Suppose we specify *minSup* = 2/5 = 40% on a set of transactions shown in Figure 1; the item {Egg} is infrequent. As a result, all supersets of {Egg} are also infrequent. All infrequent itemsets can then be pruned to reduce the search space (see Figure 3).

This pruning strategy is called an anti-monotone property and has been applied as a basis for searching frequent patterns in the well known algorithm Apriori [1], [2]. The detail of this algorithm is given in Figure 4.

The Apriori-like algorithms find all frequent itemsets by generating supersets of previously found frequent itemsets. This generate-and-test method is computational expensive. Han et al [10] proposed a different divide-and-conquer approach based on the prefix-tree structure that consumes less memory space. Toivonen [20] employed sampling techniques to deal with frequent pattern mining from large databases. Zaki et al [22] tackled the problem by means of parallel computation.

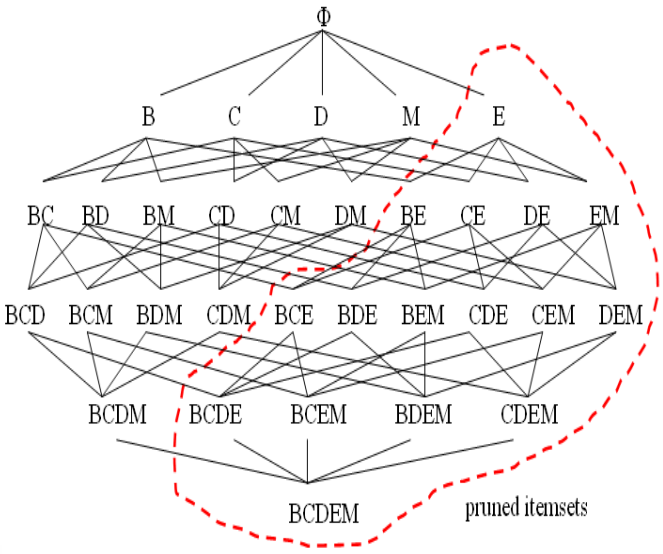


Fig. 3 a reduced search space after pruning an infrequent item E

Algorithm *Frequent pattern discovery*
Input: Transaction database, DB, of itemset I
Minimum support threshold, minSup
Output: Sets of frequent patterns of length 1 to k,
P1, .., Pk

1. P_1 = {x |x is an item in *I* and *s*(*P*) ≥ *minSup* }
// 1-item pattern
2. For (k=1; $P_k \neq \varnothing$; k++) do
2.1 C_{k+1} = *Generate_candidate*(P_k)
2.2 For each $T_i \in$ DB do
2.2.1 Increment the count *c* of all candidates in C_{k+1} that are contained in T_i
2.3 P_k = {c |c ∈ C_k and c.count ≥ *minS* }
3. Return $\cup_k P_k$ // return all sets of frequent patterns

Algorithm *Generate_candidate*

Input: Pattern at current level, P_{i-1}
Output: Pattern in larger level, C_i

1. $C_i = \varnothing$ // initialize candidate frequent pattern set
// as an empty set
2. For each pattern *J* in P_{i-1} do
2.1 For each pattern *K* in P_{i-1} and $K \neq J$ do
2.1.1 If i-2 of the elements in *J* and *K* are equal then
2.1.2 If all subsets of {$K \cup J$ } are in P_{i-1} then
2.1.3 $C_i = C_i \cup \{K \cup J\}$
3. Return C_i

Fig. 4 a pseudo code of Apriori algorithm [1], [2]

Some researchers [4], [7], [16], [17], [18] consider the issue of search space reduction through the concept of constraints. Our research is in the same direction as De Raedt et al [7]. We consider the problem of mining frequent patterns within a setting of constraint logic programming using the ECLiPSe constraint system [3].

Constraints can play an important role in improving the performance of mining algorithm. The problem of constraint-based pattern mining can therefore be formulated as the discovery of all patterns in a given dataset that satisfy the specified constraints. In the next section, we present work in progress of problem modeling and solving with respect to the frequent pattern discovery in a transaction database.

III. Logic-based and Constraint-based Programming Paradigms

The problem of frequent pattern discovery is to determine how often a candidate pattern occurs. A pattern is a set of items co-occurrence across a database. Given a candidate pattern, the task of pattern matching is then applied to search for its frequency looking for the patterns that are frequent enough. The outcome of this search is frequent patterns that suggest strong co-occurrence relations between items in the dataset. We suggest that such pattern oriented task can be efficiently implemented with logic-based languages.

A. Logic Program

In logic programming, a statement is called a clause, which is a disjunction of literals (atomic symbols or their negations) such as $p \vee q$ and $\neg p \vee r$. A statement is in clausal form if it is a conjunction of clauses such as $(p \vee q) \wedge (\neg p \vee r)$. Logic programming is a subset of first order logic in which clauses are restricted to Horn clauses.

A Horn clause, named after the logician Alfred Horn [15], is a clause that contains at most one positive literal such as $\neg p \vee \neg q \vee r$. Horn clauses are widely used in logic programming because their satisfiability property can be solved by resolution algorithm (an inference method for checking whether the formula can be evaluated to true).

A Horn clause with no positive literal, such as $\neg p \vee \neg q$, which is equivalent to $\neg(p \wedge q)$, is called *query* in Prolog and can be interpreted as '*:- p, q'* in which its value (true/false) to be proven by resolution method. A clause that contains exactly one positive literal such as *r* is called a *fact* representing a true statement, written in clausal form as '*r :-'* in which the condition part is empty and that means *r* is unconditionally true. Therefore, facts are used to represent data.

A Horn clause that contains one positive literal and one or more negative literals such as $\neg p \vee \neg q \vee r$ is called a *definite clause* and such clause can equivalently written as $(p \wedge q) \rightarrow r$ which in turn can be represented as a Prolog *rule* as *r :- p, q.* The symbol '*:-'* is intended to mean '$\leftarrow$', which is implication in first-order logic (it stands for 'if'), and the symbol ',' represents the operator $\wedge$ (or 'AND').

In Prolog, rules are used to define procedures and a Prolog program is normally composed of facts and rules. Running a Prolog program is nothing more than posing queries to obtain true/false answers. The advantages of using logic programming are the flexible form of query posing and the additional information regarding variable instantiation obtained from the Prolog system once the query is evaluated to be true.

The symbols *p*, *q*, *r* are called *predicates* in first-order logic programming and they can be quantified over variables such as *r(X) :- p(X,Y), q(Y).* This clause has the same meaning as $\forall X\,(p(X,Y) \wedge q(Y) \rightarrow r(X))$.The scope of variables is within a clause (delimit the end of clause with a period). Horn clauses are thus the fundamental concept of logic programming.

The search for patterns of interest can be efficiently programmed using the logic-based language such as Prolog. In Prolog, the feature of pattern matching can be defined through the use of arguments. For example, the following program [5] demonstrates the length function (in Prolog it is called predicate instead of function) to count the number of elements in a list. Last argument is normally a place holder for an output.

Variables in Prolog start with an uppercase letter such as Xs, L, X. The first statement of the length predicate declares the fact that the length of an empty list is zero. The last statement of length predicate can be read as "length of the list (X|Xs) is L is length of the list (Xs) is M and L is M+1."

```
length([ ], 0).  -- pattern 1: length of an empty list is 0

length( (X|Xs), L) :-  length( Xs, M), L is M+1.
    -- pattern 2: length of a list whose first
    -- element is X and remainder is Xs is 1+  length of Xs
```

In declarative language such as Prolog, programs are sets of definitions and recursion is the main control structure of the program computation. In imperative or procedural languages, such as C and Java, programs are sequences of instructions and loops are the main control structure. A logic programming language like Prolog is a declarative language in which programs are sets of *predicate* definitions.

Predicates are true or false when applied to an object or set of objects. A predicate definition has a dual meaning: (1) it describes what is the case, and (2) it describes the way to compute something.

Declarative languages are mathematically sound. It is easy to prove that a declarative program meets its specification, which is an important requirement in software industry. Declarative style makes a program better engineered, that is, easier to debug, easier to maintain and modify, and easier for other programmers to understand.

The following is the coding of frequent pattern mining program in Prolog language with a simple transaction database of five items as appeared in Figure 1(a).

```
%% FrequentPattern.pl
%       call ?- r1.
%            ?- r2(X).
%            ?- clear.
tid(5).              % all transactions=5
s(3).                % support=3 or 0.6
n([b,c,d,e,m]).      % all items

item([c, m]).
item([b, c, d, e]).
item([b, d, m]).
item([b, c, d, m]).
item([d, m]).

r1 :- n(X), cL1(X).
r2(X) :- cC2(X).
r3(X) :- cC3(X).

clear :- retractall(l1(_)),
         retractall(c1(_)),
         retractall(c2(_)) ,
         retractall(l2(_)),
         retractall(c3(_)),
         retractall(l3(_)).
cL1([]).     % Create L1
cL1([H|T]) :-
    findall(X, f([H], X), L),
    length(L, Len),
    Len >= 2 , ! ,
    write( ok-head-H-len=Len),
    nl,
    cL1(T),
    assert(l1(([H], Len)))
    ;
    cL1(T).

    % Create C2, L2
cC2(X) :-
    l1((X,_)),
    l1((X2,_)),
    X \==X2,
    write(X-X2),
    union(X, X2, Res),
    assert(c2((Res))) ,
    retract(l1((X,_))) ,
    nl.

crC2(L) :- findall(X, c2(X), L).

cL2([]).
cL2([H|T]) :-
    findall(X, f(H, X), L),
    length(L, Len),
    Len >= 2 ,!,
    write( ok-head-H-len=Len),
    nl,
    cL2(T),
    assert(l2((H,Len)))
    ;
    cL2(T).

cC3(X) :-
     l2((X,_)),
     l2((X2,_)),
     X \==X2,
     write(X-X2),
     union(X, X2, Res),
     assert(c3((Res))) ,
     retract(l2((X,_))) ,
     nl.

crC3(L) :- findall(X, c3(X), L).

cL3([]).
cL3([H|T]) :-
     findall(X, f(H, X), L),
     length(L, Len),
     Len >= 2 , !,
     write( ok-head-H-len=Len),
     nl ,
     cL3(T),
     assert(l3((H, Len)))
     ;
     cL3(T).
f(H, X) :- item(X),
           subset(H, X).
```

The Prolog implementation of frequent itemset mining uses the findall, assert, and retract predicates, which are higher order. *Higher-order predicate* is a predicate in a clause that can quantify over other predicate symbols [5]. As an example, besides the rule *r(X):- p(X,Y), q(Y)*, if we are also given the following five Horn clauses (or facts): *p(1, 2). p(1, 3). p(5, 4). q(2). q(4).*

By asking the query: *?- r(X)*, we will get the response as '*true*' and also the first instantiation information as *X=1*. If we want to know all instantiations that make *r(X)* to be true, we may ask the query: *?- findall(X, r(X), Answer).*We will get the response: *Answer = [1,5]*, which is a set of all answers obtained from the predicate *r(X)* according to the given facts. The predicate symbol *findall* quantifies over the variables *X*, *Answer*, and the predicate *r*. The predicate *findall* is thus called a higher-order predicate.

B. Constraint Logic Program

Constraint logic programming is a declarative programming style that combines the features of logic programming and constraint propagation to solve combinatorial and optimization problems. A constraint logic program is an extension of logic program by including constraints in the body of the clauses. Common structure of a constraint program is consisted of the part to define variables and constraints on variables and the part to search for a valid value on each variable. This is the style of constraint-and-search. The structure of constraint logic program is as follows:

```
solve(Variables) :-
        setup_constraints(Variables),
        search(Variables).
```

Figure 5 demonstrates the different between logic program and constraint program on the same problem of map coloring [3]. The problem is given four colors and four regions, the program has to provide coloring scheme such that two consecutive regions have different colors.

```
% Map coloring problem
%    Prolog style: Generate-and-test
color(red).
color(green).
color(blue).
color(yellow).
color_LP([A,B,C,D]) :-
                    color(A),
                    color(B),
                    color(C),
                    color(D),
                    A \= B,    A \= C,   A \= D,
                    B \= C,    B \= D,
                    C \= D.
```

```
% Map coloring problem
%   CLP style: Constrain-and-search
:- lib(fd).
color_CLP([A,B,C,D]) :-
                [A,B,C,D]::[red, green, blue, yellow],
                alldifferent([A,B,C,D]),
                labeling([A,B,C,D]).
```

Fig. 5 Logic programming versus constraint logic programming

The following implementation is the coding of frequent pattern mining in constraint logic programming using the ECLiPSe system (http://www.eclipseclp.org). The data set is transactional database containing itemset I as appeared in Figure 1(a). The constraint problem can be formulated as: given a transactional database D and a minimum support threshold σ, find all frequent itemsets FS such that FS = { $X \subseteq I$ | support(X) ≥ σ }. Screenshots of ECLiPSe system when a minimum support has been set to be 0.3 and 0.6 are shown in Figure 6.

```
% Frequent pattern discovery with constraint
:-lib(listut).
:-dynamic(c/2).
item([[b], [c], [d], [e], [m]]).
t([c, m]).
t([b, c, d, e]).
t([b, d, m]).
t([b, c, d, m]).
t([d, m]).
solve(Sigma) :-      %  Sigma=MinSupport
   retract_all(c(_,_)),
   findall(X,t(X),AllTrans),
   length(AllTrans,NoTrans),
   MinSup is Sigma*NoTrans,
   item(IT),
                    % scan in all transactions
   (foreach(Y,IT), param(MinSup)
     do findall(Y,(t(T), subset(Y,T)), Res),
        length(Res, Len),
        (Len >= MinSup -> writeln(Y-Len),
                          assert(c(1,Y)) ;  true) ),
   (for(K,1,5), param(MinSup)
     do (findall(Y, c(K,Y), RR),
         KK is K+1,
         findall(R1, (subset(S, RR),
                      myunion(S, R1),
                      length(R1,Len),
                      Len = KK),  Res11),
         maplist(flatten, Res11, Res12),
         remove_dups(Res12, Res1),
         (Res1=[] -> fail ; true), % break when [ ]
         (foreach(Y1, Res1), param(KK),
                      param(MinSup)
           do findall(Y1, (t(T), subset(Y1, T)), Res2),
              length(Res2,Len1) ,
              (Len1 >= MinSup ->
                       writeln(Y1-Len1),
                       assert(c(KK,Y1)) ; true) )
         ) ).     % end solve
myunion([X,Y], Z) :- union(X,Y,Z).
```

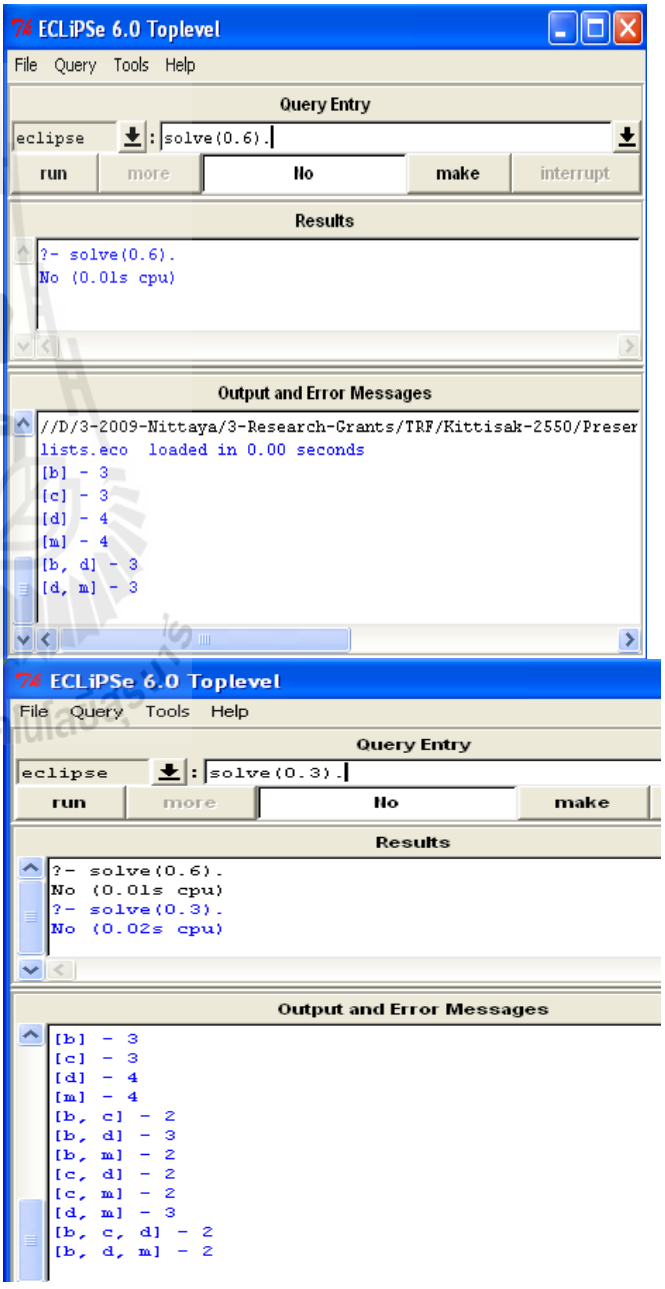


Fig. 6 Screenshots of ECLiPSe constraint system with 0.6 (top screen) and 0.3 (bottom) minimum support values, respectively

IV. Experimentation

We comparatively study the performance of our implementations of frequent pattern discovery using logic programming (SWI Prolog) and constraint logic programming (ECLiPSe constraint system). All experimentations have been performed on a 796 MHz AMD Athlon notebook with 512 MB RAM and 40 GB HD. We select four datasets from the standard UCI Machine Learning Repository (http://www.ics.uci.edu/~mlearn/MLRepository.html) to test the speed and memory usage of the programs. The details of selected datasets are summarized in Table 1.

TABLE I
DATASET CHARACTERISTICS

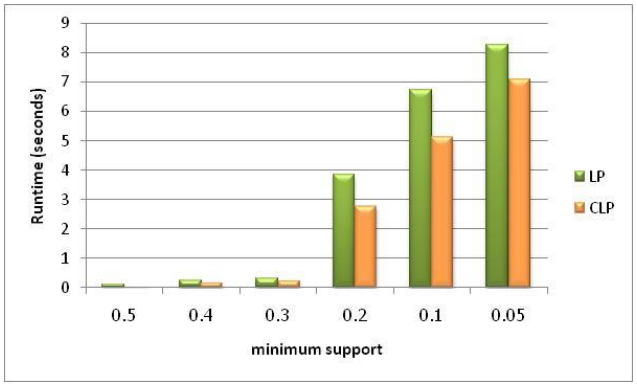
Dataset	File size	# Transactions	# Items
Vote	13.2 KB	300	17
Chess	237 KB	2,130	37
DNA	252 KB	2,000	61
Mushroom	916 KB	5,416	23

The frequent pattern discovery programs have been tested on each dataset with various *minSup* values, ranging from 0.005 to 0.5. Performance comparisons of logic programming (LP) and constraint logic programming (CLP) in terms of speed and memory usage on four datasets are shown in Figures 7 and 8, respectively. It can be noticed from the experimental results that on both speed and memory usage comparison CLP performs better than LP. However, the degree of difference is insignificant.

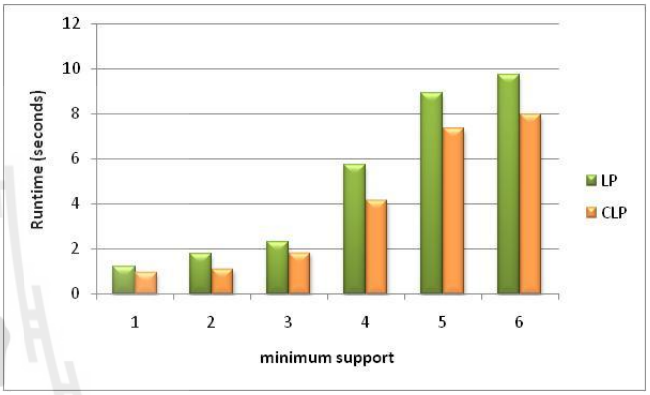
V. Conclusion

Frequent pattern discovery is one major problem in the areas of data mining and business intelligence. The problem concerns finding frequent patterns hidden in a large database. Finding such frequent patterns has become an important task because it reveals associations, correlations, and many other interesting relations among items in the databases. We suggest that the problem of frequent pattern discovery can be efficiently and concisely implemented with high-level declarative language such as Prolog. Coding in declarative style takes less effort because pattern matching is a fundamental feature supported by most logic-based languages. The implementation of Apriori algorithm using Prolog confirms our hypothesis about conciseness of the program.

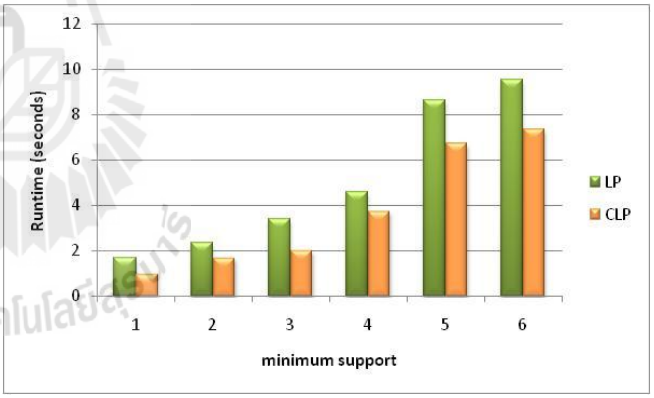
We also extend our study by implementing the algorithm with constraint logic programming paradigm using the ECLiPSe constraint system. The performance studies also support our intuition on the issue of efficiency because our constraint-based implementation is slightly more efficient than the conventional logic programming implementation in terms of speed and memory usage.



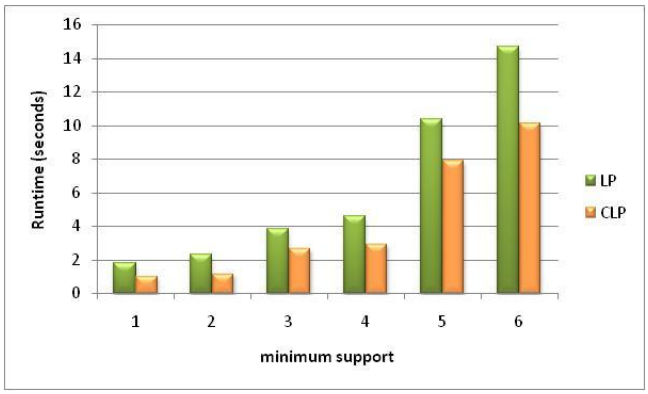
(a) Vote data



(b) Chess data

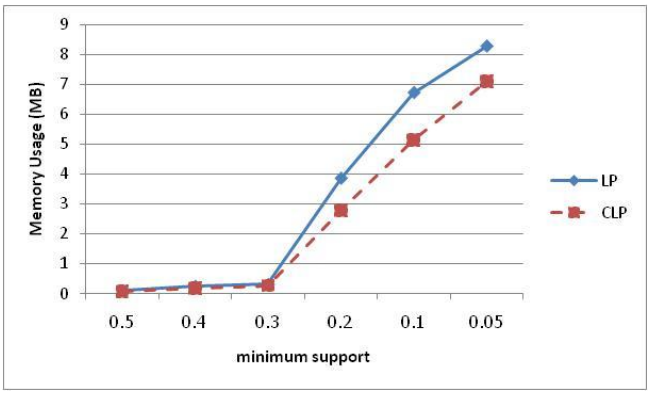


(c) DNA data

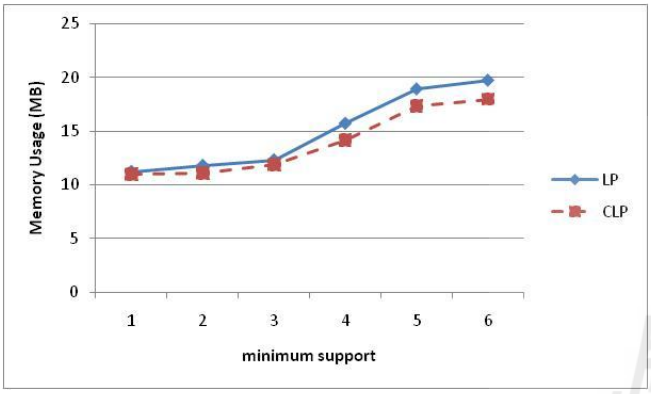


(d) Mushroom data

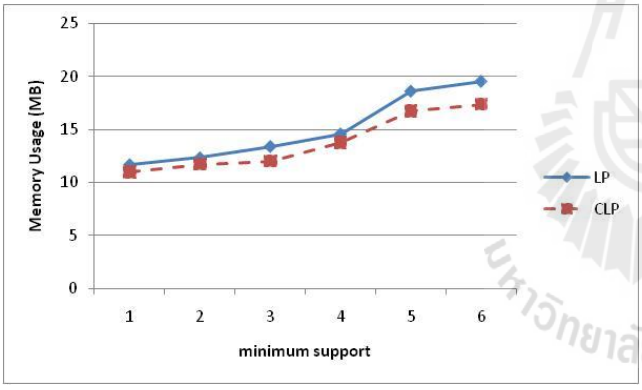
Fig. 7 speed comparison of logic program versus constraint logic program



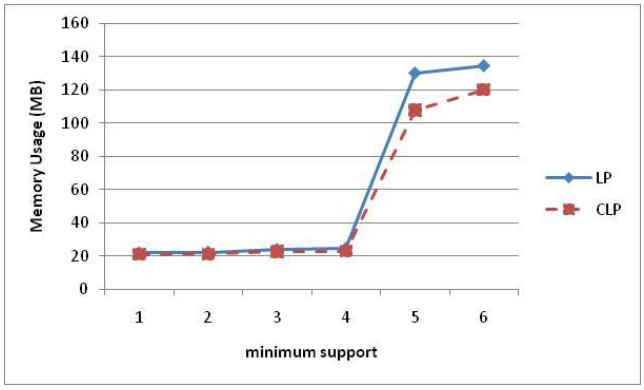
(a) Vote data



(b) Chess data



(c) DNA data



(d) Mushroom data

Fig. 8 Memory usage comparison of logic program versus constraint logic program

This preliminary study supports our belief regarding constraint-based declarative programming paradigm towards frequent pattern discovery. We focus our future research on the design of constraint formulating and processing to optimize the speed and storage requirement. We also consider the extension of the algorithm in the course of concurrency to improve its performance.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD*, 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. VLDB*, 1994, pp. 487-499.

[3] K. R. Apt and M. Wallace, *Constraint Logic Programming using ECLiPSe*, Cambridge University Press, 2007.

[4] S. Bistarelli and F. Bonchi, "Soft constraint based pattern mining," *Data and Knowledge Engineering*, vol. 62, 2007, pp. 118-137.

[5] I. Bratko, *Prolog Programming for Artificial Intelligence*, 3rd ed., Pearson, 2001.

[6] A. Cegler and J. Roddick, "Association mining," *ACM Computing Surveys*, vol.38, no.2, 2006.

[7] L. De Raedt, T. Guns, and S. Nijssen, "Constraint programming for itemset mining," in *Proc. KDD*, 2008, pp. 204-212.

[8] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge discovery in databases: an overview," *AI Magazine*, vol. 13, no. 3, 1992, pp. 57-70.

[9] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed., Morgan Kaufmann, 2006.

[10] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD*, 2000, pp. 1-12.

[11] J. Hu and X. Li, "Association rules mining including weak-support modes using novel measures," *WSEAS Transactions on Computers*, vol. 8, issue 3, 2009, pp. 559-568.

[12] M.C. Hung, S.Q. Weng, J. Wu, and D.L. Yang, "Efficient mining of association rules using merged transactions," *WSEAS Transactions on Computers*, vol. 5, issue 5, 2006, pp. 916-923.

[13] N. Kerdprasop and K. Kerdprasop, "Recognizing DNA splice sites with the frequent pattern mining technique," *International Journal of Mathematical Models and Methods in Applied Science*, vol.5, issue 1, 2011, pp. 87-94.

[14] R. Kuusik and G. Lind, "Algorithm MONSA for all closed sets finding: basic concepts and new pruning techniques," *WSEAS Transactions on Information Science & Applications*, vol. 5, issue 5, 2008, pp. 599-611.

[15] S.-H. Nienhuys-Cheng and R.D. Wolf, *Foundations of Inductive Logic Programming*, Springer, 1997.

[16] J. Pei and J. Han, "Can we push more constraints into frequent pattern mining?" in *Proc. ACM SIGKDD*, 2000, pp. 350-354.
[17] J. Pei, J. Han, and L. Lakshmanan, "Pushing convertible constraints in frequent itemset mining," *Data Mining and Knowledge Discovery*, vol. 8, 2004, pp. 227-252.
[18] R. Srikant, Q. Vu, and R. Agrawal, "Mining association rules with item constraints," in *Proc. KDD*, 1997, pp. 67-73.
[19] H. Sug, "Discovery of multidimensional association rules focusing on instances in specific class," *International Journal of mathematics and Computers in Simulation*, vol. 5, issue 3, 2011, pp. 250-257.
[20] H. Toivonen, "Sampling large databases for association rules," in *Proc. VLDB*, 1996, pp. 134-145.
[21] G. Yu, S. Shao, and X. Zeng, "Mining long high utility itemsets in transaction databases," *WSEAS Transactions on Information Science & Applications*, vol. 5, issue 2, 2008, pp. 202-210.
[22] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "Parallel algorithm for discovery of association rules," *Data Mining and Knowledge Discovery*, vol. 1, 1997, pp. 343-374.

Nittaya Kerdprasop is an associate professor and the director of Data Engineering research unit, school of computer engineering, Suranaree University of Technology, Thailand. She received her B.S. in radiation techniques from Mahidol University, Thailand, in 1985, M.S. in computer science from the Prince of Songkla University, Thailand, in 1991 and Ph.D. in computer science from Nova Southeastern University, U.S.A., in 1999. She is a member of IAENG, ACM, and IEEE Computer Society. Her research of interest includes Knowledge Discovery in Databases, Data Mining, Artificial Intelligence, Logic and Constraint Programming, Deductive and Active Databases.

Kittisak Kerdprasop is an associate professor at the school of computer engineering and one of the principal researchers of Data Engineering research unit, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in computer science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in computer science from Nova Southeastern University, USA, in 1999. His current research includes Data mining, Machine Learning, Artificial Intelligence, Logic and Functional Programming, Probabilistic Databases and Knowledge Bases.

Constraint Mining in Business Intelligence: A Case Study of Customer Churn Prediction

Nittaya Kerdprasop, Phaichayon Kongchai and Kittisak Kerdprasop

Data Engineering Research Unit,
School of Computer Engineering,
Suranaree University of Technology, Thailand
nittaya@sut.ac.th, zaguraba_ii@hotmail.com, kerdpras@sut.ac.th

Abstract

In the era of digital technologies, most enterprises have collected huge amount of data in an electronic form. Business intelligence technology has emerged as a tool to support information summarization, pattern extracting, knowledge discovery, and other knowledge-related tasks. The main part of most business intelligence software is the data mining engine to analyze and report relationships that exist in the stored data. Visualization tools are created to help data analysts easily explore the induced information. For extremely large amount of data stored in the data warehouse and data marts, simply explore information and knowledge through the visualize tool is not possible. We thus propose to put more constraints in the data mining engine of the BI software. We design the framework of the proposed BI system to predict customer churn in the telecommunication industry. The logic-based implementation and performance testing results of the constraint-based pattern mining are also illustrated in this paper.

Keywords: *Constraint data mining, Pattern induction, Business intelligence, Customer churn prediction, Constraint logic programming*

1. Introduction

Business intelligence (BI) is a broad term normally used to refer to any aspect of computer-based business applications including decision support, information management, marketing automation, and intelligent data analysis [5, 10, 14]. The task of automatically extracting patterns from data related to decision making is normally done by applying statistical techniques [5]. Such methods cannot keep pace with the exponential growth of electronic data. Business analysts are gradually exploiting a faster tool of data mining techniques. Therefore, current BI software in the market contains more or less some modules of intelligent analysis based on data mining to extract useful information hidden in the enterprise databases. The extracted information from this automatic process is however tremendous in its quantity. Analysts have to post-process the analysis results by thoroughly exploring and selecting only the most informative knowledge reported to executives. Constraint data mining is thus lately proposed by several researchers [6, 13, 22, 23, 24] as a technique to alleviate the problem of superfluous knowledge.

In this paper, we propose a framework of incorporating a constraint-based pattern mining as a knowledge discovery module in the BI software. Such module can help analysts filter from large amount of knowledge the most relevant ones to their interest. The design and implementation of our pattern mining feature are based on the

association rule induction [1]. Querying the induced rules can also be performed through the logic-based language based on the Prolog syntax. The mining steps of our system are constrained by preferences, which are to be identified by analyzer. The experimentation to evaluate to performance of the proposed system has been done through the analysis of customer churn.

Customer churn, also known as customer attrition or customer turnover, is the loss of existing customers to another company or service provider. Business sectors take customer churn as a serious subject because the cost of retaining current customers is much lower than acquiring the new ones [25]. Conventional statistical methods such as logistic regression analysis [17] are normally adopted to analyze and predict churning customers. In the context of business intelligence, emerging techniques from the data mining research field can also be applied to help analyzing churn customer in more various aspects than the conventional methods. Automatic data analysis by means of data mining and machine learning technologies has long been applied to the problem of churn analysis (as summarized in Table 1).

Table 1. Summary of data mining techniques applied to churn analysis and prediction

Literature	Year	Business Sector	Data Mining Techniques
[15]	1997	Finance	Rough set theory
[11]	1998	Finance	Genetic algorithm, Rough set theory, Decision tree induction, Logistic regression
[18]	2003	Telecommunication	Naïve Bayes
[28]	2005	Telecommunication	Support vector machine, Neural network, Decision tree induction, Naïve Bayes
[27]	2006	Telecommunication	Decision tree induction, Logistic regression
[17]	2006	Finance	Logistic regression
[2]	2007	Telecommunication	Decision tree induction
[20]	2009	Telecommunication	Hybrid of decision tree induction and logistic regression
[4]	2009	Finance, Telecommunication, Mass media, Retail	Decision tree induction, Logistic regression
[21]	2010	Telecommunication	Principal component analysis and clustering
[7]	2010	Finance, Supermarket	Probabilistic tree, AdaBoost
[9]	2010	Finance	Support vector machine with rule representation
[8]	2011	Finance, Retail, Telecommunication	Boosting, Bagging
[26]	2011	Telecommunication	Ant colony optimization, Support vector machine with rule representation
[19]	2011	Shipping	Rule induction
[12]	2011	Finance	Similarity-based approach

Most work on customer churn analysis aims at inducing an accurate churner/non-churner classification model. Besides high predictive accuracy of the model, comprehensibility is also an important issue as pointed out in recent work on churn prediction [12, 19, 26]. We also agree to the comprehensibility and applicability issues in customer churn analysis. The model representation should be in a form that is easy to understand by most users, not just the experts. We thus consider deliver model induction results as a set of association rules. In this paper, we also propose a framework to incorporate induced model to the decision support system. The design and implementation of constraint-based rule induction module are main contribution of this paper.

2. Pattern Analysis Framework and Mining Method

We design the inductive customer churn analysis framework (Figure 1) with the main purpose of providing early suggestion to strategic planners before customers actually leaving the company. The content management module is the part to access customer details from the stored data. Not every single detail is to be used by the inductive module, therefore the three modules (that is, content segmentation, content conversion, and DM format manager) are necessary for screening and extracting potentially useful features from the database. Selected data in an appropriate format are then sent to the knowledge management module to analyze and induce model that can characterize customers' patterns and predict future events from current situation. These induced models are considered valuable knowledge that will be finally sent back to generate actionable suggestion to the strategic planners.

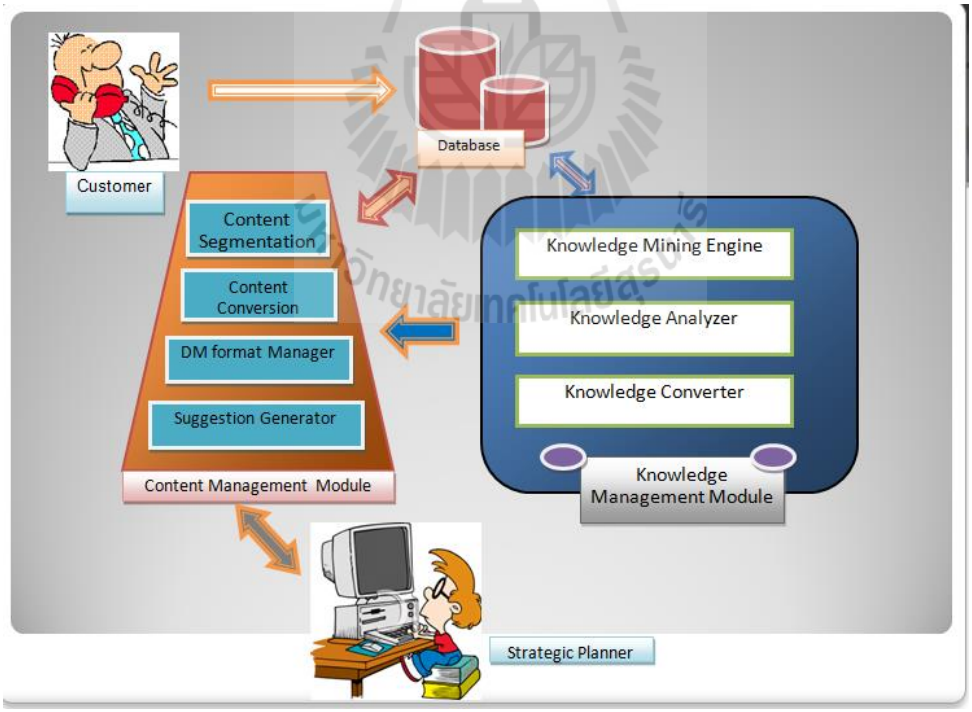


Figure 1. The pattern analysis framework to induce knowledge for supporting strategic decision

The focus of our design is the knowledge mining engine, which is the main part of the proposed business intelligence framework. The inductive algorithm in our framework is the extension of Apriori [1], which is the most well-known algorithm for association mining. We extend (in Figure 2) conventional association mining steps by considering constraints that can possibly post by analyzers or strategic planners to search for association rules that are really related to their objectives. Any irrelevant rules will automatically be removed.

Algorithm Constraint-Association-Mining

//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 ∈ D { // scan D for counts
(5) C_1 = subset(C_k, t)
(6) for each candidate c ∈ 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 $\cup_k L_k$

Figure 2. A constraint-based association mining method

3. Implementation and Running Results

We implement knowledge mining engine with the constraint-based logic programming paradigm using Eclipse 6.0 constraint system. The implementation of the constraint-based association mining is illustrated in Figure 3. On running the implemented program, we use the churn data in telecommunication industry [3, 16]. The data set contains information of 3333 customers. In the original data set, each customer record has 21 features (or variables) in which the last one is the label churn/non-churn. Details of these features are explained in Table 2. The first step of our experimentation is feature selection; the selected 12 features are state, account length, area code, international plan, voice mail plan, number vmail messages, total day calls, total eve calls, total night calls, total intl calls, number customer service calls, and churn. The other nine features are removed because of their insignificance in inducing model.

```
:-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(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),  findNotSubset(R1,R2,NotSubset),
            findSubOk(R2,Items,MinSup,R,LenItem1) ), findRule(R-Items-Conf,LenItem1).

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(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(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(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(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([]-_-_,[]).
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([],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 .
```

Figure 3. Implementation of the proposed constraint-association-mining algorithm

Table 2. Variable details of the customer churn data

Variable name	Data type	Description
state	discrete	Name of 50 states and District of Columbia
account length	continuous	How long account has been active
area code	continuous	
phone number	discrete	A surrogate for customer ID
international plan	discrete	Dichotomous categorical, yes or no
voice mail plan	discrete	Dichotomous categorical, yes or no
number vmail messages	continuous	Number of voice mail messages
total day minutes	continuous	Minutes customer used service during the day
total day calls	continuous	
total day charge	continuous	
total eve minutes	continuous	Minutes customer used service during evening
total eve calls	continuous	
total eve charge	continuous	
total night minutes	continuous	Minutes customer used service during the night
total night calls	continuous	
total night charge	continuous	
total intl minutes	continuous	Minutes customer used service to make international calls
total intl calls	continuous	
total intl charge	continuous	
number customer service calls	continuous	
churn	discrete	Dichotomous categorical, true or false

We then performed a series of eight experiments on the selected churn data set to induce association rules with various constraints:

Exp. 1: Rules are to be induced with thresholds: minimum support = 50 (that means there must be at least 50 records from the total of 3333 satisfying the rule's content) and minimum confidence = 80%. (The other experiments also specify the same minimum support and confidence.)

Exp. 2: Rules must contain the feature churn_False (that is, customer is non-churner).

Exp. 3: Rules must have at least three features.

Exp. 4: Rules must NOT contain the feature 'churn_False'.

Exp. 5: Rules must contain the feature 'churn_False' at the then-part of the rule.

Exp. 6: Rules must contain either the feature 'churn_False', or 'churn_True'.

Exp. 7: Rules must contain both the feature 'churn_True' and 'vMailPlan_no'.

Exp. 8: Rules must have at least three features, must contain both 'churn_False' and 'vMailPlan_no', must NOT contain either the feature 'vMailMessage_0', or 'intlCalls_2', and the target clause of the rules must be 'churn_False'.

Running result of experiment 8 is shown in Figure 4. We comparatively illustrate the eight experimental results in Figure 5.

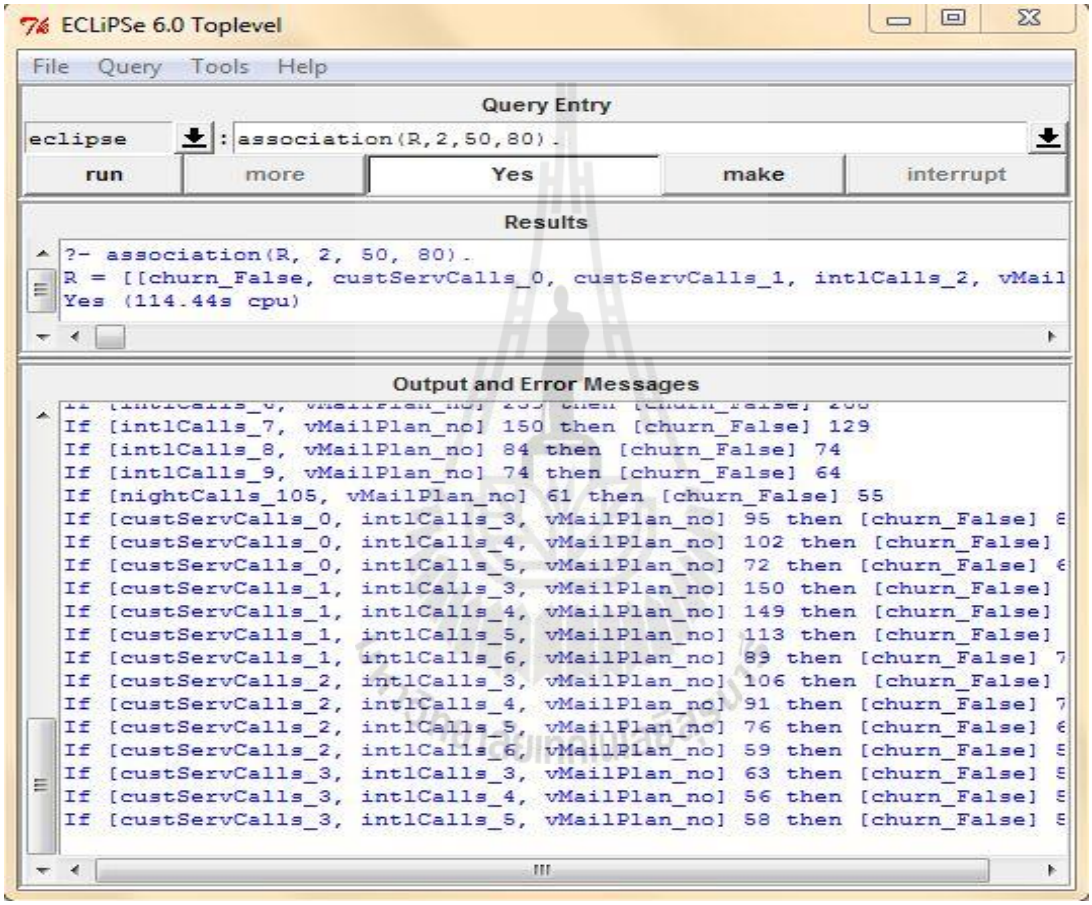


Figure 4. Running results of experiment 8: minimum support = 50 and minimum confidence = 80%. The results must contain rules that has at least three features, must contain both 'churn_False' and 'vMailPlan_no', the results must NOT contain either the feature 'vMailMessage_0', or 'intlCalls_2', and the target clause of the rules must be 'churn_False'

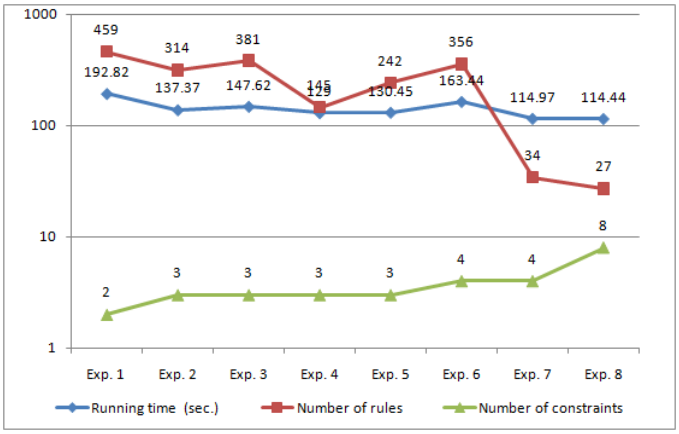


Figure 5. A comparison of computational time usage and number of rules received from varying constraints in each of the eight experiments

In Figure 5, we plot graph on a logarithmic scale for ease of comparison. Experiment 1 represents the conventional Apriori association mining in which the inherent constraints are minimum support and confidence. From experiments 2 to 8, we extend the constraints to the specification of the desired association rules such as features that must appear/not-appear in the final result, number of features in the rules, or specific feature in the consequent part of the rules. It can be noticed that when we increase the number of constraints, the number of association rules in the final result decreases considerably. We observe that running time also decreases as we add more constraints, but not at a significant rate.

4. Conclusion

A major task of customer relationship management department in many companies is customer churn analysis. The objective of this kind of analysis is to gain insight into consumers' behavior who are about to leave for another service company. Timely detection is believed to prevent these customers from attrition. Retaining current customers are known to take less effort and budget than acquiring new customers. The cost effectiveness is even higher if customers are valuable ones. Most business is therefore taking the customer retention issue seriously. Customer churn prediction models are a kind of tool that help marketing planners to sense the churning before it actually happens. Prediction models are conventionally built by the systematic process using statistical methods such as regression analysis. Since the emergence of new technology such as data mining, more and more business analysts have paid attention to this new technology. Many data mining methods including decision tree induction, support vector machines, rule induction, and so on, have been applied to the churn analysis task.

We propose in this paper that data mining methods suitable for business applications should not only yield high accuracy models, but they should provide comprehensible models for non-experts to understand. Black box models are obviously not easy to understand. We consider the rule induction method based on the association mining algorithm. Our proposed method of rule induction also incorporates module for users to specify constraints on the induced rules. The implementation based on the concept of constraint logic programming is illustrated in the paper. We present the design

framework of our rule induction module as a part of an inductive decision support system. The completion of other supporting modules in our design is our future research direction.

Acknowledgements

This research was supported by the SUT Research and Development Fund, Suranaree University of Technology.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases", Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, **(1994)**, pp. 487-499.
[2] L. Bin, S. Peiji and L. Juan, "Customer churn prediction based on the decision tree in personal handyphone system service", Proceedings of 2007 International Conference on Service Systems and Service Management, Chengdu, China, **(2007)**, pp. 1-5.
[3] C. L. Blake and C. J. Merz, Churn data set. UCI Repository of Machine Learning Databases, [http://www.ics.uci.edu/~mlearn/MLRepository.html], University of California, Irvine, **(1998)**.
[4] J. Burez and D. Van den Poel, "Handling class imbalance in customer churn prediction", Expert Systems with Applications, vol. 36, **(2009)**, pp. 4626-4636.
[5] S. Chaudhuri, U. Dayal and V. Narasayya, "An overview of business intelligence technology", Communication of the ACM, vol. 54, no. 8, **(2011)**, pp. 88-98.
[6] M. Darbari and N. Dhanda, "Applying constraints in model driven knowledge representation framework", International Journal of Hybrid Information Technology, vol. 3, no. 3, **(2010)**, pp. 15-22.
[7] K. W. De Bock and D. Van den Poel, "Ensembles of probability estimation trees for customer churn prediction", Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, Cordoba, Spain, **(2010)**, pp. 57-66.
[8] K. W. De Bock and D. Van den Poel, "An empirical evaluation of rotation-based ensemble classifiers for customer churn prediction", Expert Systems with Applications, vol. 38, **(2011)**, pp. 12293-12301.
[9] M. A. H. Farquad, V. Ravi and S. Bapi Raju, "Rule extraction from support vector machine using modified active learning based approach: an application to CRM", Proceedings of the 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Cardiff, U.K., **(2010)**, pp. 461-470.
[10] R. Fitriana, Eriyatno and T. Djatna, "Progress in business intelligence system research: a literature review", International Journal of Basics & Applied Sciences, vol. 11, no. 3, **(2011)**, pp. 96-105.
[11] A. E. Eiben, A. E. Koudijs and F. Slisser, "Genetic modeling of customer retention", Proceedings of the First European Workshop on Genetic Programming, Paris, France, **(1998)**, pp. 178-186.
[12] M. Gorgoglione and U. Panniello, "Beyond customer churn: generating personalized actions to retain customers in a retail bank by a recommender system approach", Journal of Intelligent Learning Systems and Applications, vol. 3, no. 2, **(2011)**, pp. 90-102.
[13] M. Gouider and A. Farhat, "Mining multi-level frequent itemsets under constraints", International Journal of Database Theory and Application, vol. 3, no. 4, **(2010)**, pp. 15-34.
[14] O. Isik, M. C. Jones and A. Sidorova, "Business intelligence (BI) success and the role of BI capabilities", Intelligent Systems in Accounting, Finance and Management, vol. 18, **(2011)**, pp. 161-176.
[15] W. Kowalczyk and F. Slisser, "Modelling customer retention with rough data models, Proceeding of the First European Symposium on Principles of Data Mining and Knowledge Discovery", Trondheim, Norway, pp. 4-13 **(1997)**
[16] D. T. Larose, "Discovering Knowledge in Data: An Introduction to Data Mining", John Wiley & Sons, **(2005)**.
[17] T. Mutanen, "Customer churn analysis – a case study", Research Report, No. VTT-R-01184-06, Technical Report Center of Finland, **(2006)**.
[18] S. V. Nath and R. S. Behara, "Customer churn analysis in the wireless industry: a data mining approach", Proc. of Annual Meeting of the Decision Sciences Institute, Washington, D.C., U.S.A., **(2003)**, pp. 505-510.
[19] B. Padmanabhan, A. Hevner, M. Cuenco and C. Shi, "From information to operations: service quality and customer retention", ACM Transactions on Mgt. Information Systems, vol. 2, no. 4, Article 21, **(2011)**.
[20] J. Qi, L. Zhang, Y. Liu, L. Li, Y. Zhou, Y. Shen, L. Liang and H. Li, "ADTreesLogit model for customer churn prediction", Annals of Operations Research, vol. 168, no. 1, **(2009)**, pp. 247-265.
[21] T. Sato, B. Q. Huang, Y. Huang, M.-T. Kechadi and B. Buckley, "Using PCA to predict customer churn in telecommunication dataset", Proceedings of the 6th International Conference on Advanced Data Mining and Applications, Chongqing, China, **(2010)**, pp. 326-335.

[22] M. Shahriar and S. Anam, "Towards data quality and data mining busing constraints in XML", International Journal of Database Theory and Application, vol. 2, no. 1, **(2009)**, pp. 23-30.
[23] M. Shahriar and J. Liu, "Constraint-based data transformation for integration: an information system approach", International Journal of Database Theory and Application, vol. 3, no. 1, **(2010)**, pp. 53-61.
[24] R. Srikant, Q. Vu and R. Agrawal, "Mining association rules with item constraints", Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, **(1997)**, pp. 67-73.
[25] N. B. Syam and J. D. Hess, "Acquisition versus retention: competitive customer relationship management", Working Paper, University of Houston, Houston, Texas, U.S.A., **(2006)**.
[26] W. Verbeke, D. Martens, C. Mues and B. Baesens, "Building comprehensible customer churn prediction models with advanced rule induction techniques", Expert Systems with Applications, vol. 38, **(2011)**, pp. 2354-2364.
[27] L.-S. Yang and C. Chiu, "Knowledge discovery on customer churn prediction", Proceedings of the 10th WSEAS International Conference on Applied Mathematics, Dallas, U.S.A., **(2006)**, pp. 523-528.
[28] Y. Zhao, B. Li, X. Li, W. Liu and S. Ren, "Customer churn prediction using improved one-class support vector machine", Proceedings of the 1st International Conference on Advanced Data Mining and Applications, Wuhan, China, **(2005)**, pp. 300-306.

Authors

Nittaya Kerdprasop is an associate professor at the School of Computer Engineering, Suranaree University of Technology, Thailand. She received her bachelor degree in Radiation Techniques from Mahidol University, Thailand, in 1985, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A, in 1999. She is a member of ACM and IEEE Computer Society. Her research of interest includes Knowledge Discovery in Databases, Artificial Intelligence, Logic Programming, and Intelligent Databases.

Phaichayon Kongchai is currently a doctoral student with the School of Computer Engineering, Suranaree University of Technology, Thailand. He received his bachelor degree in Computer Engineering from Suranaree University of Technology (SUT), Thailand, in 2010, and master degree in Computer Engineering from SUT in 2012. His current research includes Constraint Data Mining, Association Mining, Functional and Logic Programming Languages, Statistical Machine Learning.

Kittisak Kerdprasop is an associate professor and chair of the School of Computer Engineering, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A., in 1999. His current research includes Data mining, Artificial Intelligence, Functional and Logic Programming Languages, Computational Statistics.

International Journal of Software Engineering and Its Applications
Vol.7, No.5 (2013), pp.269-282
http://dx.doi.org/10.14257/ijseia.2013.7.5.24

Building and Querying a Decision Tree Model with Constraint Logic Programming

Nittaya Kerdprasop, Fonthip Koongaew and Kittisak Kerdprasop

Data Engineering Research Unit,
School of Computer Engineering,
Suranaree University of Technology, Thailand
nittaya@sut.ac.th

Abstract

Decision tree induction has gained its popularity as an effective automated method for data classification mainly because of its simple, easy-to-understand, and noise-tolerant characteristics. The induced tree reveals the most informative attributes that can best characterize training data and accurately predict classes of unseen data. Despite its predictive power, the tree structure can be overly expanded or deeply grown when the training data do not show explicit patterns. Such bushy and deep trees are difficult to comprehend and interpret by humans. We thus propose a logic-based method to query over a complicate tree structure to extract only parts of the tree model that are really relevant to users' interest. The implementation using ECLiPSe constraint language to perform constrained search over a decision tree model is given in this paper. The illustrative examples on medical domains support our hypothesis regarding simplicity of constrained tree-based patterns.

Keywords: *Constraint data mining, Pattern induction, Querying, Classification tree, Decision tree induction, Constraint logic programming*

1. Introduction

A decision tree is a hierarchical structure comprising of nodes and edges. Each internal node, including the root of a tree, represents a decision choice. All possible decision choices are represented as branches from a node. The terminal decision outcome appears at the leaf node [13, 20, 24]. Machine learning and data mining communities consider the automatic process of building a decision tree from the training data with labeled decision outcomes as a classification problem (if the decision outcomes are continuous values rather than the nominal ones, it is referred to as a regression problem).

Given a training data set with decision attributes and the labeled outcome, the classification process aims at constructing an optimal classifier (or a tree) with minimum classification error. The tree-based classification process is thus composing of the tree-building phase and the pruning phase [12].

Building a decision tree from a set of training data is to partition data with mixing decision classes down the tree until each leaf node contains data instances with pure class. For a large data set with many attributes, constructed tree may contain branches that reflect chance occurrences, instead of the true relationship underlying the data subset. Many tree induction algorithms [5, 16, 18, 19] apply pruning strategies as

ISSN: 1738-9984 IJSEIA

subsequent steps following the tree-building phase. A tree pruning operation, either pre-pruning or post-pruning, involves modifying a tree structure to be more simplified.

The built tree is considered corresponding to a collection of decision rules when traversing from the root node down to its leaves. A tree, or a set of decision rules, is normally applied as a classifier to help identifying appropriate decision on the future event or predicting class of unseen object. A classifier also serves as a generalized model of the training data set. Due to its simplicity and efficiency, decision tree induction has been applied in many research and application areas ranging from grid computing [4], finance [15], engineering [22], health care industry [1, 14], medicine [11], to bioinformatics [23].

Even with a tree pruning operation, a final tree structure can become a complex model when applying to the real world data with so many instances and attributes. General users and decision-makers normally prefer less complex decision trees. Many researchers solve this problem by simplifying tree structure with the trade off in classification accuracy [3], or applying some constraints during the tree-building phase [8, 9]. Our proposed method is different from most existing mechanism in that we deal with complexity problem after the tree induction phase.

We propose to construct a complete decision tree with the top-down induction approach [17]. Then we suggest that the users can manipulate the structure to be less complicate and truly reflect their interest by posing querying on this tree structure. Querying the tree model also appears in the literature [2, 6] but with quite a different purpose. Previous work on querying tree aims at extracting meta-data and statistical information from the model. Our work, on the other hand, focuses on serving users to extract only parts of the tree model that are of their interest.

We present the method and the detail of our implementation in Section 2. The prototype of our implementation based on the logic programming paradigm is also illustrated. Tree induction is normally implemented with SQL language [10, 21]; we demonstrate in this paper that it can be more effective to implement with constraint logic programming using ECLiPSe. Section 3 shows querying techniques. Efficiency of our implementation on medical data [7] is demonstrated in Section 4. Conclusion appears as the last section of this paper.

2. Building a Decision Tree Model with Logic Programming

We implement the decision tree induction method based on the ID3 algorithm [17] using logic programming paradigm and run with the ECLiPSe constraint programming system (http://www.eclipseclp.org). Program and data set are in the same format: that is, Horn clauses. Example of breast cancer data set [7] is shown in Figure 1. Format of a data set is *data([[data instances]+[attribute set]])*.

Program coding is given in Appendix. To run the program, users may simply call the predicate “run” as shown in Figure 2. The output of the program is a tree model that has been displayed as a textual format. Each branch of the tree has also been transformed as a decision rule.


```
data([ [age-"30-39", menopause-premeno, tumor_size-"30-34", inv_nodes-"0-2",node_caps-no,
        deg_malig-3,breast-left,breast_quad-left_low,irradiat-no,class-no_recurrence_events],
        …
       [age-"40-49", menopause-premeno, tumor_size-"20-24", inv_nodes-"0-2",node_caps-no,
        deg_malig-2,breast-right,breast_quad-right_up,irradiat-no,class-no_recurrence_events],
+
[ [class-no_recurrence_events, class-recurrence_events],
  [age-"10-19", age-"20-29", age-"30-39", age-"40-49",age-"50-59", age-"60-69", age-
    "70-79",age-"80-89",age-"90-99"],
 [menopause-lt40, menopause-ge40, menopause-premeno],
 [tumor_size-"0-4",tumor_size-"5-9",tumor_size-"10-14",tumor_size-"15-
    19",tumor_size-"20-24",tumor_size-"25-29",tumor_size-"30-34",tumor_size-"35-
    39",tumor_size-"40-44", tumor_size-"45-49",tumor_size-"50-54",tumor_size-"55-
    59"],
 [inv_nodes-"0-2",inv_nodes-"3-5",inv_nodes-"6-8",inv_nodes-"9-11",inv_nodes-"12-
    14", inv_nodes-"15-17",inv_nodes-"18-20",inv_nodes-"21-23",inv_nodes-"24-
    26",inv_nodes-"27-29",inv_nodes-"30-32",inv_nodes-"33-35",inv_nodes-"36-39"],
 [node_caps-yes, node_caps-no],
 [deg_malig-1, deg_malig-2, deg_malig-3],
 [breast-left, breast-right],
 [breast_quad-left_up, breast_quad-left_low, breast_quad-right_up, breast_quad-
    right_low, breast_quad-central],
 [irradiat-yes, irradiat-no]

] ).
```

Figure 1. Breast Cancer Data Set in a Horn Clause Format

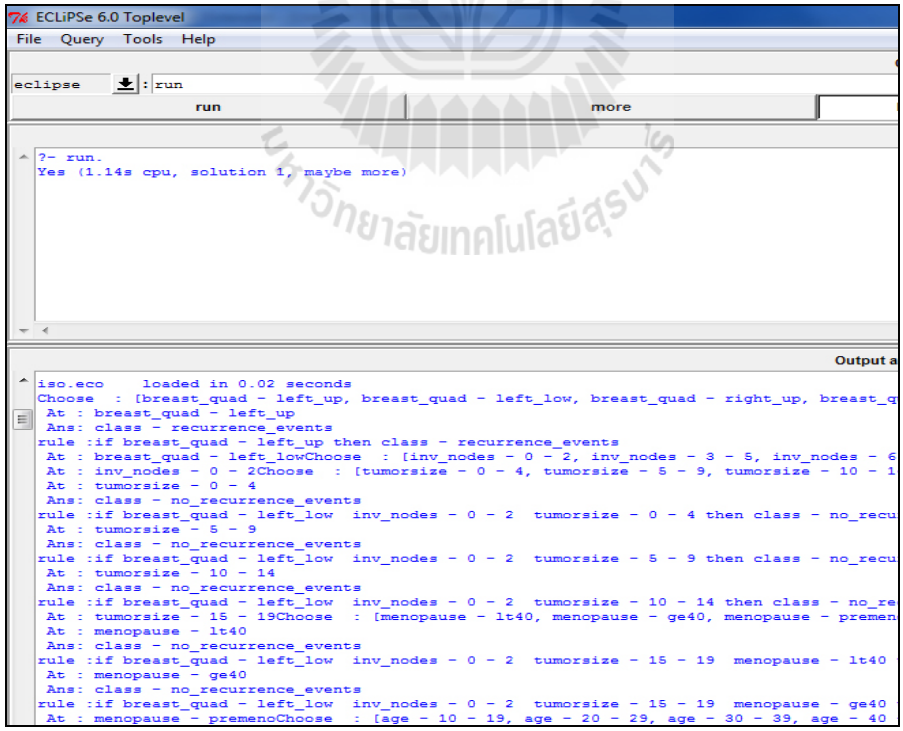


Figure 2. A Screenshot of Program Running on the Breast Cancer Data Set

3. Querying a Tree Model

Once a decision tree model has been created as shown in Figure 2, user can then query the model with 7 different styles of constraints:

findRule([]) : to display all rules extracted from a decision tree model.

findRule([X]) : to display only rules that are relevant to the attribute X (such as *irradiat* for rules that contain the attribute "irradiat" or *irradiat – yes* as a query to show all rules with "irradiat with value yes"); number of attributes is not limited.

findRule([\+X]) : to display all rules except the ones with attribute X (\+ means "not").

findRule([X1,X2]) : to display all rules that satisfy the condition X1 AND X2 (negation \+ can also be applied to the attributes X1, X2).

findRuleOr([[X1],[X2]]) : to display all rules satisfied either the attribute X1, or X2 (negation \+ can also be applied to the attributes X1, X2).

findRuleOr([[\+X1],[\+X2]]) : to display all rules extracted from a decision tree model

findRuleOr([[X1,\+X2],[\+X3,X4]]) : to display all rules that satisfy the compound operations "(X1 AND (NOT X2)) OR ((NOT X3) AND X4)".

We show the result of *findRule([])* querying over a tree model induced from the breast cancer data set in Figure 3. Then constraining the result with the query *findRule ([class - recurrence_events])*. The query result is shown in Figure 4.

```
if breast_quad - left_low  inv_nodes - 15-17  age - 40-49 then class -
if breast_quad - left_low  inv_nodes - 15-17  age - 50-59 then class -
if breast_quad - left_low  inv_nodes - 24-26 then class - recurrence_ev
if breast_quad - right_up  tumor_size - 0-4 then class - no_recurrence_
if breast_quad - right_up  tumor_size - 5-9 then class - no_recurrence_
if breast_quad - right_up  tumor_size - 10-14 then class - no_recurrenc
if breast_quad - right_up  tumor_size - 15-19 then class - no_recurrenc
if breast_quad - right_up  tumor_size - 20-24  inv_nodes - 0-2 then cla
if breast_quad - right_up  tumor_size - 20-24  inv_nodes - 3-5 then cla
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 1 then class
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 2  age - 40-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 2  age - 50-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 2  age - 50-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 2  age - 60-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 3  age - 30-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 3  age - 40-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 3  age - 50-
if breast_quad - right_up  tumor_size - 25-29  deg_malig - 3  age - 60-
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 1 then class
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 2  node_caps
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 2  node_caps
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 2  node_caps
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 2  node_caps
if breast_quad - right_up  tumor_size - 30-34  deg_malig - 2  node_caps
```

Figure 3. A Set of Rules Obtained from the Query *findRule([])*

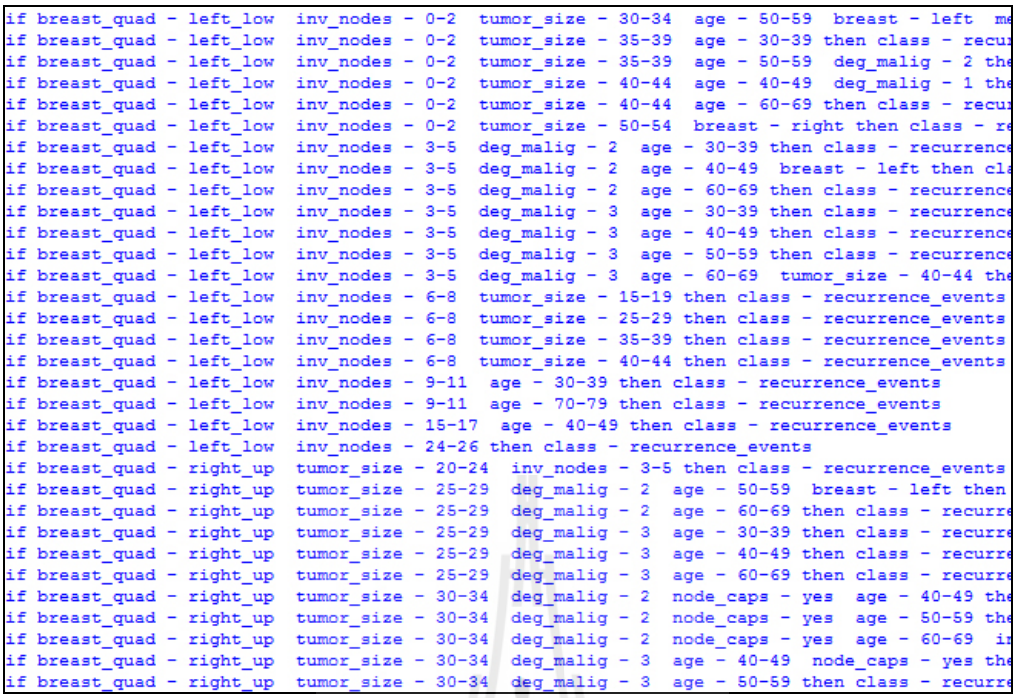


Figure 4. Result of Constrained Search with the Query *findRule ([class - recurrence_events])*

4. Experimentation and Results

To test the performance of the proposed method to query over a discrete tree model, we use the standard UCI data repository [7] including the hepatitis data set (155 instances and 15 attributes), breast cancer data set (286 instances, 10 attributes), and thyroid disease data set (2800 instances and 16 attributes). Query result over a hepatitis data set with the query is shown in Figure 5. For each data set we test the system with seven different kinds of queries as summarized in the followings.

Hepatitis data set

findRule([])
findRule([bilirubin - "0.1-1.0"])
findRule([\+bilirubin - "0.1-1.0"])
findRule([bilirubin - "0.1-1.0" , class - live])
findRule([\+bilirubin - "0.1-1.0" , class - live])
findRuleOr([[ascites - yes],[bilirubin - "0.1-1.0,age - "31-40"]]) % Version 1
findRuleOr([[ascites - yes],[bilirubin - "0.1-1.0" ,age - "31-40"]]) % Version 2

Breast cancer data set

findRule([])
findRule([breast_quad - right_up])
findRule([\+breast_quad - right_up])
findRule([breast_quad - right_up , class - no_recurrence_events])
findRule([\+breast_quad - right_up , \+class - no_recurrence_events])
findRuleOr([[breast_quad - left_low , inv_nodes - "6-8"] , [breast_quad - left_low, inv_nodes - "6-8", tumor_size - "15-19"]]) % Version 1

findRuleOr([[breast_quad - left_low , inv_nodes - "6-8"] , [breast_quad - left_low, inv_nodes - "6-8", tumor_size - "15-19"]]) % Version 2

Thyroid disease data set

findRule([])
findRule([pregnant - false])
findRule([\+pregnant - false])
findRule([query_hyperthyroid - true , query_hypothyroid - true , sex - male , on_antithyroid_medication - false])
findRule([\+query_on_thyroxine - false , \+on_antithyroid_medication , \+class - sick])
findRuleOr([[query_hyperthyroid - true , query_hypothyroid - false] , [\+ query_hypothyroid - true] , [\+class - negative]]) % Version 1
findRuleOr([[query_hyperthyroid - true , query_hypothyroid - false] , [\+ query_hypothyroid - true] , [\+class - negative]]) % Version 2

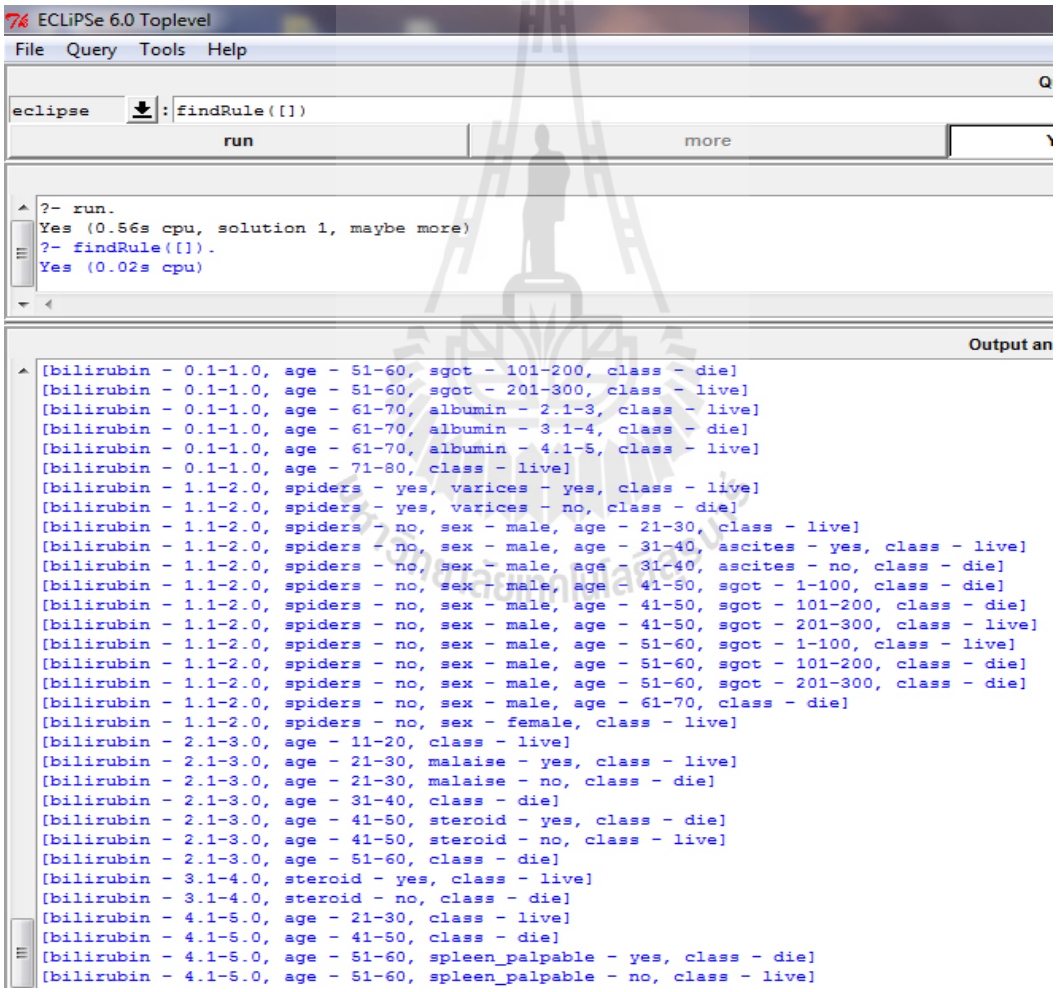


Figure 5. Running Result of Querying Hepatitis Data Model with the Query *findRule([])*

As a demonstration of querying the model, we show only one example. For the query "*findRuleOr([[\+referral_source-svhc, \+ query_hypothyroid - false, class - sick], [\+ query_ hypothyroid - false]])*", its result is as follows:

Rule 1: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - female on_thyroxine - false query_hyperthyroid - false psych - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false lithium - false goitre - false hypopituitary - false
then class - negative

Rule 2: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - female on_thyroxine - false query_hyperthyroid - false psych - true
then class - negative

Rule 3: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - female on_thyroxine - false query_hyperthyroid - true
then class - negative

Rule 4: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - female on_thyroxine - true
then class - negative

Rule 5: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - male query_hyperthyroid - false on_thyroxine - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false lithium - false goitre - false hypopituitary - false psych - false
then class - negative

Rule 6: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - male query_hyperthyroid - false on_thyroxine - true
then class - negative

Rule 7: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - male query_hyperthyroid - true on_antithyroid_medication - false
then class - sick

Rule 8: **if** referral_source - other sick - false tumor - false query_hypothyroid - true sex - male query_hyperthyroid - true on_antithyroid_medication - true
then class - negative

Rule 9: **if** referral_source - other sick - false tumor - true
then class - negative

Rule 10: **if** referral_source - other sick - true sex - female query_hyperthyroid - false query_hypothyroid - true
then class - negative

Rule 11: **if** referral_source - other sick - true sex - female query_hyperthyroid - true
then class - negative

Rule 12: **if** referral_source - other sick - true sex - male
then class - negative

Rule 13: **if** referral_source - stmw
then class - negative

Rule 14: **if** referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - female psych - false
then class - negative

Rule 15: **if** referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - female psych - true query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false
then class - sick

Rule 16: **if** referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - male
then class - negative

Rule 17: **if** referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - true
then class - sick

Rule 18: **if** referral_source - svhc query_hypothyroid - true on_thyroxine - true
then class - sick

Rule 19: **if** referral_source - svhd query_hypothyroid - true sex - female
then class - negative

Rule 20: **if** referral_source - svhd query_hypothyroid - true sex - male
then class - sick

Rule 21: **if** referral_source - svi query_hypothyroid - true sex - female sick - false on_thyroxine - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false
then class - negative

Rule 22: **if** referral_source - svi query_hypothyroid - true sex - female sick - false on_thyroxine - true query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false
then class - sick

Rule 23: **if** referral_source - svi query_hypothyroid - true sex - female sick - true
then class - negative

Rule 24: **if** referral_source - svi query_hypothyroid - true sex - male sick - false on_thyroxine - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false
then class - negative

Rule 25: **if** referral_source - svi query_hypothyroid - true sex - male sick - true
then class - sick

Performance of running results in terms of rule reduction, that is the simplification of a tree model, can be graphically shown in Figures 6-8.

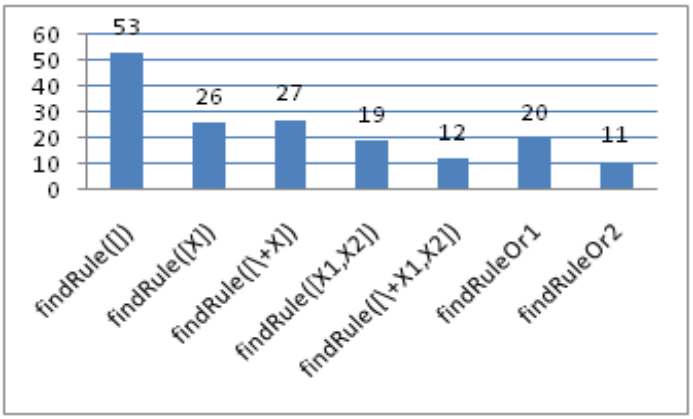


Figure 6. Performance in Terms of Model Reduction of Hepatitis Data Set

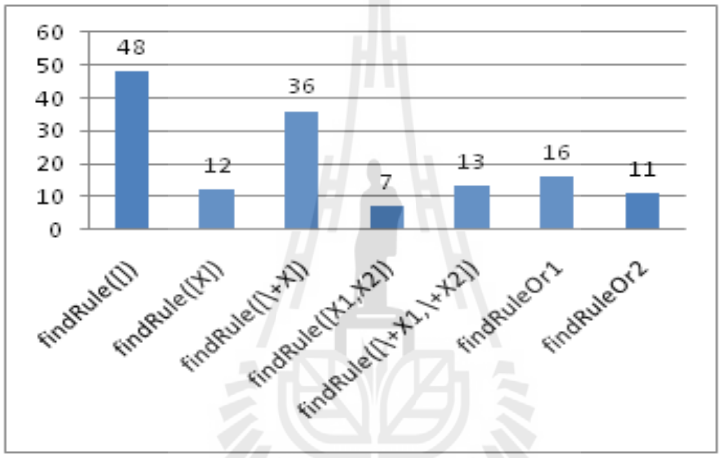


Figure 7. Performance in Terms of Model Reduction of Breast Cancer Data Set

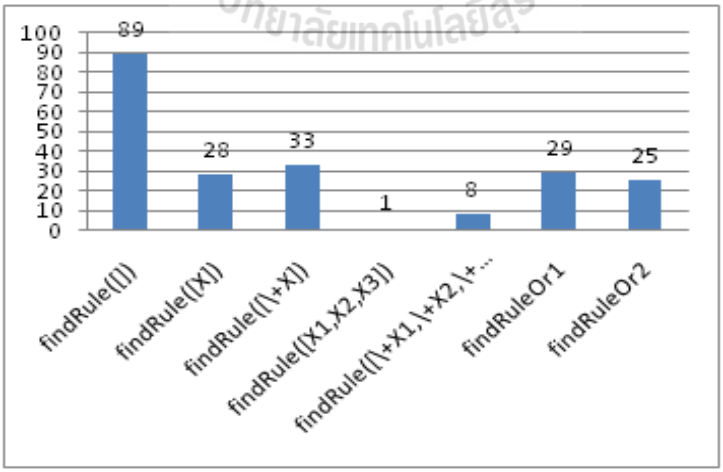


Figure 8. Performance in Terms of Model Reduction of Thyroid Disease Data Set

5. Conclusion

Classification is a data mining task that aims to induce general concept from training data. The induced concept not only explains major characteristics of the underlying data, but also acts as a classification model to predict classes of unseen data. Several learning algorithms have been proposed to induce classification concept, but the most applicable algorithm is decision tree induction. The main reason for its popularity is a simple and understandable form of a tree that has been used to represent the induced concept.

Despite its simplicity and efficiency, it could be a problem when communicating sophisticate concept as a large tree model to general users who are not an expert in decision science or computer technology. Large tree model is difficult to comprehend at a glance. Therefore, simplifying tree structure is necessary for conveying concept model to novice users. Many researchers propose a constraint-based approach during the tree-building phase to make a tree structure more simplified. We, however, consider tree simplification as a post-process of decision tree induction. We propose to grow a full decision tree. Then, apply users' preferences as a constrained search over a tree model. Only branches of a tree model that correspond to the user-specified constraints are displayed in a simple form of decision rules to the users.

The implemented prototype is expected to ease users in searching for useful knowledge from the tree model. The usability test with users who are practitioners in the field is nevertheless essential to confirm our assumption. In our future work, we also intend to consider other aspects of pushing constraints in the tree induction process.

Acknowledgements

This research was supported by the SUT Research and Development Fund, Suranaree University of Technology.

Appendix

A source code of decision tree with findRule predicates to query a tree model, implemented with constraint logic programming language ECLiPSe, is given as follows.

```
%% Program Discrete-Tree Induction with findRule queries
:- lib(listut).
:- lib(sd).
:-dynamic rule_me/1.
:-dynamic allrule/1.

append_me([H|T],L,[H|RT]) :- append_me(T,L,RT).
append_me([],L,L).

findRuleOr([H|T]) :- ( findRule(H) -> true ; ! ),
                     findRuleOr(T).
findRuleOr([]).

findRule(X) :-   allrule(L),
                 findRule(X,L).
findRule(_,[]).
findRule(X,[H|T]) :-  (findQuery(X,H)-> split_rule(H) ; true),
                      findRule(X,T).

findQuery([\+X|TX],H) :-   (findAtt(X,H) -> false ;  (findAttLabel(X,H) -> false ;  true) ),
                           findQuery(TX,H).
```



```
findQuery([X|TX],H) :-  (findAtt(X,H) -> true ;  (findAttLabel(X,H) -> true ;  false  ) ),
                        findQuery(TX,H).
findQuery([],_).

findAtt(X,[X-_|_]).
findAtt(X,[_|T]) :- findAtt(X,T).
findAtt(_,[]) :- false.

findAttLabel(X,[X|_]).
findAttLabel(X,[_|T]) :- findAttLabel(X,T).
findAttLabel(_,[]) :- false.

run :- retractall(rule_me(_)),
       retractall(allrule(_)),
       compile("//C/Users/ASUS/Desktop/id3_ok/data-sickthyroid_OK.txt"),
       data(Data+Attrs),
       main(Data,Attrs,[]),
       retractall(rule_me([_])),
       findall(X,rule_me(X),R),
       assert(allrule(R)).

main([],_,_).
main(_,[],_).
main(Data, Attrs,OldAttr) :- all_info(Data+Attrs, R1),
                             (\+hasOneClass(R1) ->  (maplist(avg_info,R1,Out),
                                                     chooseMin(Out, nil/11,OO),
                                                     OO=O/_,
                                                      writeln('Choose ':OO),
                                                     [listut]:delete(Attrs,O,NewAttrs),
                                                     (foreach(X,O),param(Data,NewAttrs,OldAttr) do
                                                     (filterData([X],Data,NewData),
                                                     write(' At':X),
                                                     append_me(OldAttr,[X],OAL),
                                                     main(NewData,NewAttrs,OAL))
                                                      ) ) ;  writeln(''),
                                getlast_goal(Data,Att-Ans),
                                write(" Ans: "), writeln(Att-Ans),
                                append_me(OldAttr,[Att-Ans],NOAL),
                                write('rule :'),
                                split_rule(NOAL)),
                                assert(rule_me(NOAL)).

split_rule(NOAL) :- mem_last(NOAL,L,RL),write('if'),
                    (foreach(RLL,RL) do write(' '),write(RLL),write(' ')),
                    write('then'),write(' '),writeln(L).

%last member in list
%mem_last([1,2,3,5,6],L,RL).
mem_last([H],H,[]).
mem_last([H|T],L,[H|RL]) :- mem_last(T,L,RL).

%mem(Data+Attrs),all_info(Data+Attrs,R).
all_info(Data+[P|Attr],R):-maplist(info(Data,P), Attr, R).

%mem(Data+_), info(Data,[p-y,p-n],[o-s,o-c,o-r], R).
%R=[[2,3], [4,0], [3,2]]
info(Data, P, O, O-R) :- maplist(info1(Data,P),O,R).
info1(Data, P, O, R) :- maplist(mcount(Data,O),P,R).

mcount(Data,O,P,Sum) :- foreach(L,Data),fromto(0,I,R,Sum),param(O,P) do
                        ((member(O,L), member(P,L))->R is I+1; R is I).

%avg_info(o-[[2,3],[4,0],[3,2]],R).
avg_info(O-LL,O/Info ):- flatten(LL,L), sumlist(L, Sum),
                         (foreach(X,LL), fromto(0,I,N,Info),param(Sum) do
                         (sumlist(X,SumX),
```

```
                                (Sum>0 -> Ratio is SumX/Sum, ! ; Ratio is 0),
                                 logInfo(X,InfoSub),
                                 N is I+(Ratio*InfoSub)
                                 )).
sumlist(L,[],L):-!.
sumlist([],[],[]):-!.
sumlist([H1|T1],[H2|T2],[HR|TR]) :- HR is H1+H2,sumlist(T1,T2,TR).

sumlists([],R,R):-!.
sumlists([H|T],PR,NR) :- sumlist(H,PR,R), sumlists(T,R,NR).

hasOneClass([]):-!.
hasOneClass([_-VL|T]) :- sumlists(VL,[],NR), hasOneClass(T,NR).

hasOneClass([],[]):-!.
hasOneClass([_-VL|T],PR) :- sumlists(VL,PR,NR), hasOneClass(T,NR).
hasOneClass([],[H|T]) :- (H=0 -> hasOneClass([],T) ; find(T)).

find([]):-!.
find([H|T]) :- (H=0 -> find(T) ; false).

% get the Last data in Fist of list in list
getlast_goal([H|_],R) :- getlast(H,R).
getlast([H],H).
getlast([_|T],R) :- getlast(T,R).

%filterData
filterData(_,[],[]).
filterData(L,[H|Data],[H|R]) :- msubset(L,H), !, filterData(L,Data,R).
filterData(L,[H|Data],R) :- \+msubset(L,H), !, filterData(L,Data,R).

%------------
msubset(S1,S2) :- foreach(X,S1), param(S2) do member(X,S2), !.
allmem([H],L) :- member(H,L), !.
allmem([H|T],L) :- member(H,L), allmem(T,L).
logInfo(XL, R) :- sumlist(XL,Sum), Sum==0, R=99, !.
logInfo(XL, R) :- sumlist(XL,Sum),
                  (foreach(X,XL), fromto(0,S,N,R), param(Sum) do
                           ( Ratio is X/Sum,
                           (Ratio>0->[iso]:log(Ratio,Log) ; Log is 1),  %log(0) is undefined
                                    [iso]:log(2,Base2),
                                    N is S-(Ratio*(Log/Base2) ) ) ).
chooseMin([],O/Tmp,O/Tmp).
chooseMin([A/H|T],O/Tmp,Min) :- (H<Tmp -> NextMin = A/H ; NextMin = O/Tmp),
                                 chooseMin(T,NextMin,Min).

 % ===== End of Program =============
```

References

[1] M. H. M. Adnan, W. Husain and N. A. Rashid, “Hybrid approaches using decision tree, naive Bayes, means and euclidean distances for childhood obesity prediction”, International Journal of Software Engineering and Its Applications, vol. 6, no. 3, **(2012)**, pp. 99-106.

[2] Y. Ben-Asher and I. Newman, “Decision trees with AND, OR queries”, Proceedings of the 10th Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, **(1995)** June 19-22, pp. 74.

[3] M. Bohanec and I. Bratko, “Trading accuracy for simplicity in decision trees”, Machine Learning, vol. 15, no. 3, **(1994)**, pp. 223-250.

[4] A. Bouyer, M. Karimi, M. Jalali and M. N. MD SAP, “A new approach for selecting best resources nodes by using fuzzy decision tree in grid resource broker”, International Journal of Grid and Distributer Computing, vol. 1, no. 1, **(2008)**, pp. 49-62.

[5] L. Breiman, J. Freidman, R. Olshen and C. Stone, “Classification and Regression Trees”, Belmont, California: Wadsworth, **(1984)**.

[6] L. Fang and K. LeFevre, "Splash: ad-hoc querying of data and statistical models", Proceedings of the 13th International Conference on Extending Database Technology, Lausanne, Switzerland, **(2010)** March 22-26, pp. 275-286.
[7] A. Frank and A. Asuncion, "UCI Machine Learning Repository", [http://archive.ics.uci.edu/ml], Irvine, University of California, School of Information and Computer Science, **(2010)**.
[8] M. Garofalakis, D. Hyun, R. Rastogi and K. Shim, "Efficient algorithms for constructing decision trees with constraints", Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, **(2000)** August 20-23, pp. 335-339.
[9] M. Garofalakis and R. Rastogi, "Scalable data mining with model constraints", ACM SIGKDD Explorations Newsletter, vol. 2, no. 2, **(2000)**, pp. 39-48.
[10] A. Hinneburg, D. Habich and W. Lehner, "COMBI-operator-database support for data mining applications", Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany, September 9-12, **(2003)**, pp. 429-439.
[11] N. Kerdprasop and K. Kerdprasop, "Knowledge induction from medical databases with higher-order programming", WSEAS Transactions on Information Science & Applications, vol. 6, no. 10, **(2009)**, pp. 1719-1728.
[12] N. Kerdprasop and K. Kerdprasop, "The development of discrete decision tree induction for categorical data", International Journal of Mathematics and Computers in Simulations, vol. 5, no. 6, **(2011)**, pp. 499-509.
[13] H. Kim and G. J. Koehler, "Theory and practice of decision tree induction", Omega International Journal of Management Science, vol. 23, no. 6, **(1995)**, pp. 637-652.
[14] M. Ko and K.-M. Osei-Bryson, "Reexamining the impact of information technology investment on productivity using regression tree and multivariate adaptive regression splines (MARS)", Information Technology and Management, vol. 9, no. 4, **(2008)**, pp. 285-299.
[15] G. Nie, W. Rowe, L. Zhang, Y. Tian and Y. Shi, "Credit card churn forecasting by logistic regression and decision tree", *Expert Systems with Applications*, vol. 38, no. 12, **(2011)**, pp. 15273-15285.
[16] K.-M. Osei-Bryson, "Post-pruning in regression tree induction: an integrated approach", *Expert Systems with Applications*, vol. 34, no. 2, **(2008)**, pp. 1481-1490.
[17] J. R. Quinlan, "Induction of decision tree", Machine Learning, vol. 1, **(1986)**, pp. 81-106.
[18] J. R. Quinlan, "Simplifying decision tree", Knowledge Acquisition for Knowledge Based Systems, vol. 1, B. Gaines and J. Boose, Eds., Academic Press, **(1989)**.
[19] J. R. Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufmann, **(1992)**.
[20] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey", IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, vol. 35, no. 4, **(2005)**, pp. 476-487.
[21] K.-U. Sattler and O. Dunemann, "SQL database primitives for decision tree classifiers", Proceedings of the 10th International Conference on Information and Knowledge Management, Atlanta, Georgia, **(2001)** November 5-10, pp. 379-386.
[22] S. M. Shaaban and H. Nabwey, "A decision tree approach for steam turbine-generator fault diagnosis", International Journal of Advanced Science and Technology, vol. 51, **(2013)** February, pp. 59-66.
[23] G. Stiglic, S. Kocbek, I. Pernek and P. Kokol, "Comprehensive decision tree models in bioinformatics, PLoS ONE, vol. 7, no. 3, article e33812, doi:10.1371/journal.pone.0033812, **(2012)**.
[24] H. S. Yazdi and N. Salehi-Moghaddami, "Multi branch decision tree: a new splitting criterion", International Journal of Advanced Science and Technology, vol. 45, **(2012)** August, pp. 91-106.

Authors

Nittaya Kerdprasop is an associate professor at the School of Computer Engineering, Suranaree University of Technology, Thailand. She received her bachelor degree in Radiation Techniques from Mahidol University, Thailand, in 1985, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A, in 1999. Her research interest includes Knowledge Discovery in Databases, Artificial Intelligence, and Logic Programming.

Fonthip Koongaew is a computer engineer and research assistant. She received her bachelor degree in Computer Engineering from Suranaree University of Technology (SUT), Thailand, in 2010, and master degree in Computer Engineering from SUT in 2012. Her current research includes Constraint Data Mining, Classification Mining, and Logic Programming Languages.

Kittisak Kerdprasop is an associate professor and chair of the School of Computer Engineering, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A., in 1999. His current research includes Data mining, Artificial Intelligence, Functional Programming Language, and Computational Statistics.

การค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ

THE DISCOVERY OF ASSOCIATION RULES USING CONSTRAINT LOGIC PROGRAMMING

ไพชยนต์ คงไชย, นิตยา เกิดประสพ และ กิตติศักดิ์ เกิดประสพ
สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์
มหาวิทยาลัยเทคโนโลยีสุรนารี อ.เมือง จ.นครราชสีมา 30000

บทคัดย่อ

การหากฎความสัมพันธ์นั้นเป็นการหาความสัมพันธ์จากข้อมูลขนาดใหญ่ เพื่อให้ได้รูปแบบความสัมพันธ์ของข้อมูลและนำไปใช้ในการวางแผนการตลาด แต่ในการหากฎความสัมพันธ์ด้วยวิธีการในปัจจุบันมีการใช้ระยะเวลาในการประมวลผลข้อมูลค่อนข้างมาก เพราะกฎที่ได้จากการวิเคราะห์ความสัมพันธ์นั้นมีจำนวนมากและบางกฎที่ได้มานั้นไม่มีประโยชน์ ในงานวิจัยนี้จึงได้เสนอการค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ ซึ่งเป็นวิธีการที่เพิ่มเงื่อนไขข้อบังคับเข้าไปในการวิเคราะห์หากฎความสัมพันธ์ เพื่อให้ผู้ใช้สามารถระบุกฎที่ปรากฏเฉพาะไอเท็มที่ต้องการและยังสามารถกำหนดความยาวของกฎได้ จากการทดลองพบว่าถ้าผู้ใช้มีการกำหนดเงื่อนไขที่เฉพาะเจาะจง เช่น เงื่อนไขและ จะทำให้จำนวนกฎและเวลาที่ใช้ในการประมวลผลลดลงถึง 96% เมื่อเทียบกับแบบไม่กำหนดเงื่อนไข

คำสำคัญ: การทำเหมืองข้อมูล, กฎความสัมพันธ์, โปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ

ABSTRACT

The discovery of association rules is a search for relationships from large data to obtain patterns or models that are useful for marketing planning. But current techniques in association mining take much time. Moreover, the discovered rules are normally abundant, redundant, and some of them are useless. Therefore, this research proposes a constraint-based technique to scope the search space and also to reduce the search time by allowing user to specify the items of interest and length of the association rules. From experimental results we found that with specific constraint such as the condition “and”, the running time

and number of discovered rules can be reduced up to 96% when compared to mining with no constraint.

KEYWORDS: Data mining, Association Rule Mining, Constraint Logic Programming

1. บทนำ

ปัจจุบันการค้นหารูปแบบของข้อมูลเพื่อช่วยในการสนับสนุนการตัดสินใจ หรือการค้นหาความสัมพันธ์ในข้อมูลเพื่อปรับมูลค่าในคลังสินค้ามีผู้นิยมเป็นจำนวนมาก แต่การค้นหาความรู้จากข้อมูลนั้นค่อนข้างยากและซับซ้อนเนื่องจากข้อมูลที่นำมาหารูปแบบนั้นมักจะมีจำนวนมาก [1] จึงต้องใช้เวลาเตรียมข้อมูล และใช้เวลาในการค้นหาความรู้เป็นเวลานานโดยในการค้นหาความสัมพันธ์ของข้อมูลนั้นจะทำให้ได้รับความรู้หรือกฎความสัมพันธ์เป็นจำนวนมาก ซึ่งจะยุ่งยากต่อการนำไปใช้เพราะจะต้องมาคัดแยกกฎที่เราต้องการนำไปใช้ออกจากกฎทั้งหมดซึ่งในขั้นตอนคัดแยกอาจจะเกิดข้อผิดพลาดทำให้เสียกฎที่ต้องการไป

ในงานวิจัยนี้จึงมุ่งเน้นพัฒนากระบวนการในการออกแบบอัลกอริทึมและพัฒนาโปรแกรม เพื่อหากฎความสัมพันธ์จากข้อมูลตามที่ผู้ใช้กำหนด ซึ่งจะทำให้ผู้ใช้ได้รับกฎที่มีความจำเป็นต่อผู้ใช้และลดจำนวนของกฎที่ไม่จำเป็น ผู้ใช้สามารถกำหนดเงื่อนต่าง ๆ เพื่อให้ได้กฎตามที่ผู้ใช้ต้องการ เช่น การกำหนดสมาชิกที่ผู้ใช้ต้องการ การกำหนดสมาชิกของกฎที่ผู้ใช้ไม่ต้องการ การกำหนดขนาดของกฎ การกำหนดเป้าหมายของกฎ เป็นต้น ซึ่งปัจจัยที่ทำให้เวลาในการค้นพบและจำนวนกฎที่ลดลงก็คือ การใช้เงื่อนไขเพื่อลดขอบเขตการค้นหากฎที่จะเป็นคำตอบที่ผู้ใช้ต้องการ กระบวนการที่ใช้ค้นหากฎความสัมพันธ์ในงานวิจัยนี้ได้ใช้เทคนิคการทำเหมืองข้อมูลประเภทค้นหาความสัมพันธ์ของข้อมูล โดยใช้อัลกอริทึมที่มีความนิยมสูงคืออัลกอริทึม Apriori [2] โดยนำมาผสมผสานกับการใช้เงื่อนไขและในขั้นตอนการพัฒนาโปรแกรมได้ใช้วิธีการเขียนแบบเชิงตรรกะด้วยเงื่อนไขบังคับ เนื่องจากเป็นเทคนิคการเขียนโปรแกรมที่สามารถระบุเงื่อนไขบังคับ เพื่อลดขนาดของขอบเขตการค้นหาคำตอบ และได้พัฒนา 3 โปรแกรมเพื่อทดสอบประสิทธิภาพของแต่ละโปรแกรมคือ โปรแกรม Apriori โปรแกรม ACIF (Association Rule Discovery With Constraints In Frequent Itemset Mining) และ โปรแกรม ACAF (Association Rule Discovery With Constraints After Itemset Mining) ซึ่งโปรแกรม ACIF เป็นโปรแกรมที่มีการทำงานเหมือนอัลกอริทึม Apriori แต่จะมีการใช้เงื่อนไขในระหว่างการค้นหาไอเท็มเซตที่ปรากฏบ่อย ส่วนโปรแกรม ACAF เป็นโปรแกรมที่มีการทำงานเหมือนอัลกอริทึม Apriori แต่จะมีการใช้เงื่อนไขบังคับหลังจากการค้นหาไอเท็มเซตปรากฏบ่อยทั้งหมด

2. ที่มาและทฤษฎีที่เกี่ยวข้อง

ในการหากฎความสัมพันธ์จากข้อมูลนั้น มีงานวิจัยหลายงานที่มุ่งเน้นการวิจัยเพื่อเพิ่มประสิทธิภาพในการค้นหากฎความสัมพันธ์ เช่น การลดเวลาในการประมวลผล การลดจำนวนกฎที่ซับซ้อน การลดทรัพยากรในการประมวลผล เป็นต้น จากการศึกษาได้สรุปงานวิจัยที่เกี่ยวข้องได้ดังต่อไปนี้

Srikant, R., Vu, Q. และ Agrawal, R. [3] ได้นำเสนอแนวคิดการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ของข้อมูลโดยใช้เงื่อนไขบังคับ โดยใช้หลักการของการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ (Association Rule) และได้ใช้เทคนิคเงื่อนไขบังคับ (Constraints) เพื่อให้ผู้ใช้ (User) สามารถระบุได้ว่ากฎที่มีความสัมพันธ์หรือรูปแบบของข้อมูล (Rule) ที่ได้มานั้นจะต้องประกอบด้วยไอเท็ม (Item) ที่ผู้ใช้ระบุด้วย เพื่อให้ได้กฎตามที่ผู้ใช้ต้องการ ทั้งนี้ในการใช้เงื่อนไขบังคับเข้ามาในการทำเหมืองข้อมูลแบบค้นหากฎความสัมพันธ์ สามารถช่วยลดโครงสร้างแลตทิซ (Lattice Structure) การที่โครงสร้างลดลงเนื่องมาจากเทคนิคการพรุน (Prunning) และเมื่อโครงสร้างลดลง ผลที่ตามมาก็คือทรัพยากร (Memory) ที่ใช้ในการประมวลผลและระยะเวลาในการทำงานของการทำเหมืองข้อมูลแบบค้นหากฎความสัมพันธ์ก็จะลดลงเช่นกัน แต่งานวิจัยนี้ยังเป็นเพียงแค่แนวคิด โดยยังไม่มีการดำเนินการจริง ต่อมา Gallo, A., Esposito, R., Meo, R. และ Botta, M. [4] ได้นำแนวคิดของ Srikant, R., Vu, Q. และ Agrawal, R. [3] ไปประยุกต์ใช้และได้เสนอเกี่ยวกับการเพิ่มประสิทธิภาพของกฎความสัมพันธ์โดยใช้เงื่อนไขบังคับ ในงานวิจัยนี้เขาได้พบปัญหาว่าการค้นหากฎความสัมพันธ์จากฐานข้อมูลจะทำให้พบไอเท็มที่ปรากฏบ่อยที่มีลักษณะคล้ายกันอยู่มาก จึงทำให้ใช้เวลาในการประมวลผลและใช้ทรัพยากรมาก ทีมงานนี้ได้เสนออัลกอริทึมซึ่งช่วยในการสอบถามข้อมูล (Query) [5, 6] ในฐานข้อมูลเพื่อให้ได้กฎความสัมพันธ์ตามที่ผู้ใช้ต้องการและช่วยลดเวลาในการค้นหาต่อจากนั้น Trifonov, T. และ Georgieva, T. [7] ได้นำแนวคิดการเพิ่มประสิทธิภาพของกฎความสัมพันธ์โดยใช้เงื่อนไขบังคับไปประยุกต์ใช้จริงกับข้อมูลเสียงของระฆัง โดยเขาได้นำเทคนิคการทำเหมืองข้อมูลเพื่อค้นหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับมาสร้างโปรแกรมด้วยภาษาจาวาและภาษาสอบถามข้อมูล SQL เพื่อง่าย สะดวกและรวดเร็วต่อคนที่ต้องการค้นหากฎความสัมพันธ์และผู้ใช้ไม่จำเป็นต้องมีความรู้ภาษาสอบถามข้อมูล แต่การหาความกฎสัมพันธ์ของงานวิจัยนี้ได้เฉพาะข้อมูลของเสียงระฆัง

จากการศึกษางานวิจัยที่เกี่ยวข้องพบว่าในงานวิจัยที่ผ่านมา เป็นเพียงการนำแนวคิดการเพิ่มประสิทธิภาพของกฎความสัมพันธ์โดยใช้เงื่อนไขบังคับ มาพัฒนาด้วยภาษาสอบถามข้อมูล SQL ทำให้ใช้เวลาและทรัพยากรมากกว่าการพัฒนาโปรแกรมด้วยวิธีการเขียนเชิงตรรกะด้วยเงื่อนไขบังคับ [8] นอกจากนี้งานวิจัยที่ผ่านมายังไม่สามารถกำหนดขนาดของกฎความสัมพันธ์ที่ค้นพบได้

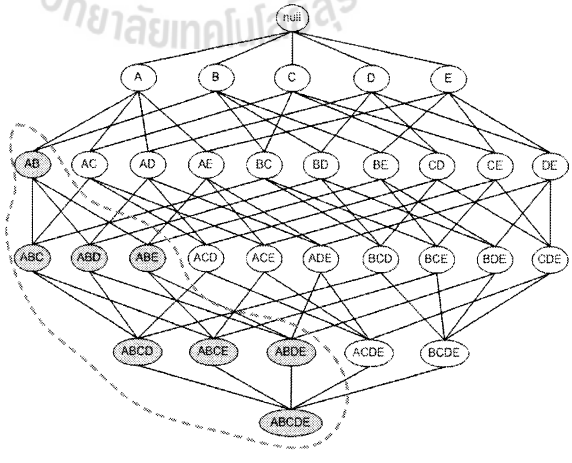
ดังนั้นงานวิจัยนี้จึงได้พัฒนาโปรแกรมหากฎความสัมพันธ์โดยใช้ภาษาโปรล็อกและมีรูปแบบการเขียนแบบใช้กฎเงื่อนไขข้อบังคับ เพื่อให้ผู้ใช้สามารถกำหนดขนาดของกฎ สมาชิกของกฎและเป้าหมายของกฎได้

3. วิธีดำเนินการวิจัย

งานวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาการทำเหมืองข้อมูลเพื่อลดปัญหา การใช้เวลาในการประมวลผลนานและกฎความสัมพันธ์ที่ได้รับจากการประมวลผลมีจำนวนมากได้ โดยโปรแกรมที่พัฒนาอนุญาตให้ผู้ใช้สามารถระบุส่วนประกอบและความยาวของกฎได้ และพัฒนาโปรแกรมด้วยภาษาโปรล็อกและได้ใช้รูปแบบการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ นอกจากนี้ได้นำอัลกอริทึม Apriori มาช่วยในการหารูปแบบของกฎความสัมพันธ์ ซึ่งในหัวข้อนี้จะนำเสนอถึง การทำเหมืองข้อมูลเพื่อค้นหาความสัมพันธ์ อัลกอริทึม Apriori โปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ การออกแบบอัลกอริทึม ACIF และ ACAF และการสอบถามเพื่อหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับ

3.1 อัลกอริทึมเอไพรออริ (Apriori Algorithm)

ในปี ค.ศ.1994 Rakesh Agrawal และ Ramakrishnan Srikant [2] ได้ปรับปรุงอัลกอริทึม AIS (Agrawal Imielinski Swami) ให้ทำงานได้เร็วขึ้นและได้เรียกอัลกอริทึมที่พัฒนาขึ้นใหม่นี้ว่า อัลกอริทึม Apriori โดยใช้เทคนิค Support-based Pruning เพื่อช่วยตัดหรือลดจำนวน Candidate Itemsets แทนที่จะแจกแจงทั้งหมด ดังรูปที่ 1



รูปที่ 1 โครงสร้างแลตทิซที่มีการใช้เทคนิคการพรุน

จากรูปจะเห็นได้ว่ามีการทำเส้นประตั้งแต่ไอเท็มเซต {A, B} ที่มีสองไอเท็ม จนมาถึงไอเท็มเซต {A, B, C, D, E} ที่มีห้าไอเท็มซึ่งมีความหมายว่า ไม่สนใจไอเท็มที่มี {A, B} เป็นซับเซต เนื่องจากค่า Support ของ A และ B ไม่ถึงเกณฑ์ Minimum Support เทคนิคนี้จึงได้ตัดเซตที่มี A และ B เป็นสมาชิกออกเพื่อช่วยลดเวลาในการสร้าง Candidate Itemsets และการทำงานของอัลกอริทึม Apriori มีการทำงานเป็นแบบลูป

3.2 โปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ (Constraint logic programming: CLP)

เป็นการเขียนโปรแกรมเหมือนโปรแกรมเชิงตรรกะปกติทั่วไปแต่ได้เพิ่มเทคนิคโดยการใส่เงื่อนไขบังคับเข้าไป โปรแกรมเชิงตรรกะปกติทั่วไปจะใช้วิธีแบบวนซ้ำด้วย Recursion [8] แต่ใน CLP จะมี Special Predicate ซึ่งเป็น Built-in อยู่ภายในซึ่งจะทำให้การเขียนโปรแกรมสะดวกยิ่งขึ้น เช่น เพรดิเคต foreach เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อดึงค่าที่อยู่ในลิสต์ออกมาทีละตัว และจะหยุดทำงานเมื่อลิสต์นั้นเป็นลิสต์ว่าง เพรดิเคต count เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อนับค่า และเพรดิเคต fromto เป็นเพรดิเคตที่ทำหน้าที่วนลูปเพื่อดึงค่าออกจากลิสต์หรือเพิ่มค่าเข้าไปในลิสต์ โดยเพรดิเคตนี้ถ้าใช้ร่วมกับ foreach จะทำหน้าที่เป็นการเพิ่มค่าเข้าไปในลิสต์ เป็นต้น

นอกจากนี้การเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับยังเหมาะกับการแก้ปัญหาสมการทางคณิตศาสตร์ [9] สมมติว่าต้องการหาค่าของ A และ B จากสมการ 2AB = 20 โดยค่าของ A และ B มีค่า 1 ถึง 5 จากนั้นทำการประมวลผล ซึ่งจะได้ผลลัพธ์เป็นสองคำตอบคือ A = 2 และ B = 5 และคำตอบที่สองจะได้ A = 5 และ B = 2 โดยการหาคำตอบในลักษณะนี้ ได้ใช้เทคนิคการค้นหาคำตอบที่เรียกว่าเทคนิคการย้อนกลับ (Backtrack)

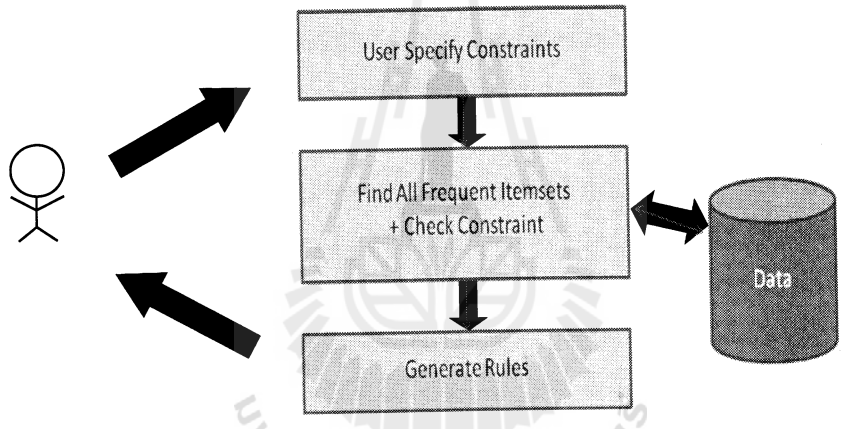
3.3 การออกแบบอัลกอริทึม ACIF และ ACAF

แนวคิดหลักของงานวิจัยนี้คือ การปรับปรุงกระบวนการค้นหากฎความสัมพันธ์ให้สามารถผนวกเงื่อนไขเงื่อนบังคับที่ผู้ใช้ระบุเข้าเป็นส่วนหนึ่งของกระบวนการเพื่อลดขอบเขตการค้นหาและเพื่อให้ผลลัพธ์ที่ตรงกับความต้องการของผู้ใช้ การผนวกเงื่อนไขบังคับนี้ผู้วิจัยได้ออกแบบไว้ 2 แนวทาง คือ ใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย และใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย ซึ่งมีรายละเอียดดังนี้

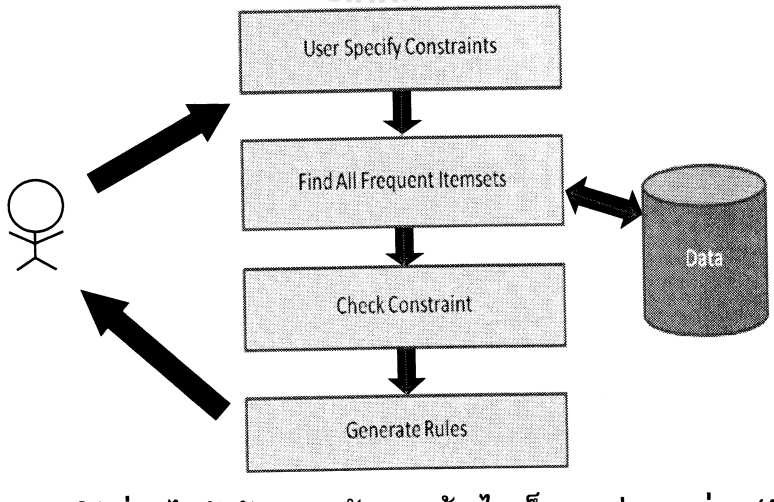
3.3.1 ใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย ACIF (Association Rule Discovery With Constraints In Frequent Itemset Mining)

เป็นรูปแบบการหากฎความสัมพันธ์ตามที่ผู้ใช้งานระบุไอเท็มที่ต้องการหรือไม่ต้องการและขนาดของกฎได้ โดยจะมีการตรวจสอบเงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย และมีกรอบแนวคิดการหากฎความสัมพันธ์ ดังรูปที่ 2 ซึ่งจะประกอบไปด้วย 3 ส่วน คือ

1) ผู้ใช้ (User) ซึ่งผู้ใช้มีหน้าที่ระบุค่าสนับสนุนขั้นต่ำ อย่างเช่นถ้าผู้ใช้ต้องการหากฎความสัมพันธ์ที่มีความถี่เท่ากับสอง หมายถึงกฎที่พบในทรานแซคชันสองทรานแซคชันจากทรานแซคชันทั้งหมด ผู้ใช้ก็จะต้องระบุค่าสนับสนุนขั้นต่ำเท่ากับสอง และผู้ใช้จะต้องระบุค่าความเชื่อมั่นขั้นต่ำและถ้าหากผู้ใช้ต้องการเพียงแค่บางกฎความสัมพันธ์ที่ประกอบไปด้วยไอเท็มที่สนใจผู้ใช้ก็จะต้องระบุเงื่อนไขด้วย



รูปที่ 2 การใช้เงื่อนไขบังคับขณะมีการสร้างไอเท็มเซตปรากฏบ่อย (ACIF)



รูปที่ 3 การใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย (ACAF)

2) หาไอเท็มเซตปรากฏบ่อยทั้งหมด (Find All Frequent Itemset) ทั้งหมด ได้แก่ไอเท็มเซตที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ และใช้เงื่อนไขบังคับตามที่ผู้ใช้ได้ระบุมา (Check Constraints) เพื่อให้ได้ไอเท็มเซตตามที่ผู้ใช้ได้ระบุ

3) เป็นการสร้างกฎความสัมพันธ์จากไอเท็มเซตปรากฏบ่อยจากการใช้เงื่อนไขบังคับ (Generate Rules) ซึ่งการสร้างกฎความสัมพันธ์นั้น กฎจะต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

3.3.2 ใช้เงื่อนไขบังคับภายหลังการสร้างไอเท็มเซตปรากฏบ่อย ACAF (Association Rule Discovery with Constraints after Itemset Mining)

รูปแบบนี้ได้ใช้แนวคิดของอัลกอริทึม Apriori เป็นหลักและได้เพิ่มส่วนของเงื่อนไขบังคับเข้าไปในอัลกอริทึม โดยมีกรอบแนวคิดการหากฎความสัมพันธ์ ดังรูปที่ 3 จากกรอบแนวคิดการหากฎความสัมพันธ์จะประกอบไปด้วย 4 ส่วน คือ

1) ผู้ใช้ (User) ซึ่งผู้ใช้มีหน้าที่ระบุค่าสนับสนุนขั้นต่ำ (Minimum Support) ค่าความเชื่อมั่นขั้นต่ำ (Minimum Confidence) และถ้าหากผู้ใช้ต้องการเพียงแค่บางกฎความสัมพันธ์ที่ประกอบไปด้วยไอเท็มที่สนใจผู้ใช้ก็จะต้องระบุเงื่อนไขด้วย

2) หาไอเท็มเซตปรากฏบ่อยทั้งหมด (Find All Frequent Itemset) ทั้งหมด ได้แก่ไอเท็มเซตที่มีค่ามากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ

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

4) จากขั้นตอนที่ 3 จะได้ไอเท็มเซตปรากฏบ่อยจากการใช้เงื่อนไขบังคับ และในขั้นตอนที่ 4 นี้จะเป็นการสร้างกฎความสัมพันธ์จากไอเท็มเซตปรากฏบ่อย (Generate Rules) ซึ่งการสร้างกฎความสัมพันธ์นั้น กฎจะต้องมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

3.4 การสอบถามเพื่อหากฎความสัมพันธ์ด้วยเงื่อนไขบังคับ

การสอบถามเพื่อหากฎความสัมพันธ์จะต้องเตรียมข้อมูลให้อยู่ในรูปแบบของภาษาโปรล็อก ดังรูปที่ 4 และในการสอบถามเพื่อหาคำตอบของโปรแกรมจะอยู่ในรูปแบบการสอบถามเชิงตรรกะ โดยจะมีการสอบถามในรูปแบบต่างๆ ดังนี้

- การสอบถามแบบไม่มีเงื่อนไขพิเศษ คือ มีการทำงานเหมือนอัลกอริทึม Apriori ที่มีการกำหนดเฉพาะค่าสนับสนุนต่ำสุดและค่าเชื่อมั่นต่ำสุด

- การสอบถามแบบใช้เงื่อนไขกำหนดขนาดของกฎ คือ การสอบถามเพื่อให้ได้ขนาดสมาชิกของกฎความสัมพันธ์ตามผู้ใช้ต้องการ เช่น ถ้าผู้ใช้ต้องการกฎที่มีสมาชิกมากกว่า 2 ไอเท็ม กฎที่ได้รับก็จะมีขนาดมากกว่า 2 ไอเท็ม

- การสอบถามแบบกำหนดสมาชิกที่ต้องการ เป็นการสอบถามเพื่อให้ได้สมาชิกตามที่ผู้ใช้กำหนดโดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้งและจะใช้เงื่อนไขได้ในครั้งแรก

```
data([
[beer],[cannedmeat],[cannedveg],[confectionery],[dairy],[fish],
[freshmeat],[frozenmeal],[fruitveg],[softdrink],[wine]]
-
    [
        [beer,cannedmeat,confectionery,wine],
        [softdrink,fruitveg,wine,confectionery],
        [dairy,cannedmeat,fish],
        [dairy,freshmeat],
        [wine,softdrink,fruitveg,confectionery],
        [frozenmeal,confectionery,fish],
        [confectionery,wine],
        [fruitveg,frozenmeal,dairy,freshmeat],
        [softdrink,cannedmeat,dairy,cannedveg],
        [cannedveg,beer],
        [softdrink,cannedmeat,beer,cannedveg],
        [softdrink,dairy,beer]
    ]
).
```

Item

Transaction

รูปที่ 4 รูปแบบของข้อมูลที่จะนำมาหากฎความสัมพันธ์

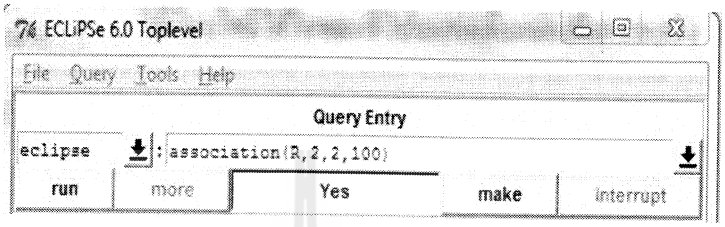
- การสอบถามแบบกำหนดสมาชิกที่ไม่ต้องการ เป็นการสอบถามเพื่อให้ได้กฎที่ไม่มีสมาชิกที่เราไม่ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้งและจะใช้เงื่อนไขได้ในกล่องข้อความครั้งที่สอง

- การสอบถามแบบใช้เงื่อนไขหรือ (OR) คือ การสอบถามเพื่อให้ได้กฎความสัมพันธ์ที่ประกอบด้วยสมาชิกบางตัวตามที่ผู้ใช้ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้ง

- การสอบถามแบบใช้เงื่อนไขและ (AND) คือ การสอบถามเพื่อให้ได้กฎความสัมพันธ์ที่ประกอบด้วยสมาชิกทุกตัวตามที่ผู้ใช้ต้องการ โดยการใช้เงื่อนไขรูปแบบนี้จะคล้ายกับการสอบถามแบบใช้เงื่อนไขหรือ (OR) ใช้ในช่วงที่โปรแกรมขึ้นกล่องข้อความมาสามครั้ง

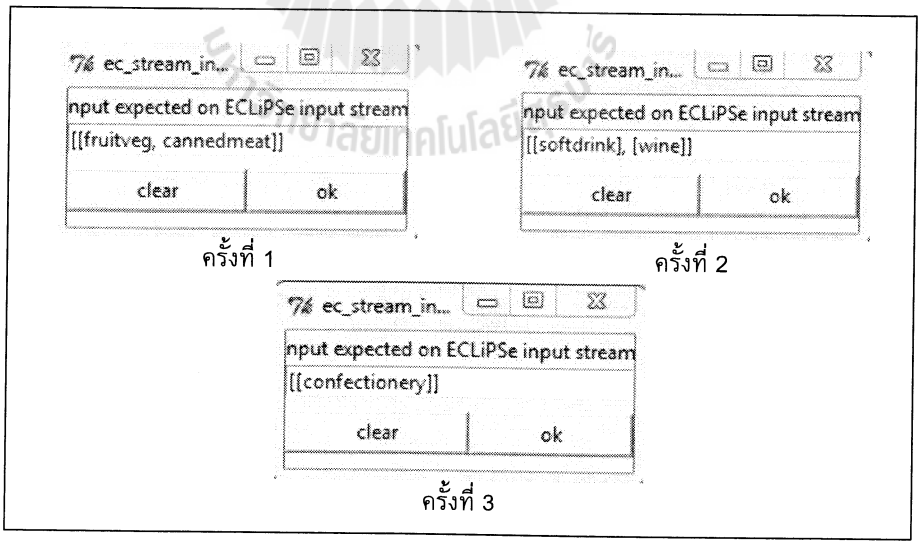
- การสอบถามแบบใช้เงื่อนไขกำหนดเป้าหมาย คือ การสอบถามโดยมีการกำหนดสมาชิกหลัง then เพื่อให้ได้กฎความสัมพันธ์ตามผู้ใช้ต้องการ และสามารถใช้เงื่อนไขนี้ได้ในขณะที่โปรแกรมขึ้นกล่องข้อความครั้งที่สาม

ตัวอย่างการสอบถามเพื่อหากฎความสัมพันธ์ที่มีขนาดมากกว่า 2 ไอเท็ม ประกอบด้วยสมาชิกกฎ คือ softdrink และ wine และไม่มี fruitveg หรือ cannedmeat เป็นสมาชิกของกฎ โดยมีเป้าหมายของกฎ คือ confectionery โดยจะมีการสอบถามดังรูปที่ 5



รูปที่ 5 ตัวอย่างการสอบถามเพื่อหากฎความสัมพันธ์

จากรูปที่ 5 เพรดิเคต association คือ เพรดิเคตสอบถามเพื่อหากฎความสัมพันธ์ หมายเลข 2 ตัวแรก หมายถึงกำหนดขนาดของกฎความสัมพันธ์มีขนาดมากกว่า 2 ไอเท็ม หมายเลข 2 ตัวถัดมาเป็นการกำหนดค่าสนับสนุนต่ำสุด และหมายเลข 100 คือค่าความเชื่อมั่นต่ำสุด หลังจากการสอบถามโปรแกรมจะขึ้นกล่องข้อความมา 3 ครั้ง ครั้งแรกกำหนดสมาชิกที่ต้องการใส่ [[softdrink], [wine]] ครั้งที่สองกำหนดสมาชิกที่ไม่ต้องการใส่ [[fruitveg, cannedmeat]] และครั้งที่สามกำหนดเป้าหมายใส่ [[confectionery]] (ดังรูปที่ 6)



รูปที่ 6 ตัวอย่างการใส่เงื่อนไขในกล่องข้อความสามครั้ง

หลังจากใส่เงื่อนไขครบทั้ง 3 กล่องข้อความและใช้ข้อมูลในการสอบถามจากรูปที่ 4 จะได้กฎความสัมพันธ์หนึ่งกฎ คือ

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

กฎความสัมพันธ์จะอยู่ในรูปแบบ If ... then ... หมายเลข 2 หลัง wine หมายความว่า พบไอเท็ม softdrink และ wine 2 ทรานแซคชันจากทรานแซคชันทั้งหมด และหมายเลข 2 หลัง confectionery หมายความว่า พบไอเท็ม softdrink, wine และ confectionery 2 ทรานแซคชันจากทรานแซคชันทั้งหมด

4. เปรียบเทียบผลการวิจัยและอภิปรายผล

4.1 เปรียบเทียบผลการวิจัย

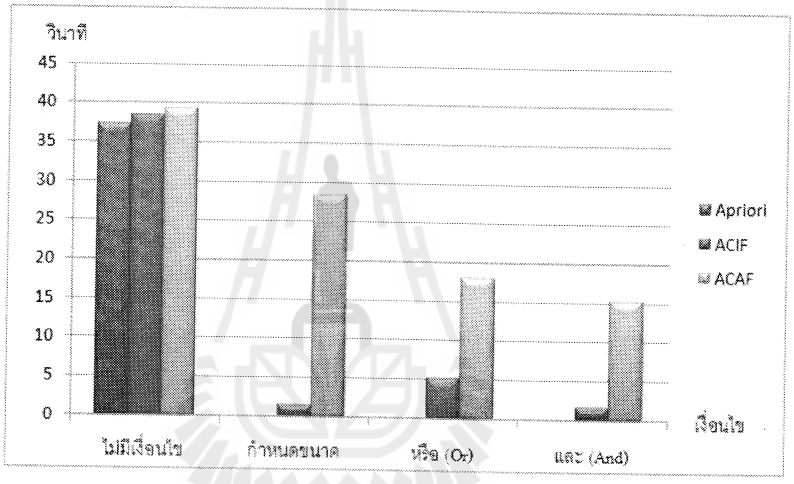
การทดสอบระบบใช้เครื่องคอมพิวเตอร์มีหน่วยประมวลผลกลาง Intel Core i7 หน่วยความจำหลัก 4 GB และข้อมูลที่ในการทดสอบใช้ข้อมูลมาตรฐาน ซึ่งเป็นข้อมูลเกี่ยวกับหมากรุก (Chess) [10] โดยมีข้อมูลทั้งหมด 3,196 แถว ประกอบด้วยแอททริบิวท์จำนวน 37 แอททริบิวท์ และทดสอบกับสามโปรแกรม คือ โปรแกรม Apriori โปรแกรม ACIF และโปรแกรม ACAF การเปรียบเทียบจะใช้เงื่อนไขในรูปแบบต่าง ๆ เพื่อหากฎความสัมพันธ์และจะใช้ตัวชี้วัด คือ เวลาในการประมวลผล (หน่วยเป็นวินาที) และจำนวนกฎความสัมพันธ์ โดยจะแสดงการเปรียบเทียบเวลาในตารางที่ 1 (รูปที่ 7) และแสดงการเปรียบเทียบจำนวนกฎความสัมพันธ์ในตารางที่ 2 (รูปที่ 8)

ตารางที่ 1 เวลาในการประมวลผลของโปรแกรม Apriori, ACIF และ ACAF

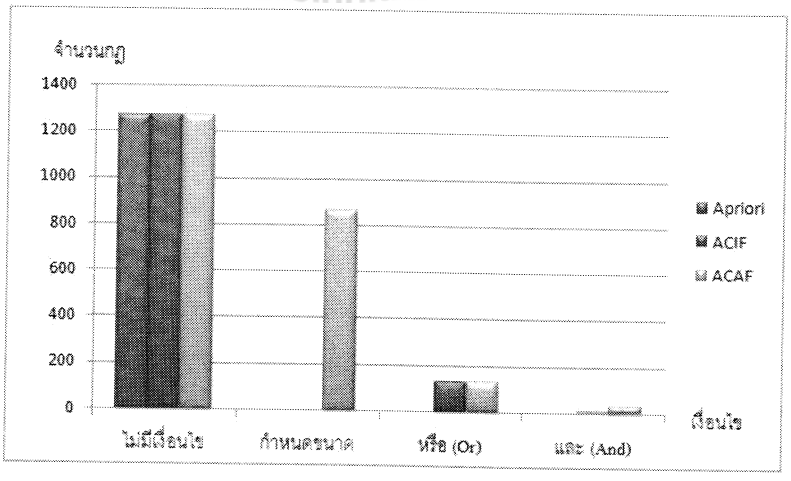
โปรแกรม	การใช้เงื่อนไขรูปแบบต่าง ๆ เพื่อหากฎความสัมพันธ์						
	ไม่มีเงื่อนไข (วินาที)	กำหนดขนาดของกฎ		ระบุไอเท็มด้วยเงื่อนไขหรือ (Or)		ระบุไอเท็มด้วยเงื่อนไขและ (And)	
	เวลาที่ใช้	เวลาที่ใช้	เวลาที่ลดลง	เวลาที่ใช้	เวลาที่ลดลง	เวลาที่ใช้	เวลาที่ลดลง
Apriori	37.29 (s)	-	-	-	-	-	-
ACIF	38.48 (s)	1.45	96%	5.09	87%	1.56	96%
ACAF	39.35 (s)	28.34	28%	18.05	55%	15.22	61%

ตารางที่ 2 เวลาในการประมวลผลของโปรแกรม 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 7 แผนภูมิการเปรียบเทียบเวลาในการประมวลผลของโปรแกรม Apriori, ACIF และ ACAF



รูปที่ 8 การเปรียบเทียบจำนวนกฎความสัมพันธ์ของโปรแกรม Apriori ACIF และ ACAF

จากตารางที่ 1 และ 2 ในการใช้เงื่อนไขในรูปแบบและสัญลักษณ์ต่าง ๆ มีคำอธิบายดังนี้

1) ไม่มีเงื่อนไข เป็นการกำหนดให้โปรแกรมไม่ใช้เงื่อนไขใดๆ นอกเหนือจากเงื่อนไขของอัลกอริทึม Apriori ที่ให้ผู้ใช้ระบุค่า 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

5) สัญลักษณ์ – หมายถึง ไม่สามารถทำได้ ส่วนเวลาที่ใช้ หมายถึง เวลาที่ใช้ในการประมวลผล และเวลาที่ลดลง หมายถึง เวลาที่ลดลงเมื่อเทียบกับการใช้เงื่อนไขในรูปแบบไม่มีเงื่อนไข โดยแสดงเป็นร้อยละ

4.2 อภิปรายผล

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

1) รูปแบบไม่มีเงื่อนไข หมายถึง การค้นหากฎความสัมพันธ์ใช้เฉพาะเงื่อนไข Minimum Support และ Minimum Confidence เท่านั้นรูปแบบนี้โปรแกรมทั้งสามจะใช้เวลาที่ไม่ค่อยจะต่างกันมากทั้งนี้เพราะทั้งสามโปรแกรมมีพื้นฐานการทำงานที่เหมือนกันคือทำงานเหมือนอัลกอริทึม Apriori อยู่แล้ว ซึ่งถ้าไม่ใช้เงื่อนไขในการหาความสัมพันธ์ของข้อมูลในการทำงานของทั้งสามโปรแกรมนี้ก็แทบจะไม่แตกต่างกันเลยและจำนวนกฎที่ได้รับก็มีจำนวนที่เท่ากันด้วย

2) รูปแบบกำหนดขนาดของกฎ การประมวลผลในรูปแบบนี้พบว่าโปรแกรมทั้งสามใช้เวลาที่แตกต่างกันมาก โปรแกรม Apriori ไม่มีฟังชันก์การกำหนดขนาดของกฎจึงไม่สามารถหากฎที่มีเฉพาะกฎที่กำหนดขนาดได้ ส่วนโปรแกรม ACIF นั้นได้ใช้เวลาเพียง 1.45 วินาทีหรือ 4% ของรูปแบบไม่มีเงื่อนไข แต่ไม่พบกฎที่มีขนาดมากกว่า 3 ไอเท็มเพราะว่าโปรแกรมไม่สามารถหากฎความสัมพันธ์ที่มีขนาดมากกว่า 3 ได้เนื่องจากในการใช้เงื่อนไขระหว่างการหาไอเท็มเซตปรากฏบ่อยนั้น ได้มีการเช็คขนาดแคนดิเดตไอเท็มเซตว่าเซตที่จะเป็นไอเท็มเซตปรากฏบ่อยนั้นจะต้องมีสมาชิกมากกว่า 3 ไอเท็ม ตัวอย่างการเช็คขนาดมีดังนี้ ในการหาไอเท็มเซตรอบแรกจะไม่มีการเช็คขนาดแต่พอถึงรอบสองจะมีการเช็คขนาด ซึ่งรอบสองนั้นจะประกอบด้วยแคนดิเดตไอเท็มเซตที่มีสมาชิก 2 ตัวและจะถูกนำไปเช็คกับเงื่อนไขว่าสมาชิกของแคนดิเดตจะต้องมากกว่า 3 ไอเท็ม ซึ่งในรอบนี้ไม่มีเซตไหนผ่านเงื่อนไข จึงไม่มีการสร้างไอเท็มเซตและแคนดิเดตไอเท็มเซตขนาด 3 ไอเท็มขึ้นไปเกิดขึ้นจึงทำให้จำนวนกฎที่ได้รับนั้นเป็นศูนย์ ในส่วนโปรแกรม ACAF นี้จะใช้เวลามากที่สุด

เพราะการใช้เงื่อนไขกำหนดขนาดจะใช้หลังจากหาไอเท็มปรากฏบ่อยบ่อยทั้งหมดแล้วจะได้ไอเท็มเซตทุกขนาด จึงทำให้สามารถเช็คเงื่อนไขได้และจะใช้เวลามากกว่าโปรแกรมอื่นแต่จะได้กฎที่ต้องการครบถ้วน

3) รูปแบบระบุไอเท็มด้วยเงื่อนไขหรือ (Or) โปรแกรม Apriori ไม่สามารถรูปแบบนี้ได้ เพราะว่าในโปรแกรมไม่สามารถใช้เงื่อนไขนี้ได้ จึงไม่สามารถหากฎที่มีเฉพาะกฎที่กำหนดเงื่อนไขไว้ได้ ส่วนโปรแกรม ACIF จะใช้เวลาค่อนข้างน้อยกว่า โปรแกรม ACAF เพราะโปรแกรม ACIF มีการใช้เงื่อนไขในระหว่างการหาไอเท็มเซตปรากฏบ่อยจึงทำให้การประมวลผลใช้เวลาน้อยกว่าการประมวลผลหลังจากการหาไอเท็มปรากฏบ่อยด้วยโปรแกรม ACAF แต่กฎที่ได้ก็มีจำนวนที่เท่ากันและเหมือนกัน

4) รูปแบบระบุไอเท็มด้วยเงื่อนไขและ (And) โปรแกรม Apriori ไม่สามารถใช้เงื่อนไขนี้ได้ ส่วนโปรแกรม ACIF จะใช้เวลาน้อยที่สุดจากทั้ง 3 โปรแกรมและจะได้ผลลัพธ์เป็นกฎความสัมพันธ์เพียง 2 กฎ ซึ่งกฎที่ได้รับนั้นจะได้รับกฎไม่ครบตามเงื่อนไขและกฎที่ได้รับมีขนาด 2 ไอเท็ม เป็นเพราะว่าในการใช้รูปแบบและจะเป็นการเลือกไอเท็มเซตที่มีสมาชิกประกอบด้วยเงื่อนไขที่ผู้ใช้กำหนดเช่นถ้ามีแคนดิเดตไอเท็มเซตขนาด 2 ไอเท็มอยู่ 5 ชุด คือ {a, b}, {a, c}, {a, d}, {b, e}, {a, f} สมมติผู้ใช้กำหนดเงื่อนไขที่ต้องการเป็น a และ b ไอเท็มเซตที่ผ่านเงื่อนไขก็คือ {a, b} เพียงชุดเดียว ซึ่งในการสร้างแคนดิเดตจำนวน 3 ไอเท็มไม่สามารถสร้างได้เพราะมีไอเท็มเซตที่ผ่านเงื่อนไขมาเพียงชุดเดียวจึงเป็นสาเหตุที่ทำให้ได้รับกฎที่น้อยกว่าความเป็นจริง ซึ่งจะต่างจากโปรแกรม ACAF ที่จะใช้เวลาค่อนข้างมากแต่จะให้กฎที่ครบถ้วนตามเงื่อนไข

5. สรุปผลการวิจัย

การทำเหมืองข้อมูลเพื่อหากฎความสัมพันธ์ในปัจจุบัน จะใช้เวลาค่อนข้างมากและกฎความสัมพันธ์ที่ได้รับ อาจจะมีมากเกินความต้องการของผู้ใช้งาน ทำให้ผู้ใช้งานนำไปประยุกต์ใช้ได้ยาก ดังนั้นงานวิจัยนี้จึงได้นำเสนอการค้นหากฎความสัมพันธ์ด้วยการเขียนโปรแกรมเชิงตรรกะด้วยเงื่อนไขบังคับ ซึ่งเป็นวิธีการเพิ่มเงื่อนไขข้อบังคับตามที่ผู้ใช้ระบุ โดยสามารถระบุไอเท็มและขนาดของกฎได้ เพื่อให้ได้รับกฎความสัมพันธ์ตามที่ผู้ใช้ต้องการ ในการทดลองได้ทดสอบประสิทธิภาพด้วย 3 โปรแกรม คือ โปรแกรม Apriori โปรแกรม ACAF และโปรแกรม ACIF เพื่อเปรียบเทียบจำนวนกฎความสัมพันธ์ที่ได้รับและเวลาที่ใช้ในการประมวลผล ซึ่งผลการทดลองแสดงให้เห็นว่า ถ้าไม่ใช้เงื่อนไขเพิ่มเติมในการค้นหาความสัมพันธ์ทั้ง 3 โปรแกรมจะให้ผลลัพธ์เป็นกฎความสัมพันธ์ที่มีจำนวนกฎที่เท่ากันและเวลาในการประมวลผลไม่แตกต่างกันมาก แต่ถ้าหากใช้เงื่อนไขกำหนดขนาดของสมาชิกโปรแกรม Apriori จะไม่สามารถกำหนดขนาดของกฎได้และโปรแกรม ACIF จะสามารถกำหนดขนาดได้ที่เงื่อนไขสมาชิกที่ไม่เกิน 2 ไอเท็ม แต่ถ้ากำหนดขนาดมากกว่านั้นโปรแกรมไม่สามารถค้นหากฎได้ ส่วนโปรแกรม ACAF จะให้กฎที่ครบถ้วนตามเงื่อนไขที่กำหนดขนาด แต่ถ้าใช้

เงื่อนไขแบบ "Or" โปรแกรม ACIF จะใช้เวลาในการค้นหากฎน้อยมากและให้กฎที่ครบถ้วน ส่วนโปรแกรม ACAF จะใช้เวลาค่อนข้างมากและจะได้รับกฎที่ครบถ้วนเช่นกัน ส่วนการใช้เงื่อนไขสุดท้ายคือ "And" โปรแกรม ACIF จะใช้เวลาน้อยแต่ให้กฎที่ถูกต้องไม่ครบถ้วน ส่วนโปรแกรม ACAF จะใช้เวลามากกว่าแต่จะให้กฎที่ครบถ้วน จึงสรุปได้ว่าโปรแกรม ACAF ถึงจะใช้เวลามากกว่าโปรแกรม ACIF แต่จะให้กฎความสัมพันธ์ที่ครบถ้วนและถูกต้องมากกว่า

กิตติกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนจากมหาวิทยาลัยเทคโนโลยีสุรนารี

เอกสารอ้างอิง

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

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

[3] Srikant, R., Vu, Q., Agrawal, R. (1997). "Mining association rules with item constraints". **Proceedings of 1997 ACM KDD**. pp. 67 - 73.

[4] Gallo, A., Esposito, R., Meo, R., Botta, M. (2005). "Optimization of Association Rules Extraction Through Exploitation of Context Dependent Constraints", **AI*IA 2005**. pp. 258 - 269.

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

[6] Ng, R.T., Lakshmanan, L.V.S., Han, J., Pang, A. (1998). "Exploratory mining and pruning optimizations of constrained association rules". **Proceedings of 1998 ACM SIGMOD Int. Conf. Management of Data**. pp. 13 - 24.

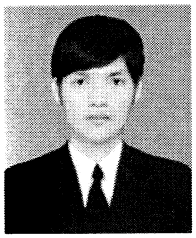
[7] Trifonov, T., Georgieva, T. (2009). "Application for Discovering the Constraint-Based Association Rules in an Archive for Unique Bulgarian Bells". **European Journal of Scientific**. Vol.31. (No.3): pp. 366 - 371.

[8] Kerdprasop, K., Kerdprasop, N. (2006). "Frequent pattern discovery with declarative programming". **Proceedings of the 1st Rajamangala University of Technology Conference. Trang, Thailand**.

[9] Simonis, H. (2008). **Constraint logic programming**. COSYTEC SA.

[10] Goethals, B. (2014). **Frequent Itemset Mining Dataset Repository. [Online].** Available: http://fimi.ua.ac.be/data/

ประวัติผู้เขียนบทความ



ไพชยนต์ คงไชย ปัจจุบันเป็นนักศึกษาปริญญาเอกที่มหาวิทยาลัยเทคโนโลยีสุรนารี สาขาวิศวกรรมคอมพิวเตอร์ สำเร็จการศึกษาในระดับปริญญาตรีและปริญญาโทในสาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี เมื่อปี พ.ศ. 2553 และ 2555 ตามลำดับ งานวิจัยที่มีความสนใจ คือ การทำเหมืองข้อมูลโดยการใช้เงื่อนไขรวมถึงบังคับ การหากฎความสัมพันธ์ การเขียนโปรแกรมเชิงฟังก์ชันและการเขียนโปรแกรมเชิงตรรกะ สถิติและการเรียนรู้ของเครื่อง อีเมล์: Zaguraba_ii@hotmail.com

รองศาสตราจารย์ ดร. นิตยา เกิดประสพ เป็นอาจารย์ประจำสาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี สำเร็จการศึกษาในระดับปริญญาตรีสาขารังสีเทคนิคจากมหาวิทยาลัยมหิดลในปี พ.ศ. 2528 และปริญญาโทสาขาวิทยาการคอมพิวเตอร์จากมหาวิทยาลัยสงขลานครินทร์ในปี พ.ศ. 2534 และปริญญาเอกในสาขาวิทยาการคอมพิวเตอร์จากมหาวิทยาลัย Nova Southeastern ที่สหรัฐอเมริกาในปี พ.ศ. 2542 เป็นสมาชิกสมาคมวิชาชีพ ACM และ IEEE Computer Society งานวิจัยที่มีความสนใจ คือ การค้นหาความรู้ในฐานข้อมูล ปัญญาประดิษฐ์ การเขียนโปรแกรมเชิงตรรกะ และฐานข้อมูลอัจฉริยะ อีเมล์: nittaya@sut.ac.th

รองศาสตราจารย์ ดร. กิตติศักดิ์ เกิดประสพ เป็นหัวหน้าสาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี สำเร็จการศึกษาในระดับปริญญาตรีภาควิชาคณิตศาสตร์จากมหาวิทยาลัยศรีนครินทรวิโรฒในปี พ.ศ. 2529 สำเร็จการศึกษาระดับปริญญาโทสาขาวิทยาการคอมพิวเตอร์จากมหาวิทยาลัยสงขลานครินทร์ในปี พ.ศ. 2534 และปริญญาเอกในสาขาวิทยาการคอมพิวเตอร์จากมหาวิทยาลัย Nova Southeastern ที่สหรัฐอเมริกาในปี พ.ศ. 2542 งานวิจัยที่มีความสนใจ คือ เหมืองข้อมูล ปัญญาประดิษฐ์ การเขียนโปรแกรมเชิงฟังก์ชันและการเขียนโปรแกรมเชิงตรรกะ สถิติและการคำนวณ อีเมล์: kerdpras@sut.ac.th

ภาคผนวก ข

รหัสต้นฉบับของโปรแกรม Top-K-patterns

```
% Program Top-K-patterns
% Created by Phaichayon Kongchai and Nittaya Kerdprasop
%           Data Engineering Research Unit, Suranaree University of Technology, Thailand.
%
% Start the program by calling the top_K_patterns predicate with user-specified K, such as
%                   top_K_patterns(3)
%
%    To specify more constraints, user may use the following predicate
%                   association(R, LengthI,MinSup,Conf,K)
%    where R is the output top-k rule set
%          LengthI is the length of itemset
%          MinSup is minimum support threshold
%          Conf  is the minimu confidence threshold
%          K is the number of top rules, in which top is ranked by confidence-support value.

:- compile("sample_data.txt"). % load file, the file name can be changed.
:- lib(sd).
:- lib(ic).
:- lib(ordset).
:- assert(rules(_)).
:- assert(k(_)).
:- writeln('This is the Mining Association Rule.'),
  writeln('Put the query as   ?- top_K_patterns(K).'),
  writeln('You can also query by ?- association(R,Length,MinSup,Conf,K).').

top_K_patterns(K) :- association(R,0,30,30,K).  % these default constraints can be changed

association(R,LengthI,MinSup,Conf,K) :-
            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=_-_,
            retractall(rules(X)),retractall(k(X)), assert(k(0)), writeln('------All-Rules----'), % fix to 6 loops
      ( 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 ),!
      ), findall(Point,rules([_ - _ - _ - _- Point]),Bag), sort(0,>,Bag,SPoint),
          findall(All,rules(All),AllR), writeln('-------Result-------'),
          topk(SPoint,AllR,K).

topk([],_,_).
topk([Chk|T],Bag,K) :-
          sortRule(Bag,Chk,K),
          topk(T,Bag,K).

sortRule([],_,_).
sortRule([Rule|T],Chk,K) :-
      k(Count),
      Count = K -> ! ;
      [L - R - S - C- P] = Rule,
      (P =:= Chk -> k(Count), CC is Count+1, write(CC),write('. If '),
      write(L),write(' then '),
      write(R), write(' | Sup :'),write(S), write(' | Conf :'),write(C),
      write(' | Score :'),writeln(P),  % Score is average of Supp and Conf
      retractall(k(_)), asserta(k(CC))  ;  !),
      sortRule(T,Chk,K).
```

```
% 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,Support) :-
            (foreach(I,Items), fromto(R,S1,S0,[]), param(X) do
                 (my_subset(X,I) -> S1 = [ I | S0], ! ; S1=S0)
              ),
             length(R,LenItem),
               length(Items,Transctions),
               Support is (LenItem/Transctions)*100, %/
             Support >= 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),
                 confOk(LenItem-LenItemX-MinConf,ItemX,R1),! ; ! %,write('rrrrr'),write(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
confOk(LenItem-LenItemX-MinConf,ItemX,Re) :- Re11 is (LenItem/LenItemX)*100,Re11 >= MinConf ->
%/
                              write('If '),
                              write(ItemX),write(' then '),AvgSupConf is (LenItem+Re11)/2,
                              write(Re),write(' | Score : '),writeln(AvgSupConf),
                              Rule = [ItemX-Re-LenItem-Re11-AvgSupConf], assert(rules(Rule)) ; ! .

% 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).

% ====================== END OF PROGRAM ========================== %
```

ประวัติผู้วิจัย

รองศาสตราจารย์ ดร.นิตยา เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา สหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของกระทรวงวิทยาศาสตร์ฯ โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง "The application of inductive logic programming to support semantic query optimization" หลังสำเร็จการศึกษาได้ปฏิบัติราชการในตำแหน่งอาจารย์ ประจำสาขาคอมพิวเตอร์ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ต่อมาในปีพุทธศักราช 2543 ได้มาปฏิบัติงานในตำแหน่งอาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จนถึงปัจจุบัน งานวิจัยที่ทำในขณะนี้คือการประยุกต์เทคโนโลยีเหมืองข้อมูลกับงานด้านการแพทย์และสาธารณสุข และการพัฒนาเทคนิคเพื่อเพิ่มความสามารถในการจัดการความรู้ของระบบเหมืองข้อมูล

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา สหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของทบวงมหาวิทยาลัย (หรือสำนักงานคณะกรรมการอุดมศึกษาในปัจจุบัน) โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง "Active database rule set reduction by knowledge discovery" หลังสำเร็จการศึกษาได้ปฏิบัติงานในตำแหน่งอาจารย์ ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ปัจจุบันดำรงตำแหน่งหัวหน้าหน่วยวิจัยวิศวกรรมความรู้ เน้นการวิจัยเกี่ยวกับการพัฒนาระบบเหมืองข้อมูลประสิทธิภาพสูง การประยุกต์เหมืองข้อมูลกับงานวิศวกรรมและการศึกษา รวมถึงการวิจัยพื้นฐานเกี่ยวกับเทคนิคการวิเคราะห์ข้อมูลโดยวิธีอัตโนมัติ โดยมีผลงานวิจัยตีพิมพ์ในวารสารวิชาการและเอกสารการประชุมวิชาการจำนวนมากกว่า 80 เรื่องในสาขาฐานข้อมูล การวิเคราะห์ข้อมูลและการค้นหาความรู้