การจัดกลุ่มข้อมูลด้วยเทคนิคราฟเคมีดอยส์แบบขนาน บนหน่วยประมวลผลกลางแบบหลายแกนหลัก

นายวีรศักดิ์ ช่องงูเหลือม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2555

PARALLELIZE ROUGH K-MEDOIDS CLUSTERING ON MULTICORE PROCESSOR

Weerasak Chongnguluam

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2012

การจัดกลุ่มข้อมูลด้วยเทคนิคราฟเคมีดอยส์แบบขนาน
บนหน่วยประมวลผลกลางแบบหลายแกนหลัก

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.นิตยา เกิดประสพ)
ประธานกรรมการ

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(อ. ดร.ธรา อั่งสกุล)
กรรมการ

_____ _____

(ศ. ดร.ชูกิจ ลิมปิจำนงค์) (รศ. ร.อ. ดร.กนต์ธร ชำนิประศาสน์)
รองอธิการบดีฝ่ายวิชาการ คณบดีสำนักวิชาวิศวกรรมศาสตร์

วีรศักดิ์ ช่องงูเหลือม : การจัดกลุ่มข้อมูลด้วยเทคนิคราฟเคมีดอยส์แบบขนาน บนหน่วยประมวลผลกลางแบบหลายแกนหลัก (PARALLELIZE ROUGH K-MEDOIDS CLUSTERING ON MULTICORE PROCESSOR) อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ, 92 หน้า.

หน่วยประมวลผลกลางแบบหลายแกนหลัก (Multi-Core Processors) นั้น ปัจจุบันมีใช้งานกันแพร่หลายทั้งบนเครื่องคอมพิวเตอร์ส่วนบุคคล เครื่องคอมพิวเตอร์แบบพกพา รวมทั้งหน่วยประมวลผลสำหรับโทรศัพท์พกพาทั่วไป เพื่อให้ใช้ประโยชน์สูงสุดกับหน่วยประมวลผลเหล่านี้ อัลกอริทึมเดิมจำเป็นต้องได้รับการออกแบบใหม่ ในงานวิจัยนี้ได้นำเสนอ การประมวลผลแบบขนานสำหรับอัลกอริทึมการจัดกลุ่มที่เรียกว่า ราฟเคมีดอยส์ (Rough K-Medoids) ซึ่งเป็นอัลกอริทึมที่นำเอาทฤษฎีทางด้านราฟเซต มาประยุกต์กับการจัดกลุ่มแบบเค-มีดอยส์ วิธีการนี้ทำให้การจัดข้อมูลในแต่ละกลุ่มแบ่งออกเป็น 2 แบบ คือ แบบจัดให้ข้อมูลอยู่กับกลุ่มใดกลุ่มหนึ่งเท่านั้น (Lower Approximation) และ แบบที่ข้อมูลอยู่กับหลายๆ กลุ่มได้ (Upper Approximation) โดยนำอัลกอริทึมที่ออกแบบมาพัฒนาด้วยวิธีการโปรแกรมเชิงฟังก์ชันด้วยภาษา เออร์แลง (Erlang) ผลการทดลองแสดงให้เห็นว่าความเร็วของอัลกอริทึมแบบขนานสำหรับ ราฟเคมีดอยส์ นั้นดีขึ้น เมื่อเทียบกับอัลกอริทึมของราฟเคมีดอยส์ แบบทำงานเป็นลำดับ

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2555 ลายมือชื่ออาจารย์ที่ปรึกษา_____

WEERASAK CHONGNGULUAM : PARALLELIZE ROUGH K-MEDOIDS CLUSTERING ON MULTICORE PROCESSOR. THESIS ADVISOR : ASSOC. PROF. KITTISAK KERDPRASOP, Ph.D., 92 PP.

MULTI-CORE PROCESSORS/PARALLEL COMPUTING/
DATA MINING/CLUSTERING/ROUGH K-MEDOIDS/ERLANG

Multi-core processors have recently been available on most personal computers, laptop computers, and also smart phones. To get the maximum benefit of computational power from the multi-core architecture, we need a new design on existing algorithms. In this paper, we propose the parallelization of Rough K-Medoids clustering algorithm. In the Rough K-Medoids clustering, each cluster has been formed regarding the two approximations, a lower (data points have been assigned to a specific cluster) and an upper (data points can be assigned to several clusters) approximation. To make Rough K-Medoids clustering be better parallelized, we employ Erlang as a language for concurrent programming with functional paradigm. The experimental results demonstrate considerable speedup rate of the proposed parallel Rough K-Medoids clustering method, compared to the serial Rough K-Medoids approach.

School of Computer Engineering Student's Signature_____
Academic Year 2012 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณ บุคคลต่างๆ ที่ได้กรุณาให้คำปรึกษา แนะนำ ช่วยเหลือ ทั้งในด้านวิชาการ และการดำเนินงานวิจัยดังต่อไปนี้ รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ อาจารย์ที่ปรึกษาวิทยานิพนธ์ รองศาสตราจารย์ ดร.นิตยา เกิดประสพ อาจารย์ ดร.ธรา อั่งสกุล อาจารย์ ดร.ชาญวิทย์ แก้วกสิ และ ผู้ช่วยศาสตราจารย์ ดร.ปรเมศวร์ ห่อแก้ว ที่กรุณาให้คำแนะนำในกระบวนการทำวิจัย และการเขียนวิทยานิพนธ์ฉบับนี้ ขอขอบคุณ เพื่อน ๆ บัณฑิตศึกษาทุกท่าน ที่ให้คำปรึกษา กำลังใจ และให้ความช่วยเหลืออย่างดีมาตลอด ขอขอบคุณ เพื่อน พี่ น้อง ศิษย์เก่าวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ที่ช่วยเหลือ และเป็นที่ปรึกษา ระหว่างศึกษาอยู่ที่ มหาวิทยาลัยเทคโนโลยีสุรนารี

สุดท้ายนี้ ผู้วิจัยขอขอบคุณคณาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ต่าง ๆ ทั้งในอดีต และปัจจุบัน และขอกราบขอบพระคุณบิดา มารดา ที่ให้ความรัก กำลังใจ การอบรมเลี้ยงดูและส่งเสริมการศึกษาเป็นอย่างดีมาโดยตลอด รวมถึงเป็นกำลังใจ เป็นแบบอย่างที่ดี และเป็นแรงพลังดัน อันยิ่งใหญ่แก่ผู้วิจัย จนทำให้ผู้วิจัยประสบความสำเร็จในชีวิตเรื่อยมา

วีรศักดิ์ ช่องงูเหลือม

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในปัจจุบัน เป็นยุคของเทคโนโลยีสารสนเทศและการสื่อสาร ไม่ว่าจะเป็นการดำเนินกิจกรรมใด ๆ ทั้งในทางธุรกิจ การศึกษา วิจัย หรือ การดำเนินงานของหน่วยงานภาครัฐ จำเป็นต้องยุ่งเกี่ยวกับปริมาณข้อมูลมหาศาล จึงเป็นไปได้ยากที่จะทำการแยกแยะ จัดกลุ่ม หาความสัมพันธ์และวิเคราะห์ข้อมูล เพื่อการจัดการข้อมูลได้อย่างมีประสิทธิภาพและนำมาใช้ประโยชน์ได้เหมาะสมที่สุด จึงต้องมีการนำเทคโนโลยีสมัยใหม่ ทางด้านเทคโนโลยีสารสนเทศและการสื่อสารเข้ามาช่วย วิธีการที่กำลังได้รับความนิยมในปัจจุบัน ในการแยกแยะ จัดกลุ่ม หาความสัมพันธ์และวิเคราะห์ข้อมูลคือ การทำเหมืองข้อมูล (Data Mining)

การทำเหมืองข้อมูลเป็นกระบวนการในการสกัดความรู้จากข้อมูลในฐานข้อมูลขนาดใหญ่โดยอัตโนมัติ เพื่อให้ได้ความรู้หรือสารสนเทศที่ยังไม่ทราบมาก่อน แล้วนำสารสนเทศที่ค้นพบมาสร้างแบบจำลองเพื่อการตัดสินใจทางธุรกิจหรือการคาดการณ์ทางด้านวิทยาศาสตร์ การทำเหมืองข้อมูลเป็นขั้นตอนหนึ่งของ “การค้นหาความรู้จากฐานข้อมูล” (Knowledge Discovery in Databases) แต่ปัจจุบันเมื่อเวลากล่าวถึงการทำเหมืองข้อมูลมักจะหมายถึง การค้นหาความรู้จากฐานข้อมูลด้วยเช่นกัน

เทคนิคอย่างหนึ่งของการทำเหมืองข้อมูลที่นิยมใช้เพื่อทำการแบ่งข้อมูลออกเป็นกลุ่มย่อยตามคุณลักษณะที่ใกล้เคียงกัน เรียกว่า การทำ Clustering การจัดกลุ่มข้อมูลมีวัตถุประสงค์เพื่อให้เห็นรูปแบบความคล้ายกันของข้อมูล ตัวอย่างเช่น ธุรกิจค้าปลีกอาศัยการจัดกลุ่มข้อมูลการซื้อสินค้าของลูกค้า เพื่อนำมาวิเคราะห์พฤติกรรมการซื้อสินค้าของลูกค้า หรือ การจัดกลุ่มผู้เข้าใช้งานเว็บไซต์โดยนำข้อมูลมาจากแฟ้มการเข้าใช้งานเว็บ (Web log file) ที่ถูกจัดเก็บโดยโปรแกรมใน Web server

อัลกอริทึมที่นิยมใช้ในการจัดกลุ่มข้อมูลคืออัลกอริทึม เค-มีนส์ (K-Means Clustering) โดยทำการแบ่งข้อมูลออกเป็นจำนวน K กลุ่ม ขั้นตอนการทำงานเริ่มจากเลือกค่าเริ่มต้นเพื่อทำหน้าที่เป็นค่ากลาง (mean) ให้กับแต่ละกลุ่ม จากนั้นจัดข้อมูลเข้ากลุ่มที่ใกล้เคียงกับค่ากลางปัจจุบันของแต่ละกลุ่มที่สุด หลังจากจัดข้อมูลทั้งหมดเข้ากลุ่มเสร็จแล้วทำการหาค่าเฉลี่ยของข้อมูลในแต่ละกลุ่ม

เพื่อทำหน้าที่เป็นค่ากลางของกลุ่ม เมื่อได้ค่ากลางแล้วจะทำการจัดกลุ่มใหม่ไปเรื่อย ๆ จนกว่าจะได้ตำแหน่งที่เสถียร นั่นคือค่ากลางของแต่ละกลุ่มไม่เปลี่ยนแปลง อัลกอริทึม K-Means Clustering นั้นง่ายต่อการเขียนโปรแกรมขึ้นมาใช้งาน แต่ว่าอาจจะทำให้เกิดข้อผิดผลาดสูงถ้าข้อมูลที่นำมาวิเคราะห์มีบางข้อมูลแตกต่างจากข้อมูลอื่นมากเกินไป (outliers) หรือมีข้อมูลรบกวน (noises)

อัลกอริทึมที่ได้รับการพัฒนาสืบเนื่องจาก K-Means Clustering คือ K-Medoids Clustering ทั้งนี้เนื่องจากข้อจำกัดของ K-Means Clustering ที่ค่าคุณลักษณะต่าง ๆ ต้องเป็นเชิงตัวเลข (numerical) เมื่อนำมาใช้กับข้อมูลวัตถุที่ไม่ใช่เชิงตัวเลข จึงไม่สามารถคำนวณค่าเฉลี่ยได้ ทำให้เกิดอัลกอริทึม K-Medoids Clustering สำหรับจัดกลุ่มข้อมูลวัตถุที่ค่าคุณลักษณะเป็นประเภทเชิงกลุ่ม (categorical) งานวิจัยนี้ใช้แนวทางการจัดกลุ่มข้อมูลด้วยอัลกอริทึม K-Medoids Clustering และเพิ่มความยืดหยุ่นให้มากขึ้นด้วยหลักการของ Rough Set Theory

ทฤษฎี ราฟเซต (Rough Set Theory) ได้รับการพัฒนาโดยนักคณิตศาสตร์และคอมพิวเตอร์ชาวโปแลนด์ชื่อ Zdzislaw Pawlak (Pawlak, 1982) และเมื่อไม่นานมานี้ Lingras และ West (2002) ได้นำเสนออัลกอริทึมสำหรับการจัดกลุ่มข้อมูลที่เรียกว่า Rough K-Means Clustering เพื่อทำการแบ่งข้อมูลโดยประมาณออกเป็น กลุ่มที่ต่ำกว่า (lower approximation) กับ กลุ่มที่สูงกว่า (upper approximation) เพื่อใช้จัดกลุ่มข้อมูลที่ในบางครั้งไม่อาจแบ่งกลุ่มได้อย่างชัดเจนแบบอัลกอริทึม K-Means Clustering

ต่อมา Georg Peters และ Martin Lampart (2006) ได้นำเสนออัลกอริทึมในการจัดกลุ่มที่เรียกว่า Rough K-Medoids Clustering โดยได้นำทฤษฎี ราฟเซต มาประยุกต์กับ K-Medoids Clustering สำหรับจัดกลุ่มข้อมูลวัตถุโดยประมาณออกเป็นกลุ่มที่ต่ำกว่าและสูงกว่า

การทำเหมืองข้อมูลด้วยวิธีการจัดกลุ่มนั้นต้องใช้เวลาการประมวลผลนานเพราะการจัดกลุ่มทำได้หลายรูปแบบ ขึ้นอยู่กับว่าจะเลือกคุณลักษณะใดเป็นเป้าหมายหลักของแต่ละกลุ่ม และต้องมีการวนซ้ำหลาย ๆ ครั้งเพื่อคำนวณว่าข้อมูลนั้นควรจะอยู่ในกลุ่มใดมากที่สุด

ในปัจจุบันเทคโนโลยีของหน่วยประมวลผลกลางได้รับการพัฒนาให้มีประสิทธิภาพมากยิ่งขึ้น นอกจากจะเพิ่มความเร็วของสัญญาณนาฬิกาแล้วปัจจุบันยังได้พัฒนาให้ภายในหนึ่งแผ่นของวงจรมีหลายแกนหลัก (multi-core) และราคาไม่แพงมาก ผู้ใช้คอมพิวเตอร์ส่วนบุคคลทั่วไปสามารถหาซื้อมาใช้ได้ แต่ในส่วนของซอฟต์แวร์ยังไม่มีการพัฒนาโปรแกรมให้ใช้งานประสิทธิภาพของแกนหลักที่มีหลาย ๆ แกนนี้ได้อย่างเต็มที่เท่าที่ควร

งานวิจัยนี้จึงมองเห็นว่าควรที่จะนำเอาเทคโนโลยีทางการพัฒนาโปรแกรมสำหรับใช้งานหน่วยประมวลผลกลางแบบมีหลายแกนหลักภายในแผ่นวงจรหนึ่งแผ่น มาพัฒนาโปรแกรมสำหรับใช้งานทางด้านการทำเหมืองข้อมูล โดยเลือกอัลกอริทึมที่ใช้จัดกลุ่มข้อมูลที่เรียกว่า Rough K-

Medoids Clustering ที่ทำงานในแบบลำดับ (serial) มาปรับปรุงให้สามารถทำงานแบบคู่ขนานได้ เพื่อให้สามารถแบ่งกลุ่มข้อมูลได้อย่างรวดเร็วและมีประสิทธิภาพกว่าการพัฒนาโปรแกรมแบบเดิม โดยงานวิจัยฉบับนี้จะเน้นการออกแบบขั้นตอนวิธีแบบ Rough K-Medoids Clustering ที่เหมาะต่อการทำงานในแบบขนาน แล้วนำขั้นตอนวิธีนี้มาพัฒนาโปรแกรมบนหน่วยประมวลผลที่มีหลายแกนหลัก เพื่อให้เกิดการกระจายการทำงานให้ทำพร้อม ๆ กัน แล้วนำผลลัพธ์การทำงาน เวลาที่ใช้มาเปรียบเทียบกับการพัฒนาโปรแกรมแบบเดิมที่เป็นการทำงานแบบลำดับ

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อศึกษาค้นคว้าและวิเคราะห์อัลกอริทึม Rough K-Medoids Clustering ที่ใช้ในการจัดกลุ่มข้อมูล และลดเวลาการจัดกลุ่มโดยการพัฒนาโปรแกรมให้ทำงานบนหน่วยประมวลผลกลางแบบหลายแกนหลักบนแผ่นวงจรเดียว

1.2.2 เพื่อศึกษาวิธีการพัฒนาโปรแกรมบนหน่วยประมวลผลกลางแบบหลายแกนหลัก แล้วนำมาประยุกต์ใช้กับการจัดกลุ่มข้อมูลด้วยอัลกอริทึม Rough K-Medoids Clustering ได้

1.2.3 หาเทคนิคในการที่จะลดเวลาการทำงานของอัลกอริทึม Rough K-Medoids Clustering จากวิธีการเดิม ด้วยการแบ่งการคำนวณเป็นแบบขนานแล้วให้ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักได้

1.2.4 เพื่อสรุปเทคนิคที่ใช้ในการพัฒนาโปรแกรมบนหน่วยประผลกลางแบบมีหลายแกนหลักเพื่อใช้จัดกลุ่มข้อมูล สำหรับเป็นแนวทางในการพัฒนาโปรแกรมทางด้านการทำเหมืองข้อมูลแบบขนานต่อไปได้ในอนาคต

1.3 ขอบเขตของการวิจัย

1.3.1 งานวิจัยชิ้นนี้จะทดสอบอัลกอริทึม Rough K-Medoids Clustering เท่านั้น

1.3.2 ค่าของตัวแปรต่างๆที่เกี่ยวข้องกับทฤษฎีราฟเซต เลือกใช้ตามงานวิจัยของ Georg Peters และ Martin Lampart (2006) เท่านั้น

1.3.3 ทดสอบกับข้อมูล 4 ชุดเทียบกัน ได้แก่

- Synthetic ได้จากการสังเคราะห์
- Colon cancer (http://molbio.princeton.edu/colondata)
- Forest (http://kdd.ics.uci.edu)
- Control chart data (http://kdd.ics.uci.edu)

1.3.4 งานวิจัยนี้พัฒนาด้วยภาษา Erlang และทำงานแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก

1.3.5 ในการพัฒนาจะเขียนโปรแกรมทั้งในแบบลำดับ และแบบขนาน เพื่อเปรียบเทียบประสิทธิภาพกัน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 สามารถทำการจัดกลุ่มข้อมูลของวัตถุที่ไม่ได้แบ่งแยกอย่างชัดเจน ด้วยอัลกอริทึม Rough K-Medoids Clustering ได้อย่างรวดเร็ว

1.4.2 ได้โปรแกรมที่จัดกลุ่มข้อมูลโดยใช้งานประสิทธิภาพของหน่วยประมวลผลกลางที่มีหลายแกนหลักได้อย่างเต็มประสิทธิภาพ

1.4.3 ความรู้ที่ได้จากงานวิจัยนี้สามารถใช้เป็นแนวทางในการปรับปรุงอัลกอริทึมในการทำเหมืองข้อมูลแบบอื่น ๆ ให้ทำงานได้อย่างมีประสิทธิภาพด้วยการพัฒนาโปรแกรมให้ทำงานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก

1.5 คำอธิบายศัพท์

1.5.1 การจัดกลุ่มข้อมูล (Clustering)

หมายถึง การแบ่งกลุ่มข้อมูลด้วยกระบวนการอัตโนมัติ โดยข้อมูลที่อยู่ในกลุ่มเดียวกันจะมีคุณลักษณะที่คล้ายกันกว่าข้อมูลที่อยู่ต่างกลุ่มกัน

1.5.2 ค่าเฉลี่ย (Means)

หมายถึง ค่าเฉลี่ยนของข้อมูล ในที่นี้หมายถึงค่าเฉลี่ยของข้อมูลที่อยู่ภายในกลุ่มเดียวกัน โดยค่านี้จะเป็นค่ากลางสำหรับกลุ่มในอัลกอริทึมแบบ K-Means Clustering

1.5.3 Medoids

หมายถึง คือค่าของวัตถุที่แทนตำแหน่งศูนย์กลางของกลุ่มเหมือนกับค่า Means ในอัลกอริทึม K-Means Clustering แต่ว่าค่า Medoids เองจะเป็นข้อมูลในกลุ่มด้วย ไม่เหมือน Means ที่เกิดจากการคำนวณค่าเฉลี่ย

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการนำเสนอแนวคิดและทฤษฎีพื้นฐานที่ใช้ในการจัดกลุ่มข้อมูล และการเขียนโปรแกรมแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก ในส่วนของการจัดกลุ่มข้อมูลจะอธิบายพื้นฐานเกี่ยวกับการจัดกลุ่มข้อมูล วิธีการวัดระยะห่างของข้อมูลในแบบต่าง ๆ อัลกอริทึมพื้นฐานในการจัดการแบ่งข้อมูลได้แก่ อัลกอริทึม K-Means Clustering อัลกอริทึม K-Medoids Clustering หลังจากนั้นเป็นการอธิบายทฤษฎี ราฟเซต แล้วจึงอธิบายอัลกอริทึมในการแบ่งข้อมูลที่นำเอาทฤษฎี ราฟเซต นี้ไปใช้ได้แก่ อัลกอริทึม Rough K-Means Clustering และอัลกอริทึม Rough K-Medoids Clustering ส่วนถัดไปจะเป็นการอธิบายการเขียนโปรแกรมในแบบขนาน และ การเขียนโปรแกรมบนหน่วยประมวลผลกลางที่มีหลายแกนหลัก การเขียนโปรแกรมทั่วไปด้วยภาษา Erlang และการใช้ภาษา Erlang เพื่อเขียนโปรแกรมให้สามารถประมวลผลบนหน่วยประมวลผลแบบหลายแกนหลัก สุดท้ายจะเป็นการสรุปงานวิจัยที่เกี่ยวข้องและตารางเปรียบเทียบระหว่างงานวิจัยที่เกี่ยวข้องกับงานวิจัยของวิทยานิพนธ์ฉบับนี้

2.1 การจัดกลุ่มข้อมูล (Data Clustering)

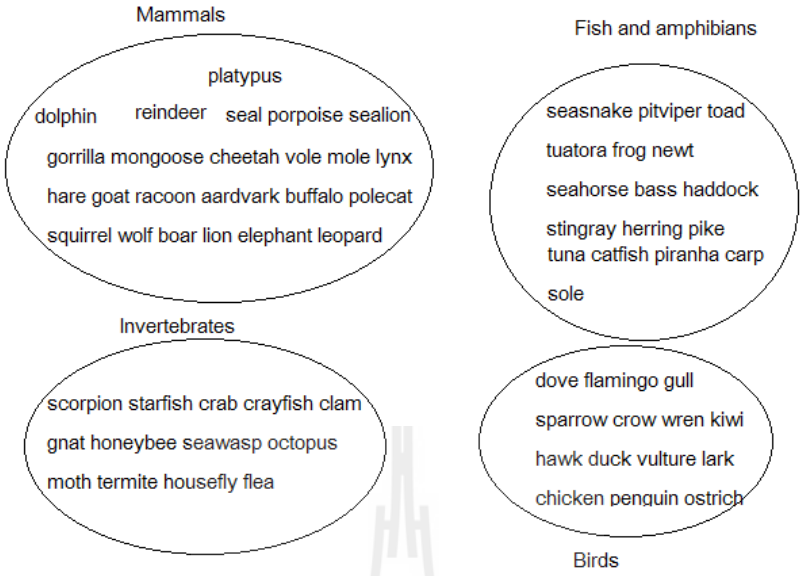
เมื่อเราต้องการทำความเข้าใจข้อมูลขนาดใหญ่ เราสามารถแบ่งข้อมูลออกเป็นกลุ่มขนาดเล็ก ๆ ที่เราสังเกตว่ามีบางอย่างคล้ายกันอยู่ในกลุ่มเดียวกัน การทำความเข้าใจกับข้อมูลขนาดเล็ก ๆ นี้จะช่วยให้สามารถทำความเข้าใจข้อมูลทั้งหมดได้ (Myatt and Johnson, 2009) การจัดกลุ่มข้อมูลถูกใช้อย่างกว้างขวางและเป็นวิธีการที่ยืดหยุ่นในการวิเคราะห์ข้อมูล สังเกต และ การจัดข้อมูลเป็นกลุ่มโดยอัตโนมัติ การสังเกตข้อมูลในกลุ่มใดกลุ่มหนึ่งจะพบว่ามีความคล้ายกันมากกว่า เมื่อเทียบกับข้อมูลที่อยู่คนละกลุ่ม วิธีการนี้ประสบความสำเร็จในการนำไปประยุกต์ใช้ทางด้านวิทยาศาสตร์ และ โปรแกรมทางด้านธุรกิจ รวมทั้งในการวิเคราะห์อาการป่วย การทำประกัน การจัดการข้อมูลประวัติทางด้านการเงิน จัดการผลการค้นหา และ การตลาด ตัวอย่างเช่น การจัดกลุ่มข้อมูลถูกใช้วิเคราะห์ประวัติการซื้อสิ้นค้าของธุรกิจค้าปลีกร่วมกับข้อมูลประวัติของลูกค้าเช่น อายุ และ ที่อยู่จะช่วยทำให้จัดการรณรงค์ทางการตลาดต่าง ๆ ได้ถูกกลุ่มเป้าหมาย

ตัวอย่างการจัดกลุ่มข้อมูลของสัตว์ในตารางที่ 2.1 (นำมาจาก http://archive.ics.uci.edu/ml/datasets/Zoo; Murphy and Aha, 1994) สัตว์แต่ละชนิดจะมีตัวเลขบอกค่าของลักษณะต่าง ๆ บางลักษณะบอกเป็นตัวเลข 1 และ 0 โดยที่ 1 แทนความหมายว่ามีลักษณะนั้นและ 0 แทนว่าไม่มี เช่น มีผมหรือไม่ (hair) หรือ สามารถผลิตน้ำนมได้หรือไม่ (milk) รวมทั้ง การนับจำนวนของขา (legs) การวิเคราะห์ด้วยการแบ่งข้อมูลจะจัดข้อมูลสัตว์ที่คล้ายกันเข้ากลุ่มเดียวกัน โดยอาศัยค่าของตัวแปรในตารางที่ 2.1 ชุดข้อมูลสามารถถูกแบ่งออกได้หลาย ๆ วิธี รูปที่ 2.1 แสดงตัวอย่างหนึ่งที่เป็นไปได้ในการแบ่งกลุ่มข้อมูล ในรูปที่ 2.1 สัตว์ถูกแบ่งออกเป็น 4 กลุ่มดังนี้

1) สัตว์เลี้ยงลูกด้วยนม (mammals)
2) ปลาและสัตว์ครึ่งบกครึ่งน้ำ (fish and amphibians)
3) สัตว์ไม่มีกระดูกสันหลัง (invertebrates)
4) นก (birds)

ในแต่ละกลุ่มประกอบไปด้วยรายชื่อชนิดของสัตว์ที่มีลักษณะคล้ายกัน ในกลุ่มของสัตว์เลี้ยงลูกด้วยนมส่วนใหญ่ประกอบด้วยสัตว์ที่มีลักษณะเหมือนกันคือมีขา 4 ขาและสามารถผลิตน้ำนมได้ ในกลุ่มของปลาและสัตว์ครึ่งบกครึ่งน้ำส่วนใหญ่ประกอบไปด้วยสัตว์ที่อาศัยอยู่ในน้ำ ในกลุ่มของสัตว์ไม่มีกระดูกสันหลังส่วนใหญ่ประกอบไปด้วยสัตว์ที่มีขา 4 ขา และในกลุ่มนกส่วนใหญ่ประกอบไปด้วยสัตว์ที่มีขา 2 ขา และมีปีกบินได้

ด้วยการตรวจสอบขั้นต้นอาจพบว่า มีจำนวนกลุ่มไม่เพียงพอกับข้อมูลสัตว์ทั้งหมด ทำให้สัตว์บางชนิดมีลักษณะไม่ค่อยตรงกับลักษณะของกลุ่ม ตัวอย่างเช่นในรูปที่ 2.1 ปลาหมึก (octopus) ถูกจัดอยู่ในกลุ่มของผึ้ง (honeybee) ในการจัดการปัญหานี้ ชุดข้อมูลจะถูกแบ่งออกเป็น 10 กลุ่ม แทนที่จะมีเพียงแค่ 4 กลุ่ม ทำให้ชนิดของสัตว์ในแต่ละกลุ่มจะน้อยลงและสัตว์ในกลุ่มเดียวกันมีความคล้ายกันมากขึ้นดังรูปที่ 2.2 ในกลุ่มสัตว์เลี้ยงลูกด้วยนมแบ่งย่อยเป็นสัตว์ที่อาศัยอยู่บนบก (land-living mammals) กลุ่มของสัตว์เลี้ยงลูกด้วยนมที่อาศัยอยู่ในน้ำ (aquatic mammals) และกลุ่มของตัวตุ่น ตุ่นนั้นมีกลุ่มแยกออกต่างหากเพราะว่าทั้งอาศัยอยู่ในน้ำ และ มีการวางไข่ ในส่วนของสัตว์ในกลุ่มปลาและสัตว์ครึ่งบกครึ่งน้ำ รวมถึงสัตว์ไม่มีกระดูกสันหลังก็สามารถถูกแบ่งได้เป็นกลุ่มย่อย ๆ ส่วนกลุ่มของนกยังคงเหมือนเดิมเพราะสัตว์ทุกตัวในกลุ่มคล้ายกันหมดอยู่แล้ว รูป 2.3 แสดงให้เห็นกลุ่มย่อยของข้อมูลที่สัมพันธ์กับกลุ่มใหญ่เดิม

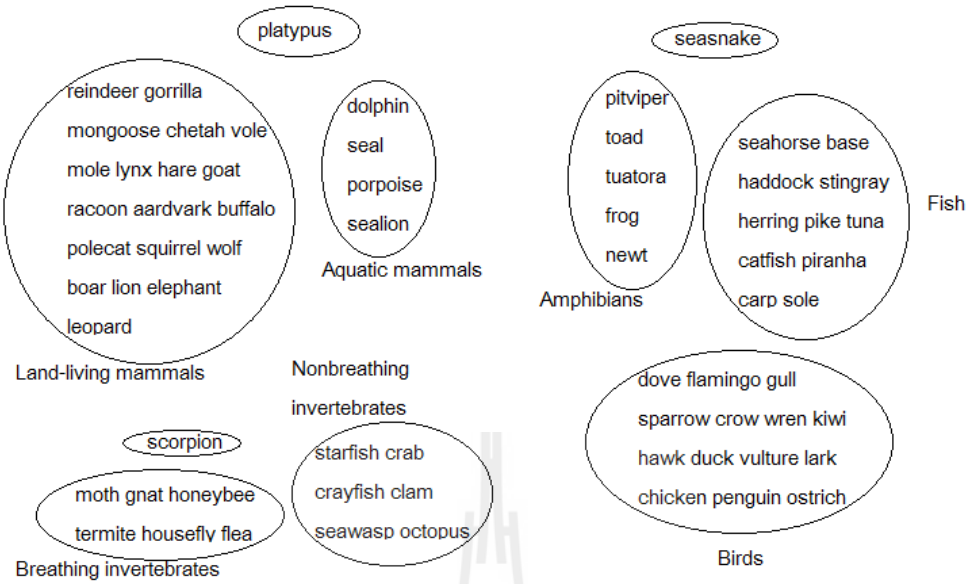


รูปที่ 2.1 แสดงกลุ่มของสัตว์ที่ถูกแบ่งออกเป็น 4 กลุ่ม ได้แก่ mammals, fish and amphibians, invertebrates, และ birds (Myatt and Johnson, 2009:69)

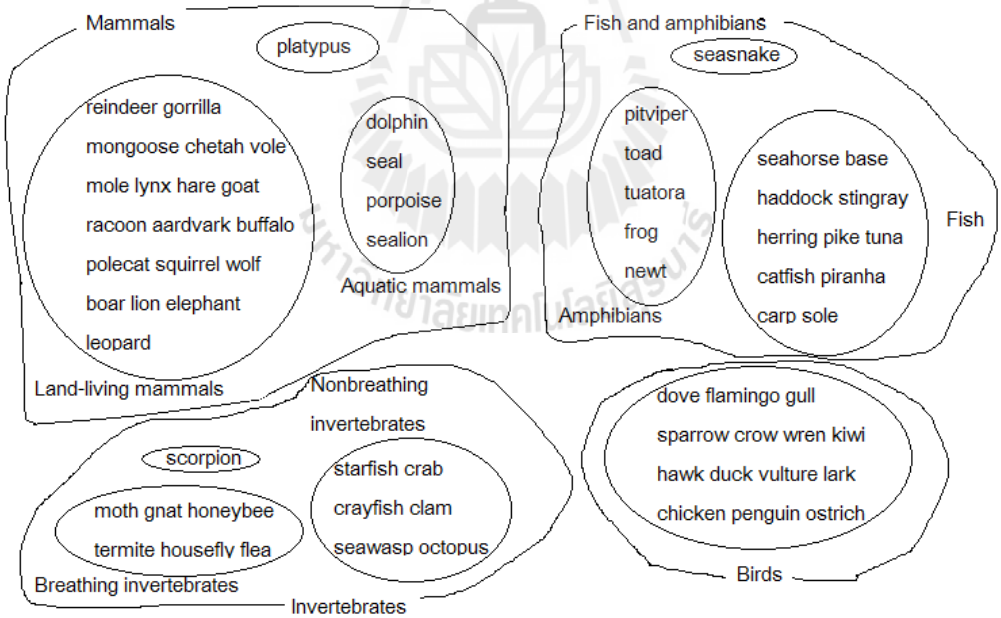
การวิเคราะห์โดยใช้การแบ่งกลุ่ม ไม่จำเป็นต้องทำความเข้าใจข้อมูลและไม่ต้องมีสมมติฐานใด ๆ เกี่ยวกับข้อมูลมาก่อน ตัวอย่างเช่น ตัวแปรไม่จำเป็นต้องผ่านการแจกแจงปรกติ (normal distribution) ไม่จำเป็นต้องมีการกำหนดตัวแปรอิสระ (independent) และตัวแปรตาม (response) เหมือนกับการสร้างแบบจำลองการทำนายผล (predictive models) อย่างไรก็ตามผู้วิเคราะห์ต้องใส่ใจต่อการเลือกตัวแปรที่เหมาะสมและมีความสัมพันธ์กันภายในกลุ่มตัวอย่าง ผลลัพธ์จากการวิเคราะห์ข้อมูลแบบจัดกลุ่มคือ กลุ่มข้อมูลที่ถูกสังเกตว่ามีลักษณะร่วมกัน

ตารางที่ 2.1 ตัวอย่างข้อมูลสัตว์ (Myatt and Johnson, 2009:71)

Name	Hair	Feathers	Eggs	Milk	Airborne	Aquatic	Predator	Toothed	Backbone	Breathes	Venomous	Fins	Legs	Tail
Aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0
Bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1
Boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1
Buffalo	1	0	0	1	0	0	0	1	1	1	0	0	4	1
Carp	0	0	1	0	0	1	0	1	1	0	0	1	0	1
Catfish	0	0	1	0	0	1	1	1	1	0	0	1	0	1
Cheetah	1	0	0	1	0	0	1	1	1	1	0	0	4	1
Chicken	0	1	1	0	1	0	0	0	1	1	0	0	2	1
Clam	0	0	1	0	0	0	1	0	0	0	0	0	0	0



รูปที่ 2.2 แสดงการแบ่งกลุ่มสัตว์ออกเป็น 10 กลุ่ม (Myatt and Johnson, 2009:69)



รูปที่ 2.3 แสดงให้เห็นว่าแต่ละกลุ่มเป็นกลุ่มย่อยของ 4 กลุ่มแรก (Myatt and Johnson, 2009:70)

2.2 วิธีการวัดระยะห่างของข้อมูล

ในการวิเคราะห์ข้อมูลโดยการแบ่งกลุ่มนั้นสามารถทำได้หลายวิธี แต่ว่าทุกวิธีนั้นจะมีขั้นตอนพื้นฐานหนึ่งที่ต้องทำเหมือนกันนั่นคือการวัดระยะห่างของข้อมูล (Distance measures) ระหว่างข้อมูลที่ทำการสังเกตกับค่ากลางของกลุ่มข้อมูล (Johnson and Wichern, 1992) ซึ่งแต่ละวิธีการก็เหมาะสมข้อมูลแต่ละประเภท และ มีข้อดี ข้อเสียที่แตกต่างกัน ซึ่งการวัดระยะที่จะอธิบายในที่นี้มีด้วยกันทั้งหมด 3 แบบได้แก่การวัดระยะแบบ Euclidean, Manhattan และ Minkowski

2.2.1 Euclidean Distance

Euclidean Distance เป็นการวัดระยะห่างระหว่างจุดสองจุด เช่นจุด **p** และจุด **q** ใน Cartesian coordinates ถ้ามี **p** = (p_1, p_2, ..., p_n) และ **q** = (q_1, q_2, ..., q_n) เป็นจุดสองจุดใน Euclidean n-space ระยะระหว่าง **p** ไปจนถึง **q** หรือ **q** ไปจนถึง **p** จะเท่ากับ (Deza and Deza, 2009)

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

โดยที่

d(p, q) = ระยะทางจาก p ไปจนถึง q

d(q, p) = ระยะทางจาก q ไปจนถึง p

n = จำนวนมิติของข้อมูล

ตัวอย่างเช่น ถ้า p = (10, 20) และ q = (30, 40) เป็นจุดสองจุด ระยะห่างระหว่าง p และ q จะเท่ากับ d(p, q) = $\sqrt{(10-30)^2 + (20-40)^2}$ = 28.28

2.2.2 Manhattan Distance

การวัดระยะทางแบบ Manhattan เรียกอีกอย่างว่า Taxicab (Krause, 1987) ถูกคิดขึ้นมาโดย Hermann Minkowski ในคริสต์ศตวรรษที่ 19 โดยวิธีการแบบ Manhattan นั้นวัดระยะระหว่างจุดสองจุดโดยใช้ผลรวมของค่าสัมบูรณ์ของผลต่างของแต่ละมิติ รูปแบบทั่วไปสามารถอธิบายได้ดังนี้

กำหนดระยะทาง d เป็นระยะทางระหว่างเวกเตอร์ p และ q ที่อยู่ใน n มิติ ระยะทางระหว่าง p กับ q อธิบายได้ด้วยสมการดังต่อไปนี้

$$d(p,q) = \|p - q\| = \sum_{i=1}^{n}|p_i - q_i|$$

ซึ่ง

$p = (p_1, p_2, \ldots, p_n)$ และ

$q = (q_1, q_2, \ldots, q_n)$

ตัวอย่างเช่น กำหนด p = (10, 20) และ q = (30, 40) แล้ว ระยะห่างระหว่างจุด p และ q คือ d(p, q) = |10 - 30| + |20 - 40| = 40

2.2.3 Minkowski Distance

การวัดระยะทางแบบ Minkowski (Myatt and Johnson, 2009) นั้นมองได้ว่าเป็นรูปแบบทั่วไปของทั้งวิธีการ Euclidean Distance และ วิธีการ Manhattan Distance นิยามของวิธีการ Minkowski มีดังนี้ กำหนดให้ **p** และ **q** เป็นจุดบน n มิติ ระยะทางระหว่างเวกเตอร์ $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ และ $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ คำนวณได้ดังนี้

$$d(p, q) = \sqrt[\lambda]{\sum_{i=1}^{n} \left| p_i - q_i \right|^{\lambda}}$$

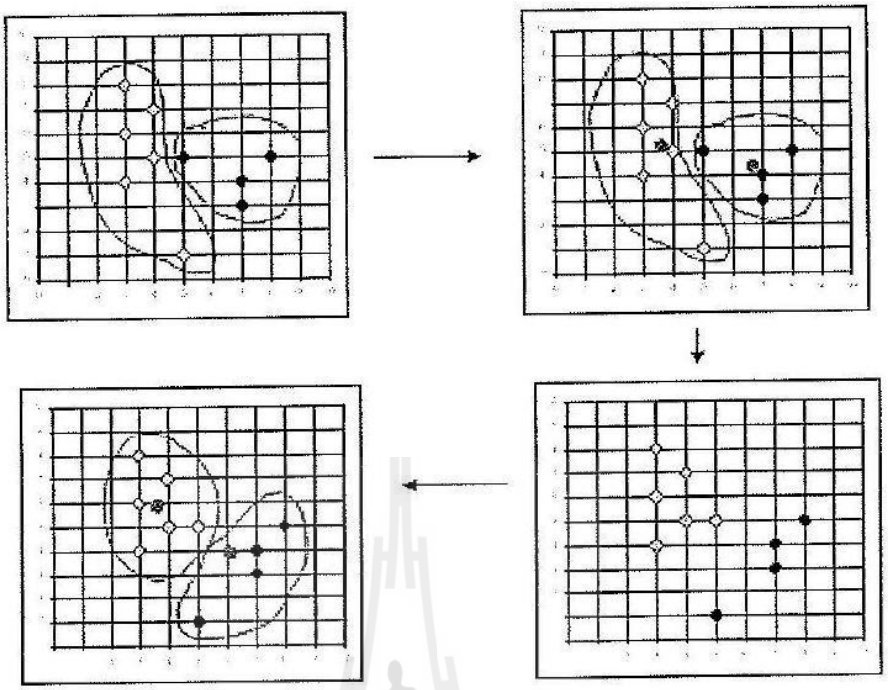
โดยที่ถ้าค่า λ มีค่าเป็น 1 จะได้ผลการคำนวณเหมือนกับ Manhattan Distance ถ้าเป็น 2 จะได้ผลเหมือนกับ Euclidean Distance

ตัวอย่างการคำนวณ ถ้ากำหนดให้ p = (10, 20) , q = (30, 40) และ $\lambda = 3$ แล้ว ระยะห่างระหว่างจุด p และ q คือ

$$d(p, q) = \sqrt[3]{|10-30|^3 + |20-40|^3} = 25.198$$

2.3 อัลกอริทึม K-Means Clustering

อัลกอริทึม K-Means Clustering เป็นอัลกอริทึมเพื่อการจัดกลุ่มข้อมูลที่ได้รับความนิยมมากแบบหนึ่ง โดย K-Means Clustering นั้นอาศัยเทคนิคการจัดกลุ่มด้วยการแบ่งข้อมูล หรือ partitioning (กิตติศักดิ์ เกิดประสพ, 2552) อัลกอริทึมนี้จะเริ่มต้นด้วยการแบ่งข้อมูลออกเป็น K กลุ่ม จากนั้นคำนวนค่ากึ่งกลางของแต่ละกลุ่ม (ค่า mean) เพื่อใช้เป็นค่าอ้างอิงสำหรับการวัดระยะห่างระหว่างข้อมูลแต่ละตัวแบบกลุ่มทั้ง K กลุ่ม ข้อมูลจะถูกจัดเข้าอยู่กลุ่มที่ใกล้ที่สุด กระบวนการจะดำเนินไปจนกระทั่งข้อมูลทั้งหมดไม่มีการเปลี่ยนกลุ่มอีกต่อไป แสดงตัวอย่างได้ดังรูปที่ 2.4 และแสดง pseudo code (Oyelade, Oladipup, and Obagbuwa, 2010) ได้ดังรูปที่ 2.5



รูปที่ 2.4 แสดงตัวอย่างของอัลกอริทึม K-Means Clustering (กิตติศักดิ์ เกิดประสพ, 2552)

Step 1: Accept the number of clusters to group data into and the dataset to cluster as input values

Step 2: Initialize the first K clusters
- Take first k instances or
- Take Random sampling of k elements

Step 3: Calculate the arithmetic means of each cluster formed in the dataset.

Step 4: K-means assigns each record in the dataset to only one of the initial clusters
- Each record is assigned to the nearest cluster using a measure of distance (e.g Euclidean distance).

Step 5: K-means re-assigns each record in the dataset to the most similar cluster and re-calculates the arithmetic mean of all the clusters in the dataset.

รูปที่ 2.5 แสดง Pseudo code ของอัลกอริทึม K-Means Clustering (Oyelade, Oladipupo and Obagbuwa, 2010)

2.4 อัลกอริทึม K-Medoids Clustering

K-Medoids Clustering เป็นอัลกอริทึมที่ถูกนำเสนอโดย Kaufmann (Kaufman and Rousseeuw, 1990) โดยแทนที่จะคำนวณตำแหน่งกลางของกลุ่มด้วยค่าเฉลี่ยของกลุ่ม หรือค่า mean ใน K-Medoids Clustering ตำแหน่งกลางจะแทนที่ด้วยค่าของวัตถุจริง ๆ ที่อยู่ในกลุ่มนั้น ตัวอย่างแสดงผลจากการจัดกลุ่มแบบ K-Medoids ตามรูปที่ 2.6 โดยแบ่งข้อมูลจำนวน 10 ตัวออกเป็น 2 กลุ่ม มี c1 และ c2 เป็นค่ากลางของแต่ละกลุ่ม เทียบกับค่า mean ที่ได้จากอัลกอริทึม K-Means Clustering ที่มี k1 และ k2 เป็นค่ากลางของกลุ่ม

ข้อดีหลัก ๆ ของอัลกอริทึมแบบ K-Medoids Clustering เมื่อเปรียบเทียบกับ K-Means Clustering คือ

1) ในแต่ละกลุ่มจะใช้ข้อมูลของวัตถุตัวใดตัวหนึ่งในกลุ่มนั้นแทนตำแหน่งกลางของกลุ่ม แทนที่จะใช้เป็นค่า mean ซึ่งเป็นค่าเฉลี่ยที่เกิดจากการคำนวณ ไม่ใช่ตำแหน่งที่มีอยู่จริงของข้อมูล

2) K-Medoids ให้ผลลัพธ์ที่ดีกว่าในกรณีที่มี outliers เนื่องจากตำแหน่งกลางของกลุ่มจะยังคงใกล้กับข้อมูลส่วนใหญ่ของกลุ่ม ขณะที่ K-Means Clustering การมี outliers อาจจะทำให้ตำแหน่งกลางของกลุ่มหลุดออกไปอยู่ไกลจากข้อมูลอื่น ๆ ในกลุ่มได้
3) ข้อมูลรบกวนหรือข้อมูลที่ผิด (noise) นั้นมีผลกระทบกับ K-Medoids Clustering น้อยกว่า K-Means Clustering ด้วยเหตุผลเดียวกันกับกรณี outliers
4) การกำหนดเงื่อนไขเป้าหมาย (objective criterion) สามารถกำหนดได้หลายแบบขึ้นอยู่กับความต้องการของผู้ใช้งาน

ข้อเสียหลัก ๆ ของ K-Medoids Clustering คือ

1) เป็นขั้นตอนวิธีที่ใช้การคำนวณเชิงการจัด (combinatorics) ซึ่งทำให้ประสิทธิภาพไม่ดีเท่ากับ K-Means Clustering เนื่องจาก K-Medoids Clustering ต้องทำการคำนวณเปรียบเทียบกับทุกตำแหน่งเพื่อหาตำแหน่งที่ให้ค่าเงื่อนไขเป้าหมายดีที่สุด แต่ K-Means Clustering นั้นเปรียบเทียบแค่กับค่าเฉลี่ยใหม่ในแต่ละรอบ
2) จำเป็นต้องใช้ข้อมูลจริง ๆ ในกลุ่มแทนตำแหน่งกลางของกลุ่ม ทำให้ไม่เคร่งคัดเรื่องคุณภาพของตำแหน่งกลางเมื่อเทียบกับตำแหน่งกลางที่หาโดยค่า mean แบบในอัลกอริทึม K-Means Clustering
3) เมื่อมีการเปลี่ยนแปลงการกระจายตัวของข้อมูลแม้เพียงเล็กน้อย จะทำให้ตำแหน่งกลางของกลุ่มเปลี่ยนไปแบบไม่ต่อเนื่องโดยจะกระโดดจากวัตถุหนึ่งไปอีกวัตถุหนึ่งในกลุ่ม

การทำงานของอัลกอริทึม K-Medoids Clustering จะต้องมีการนิยามเงื่อนไขเป้าหมาย (objective criterion) เป็นความหนาแน่นของกลุ่มข้อมูล (Compactness of the Clustering, CPC) ซึ่งคำนวณได้ดังนี้

$$\text{CPC} = \sum_{k=1}^{K} CPC(C_k)$$

โดยความหนาแน่นของข้อมูลแต่ละกลุ่มคำนวณได้จาก

$$\text{CPC}(\text{C}_\text{k}) = \sum_{x_n \in C_k} d(x_n, m_k)$$

โดย

C_k คือ กลุ่มข้อมูล k

K คือ จำนวนกลุ่มที่ต้องการแบ่ง

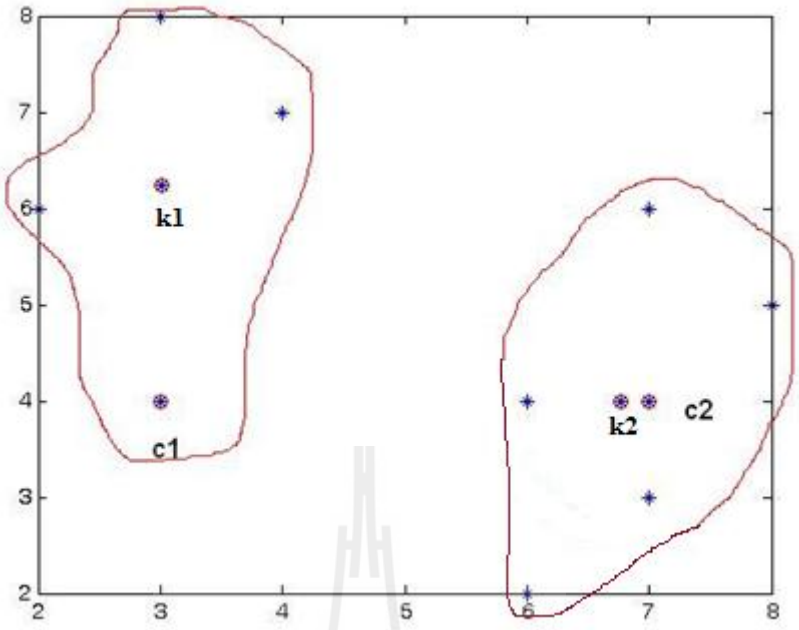
X_n คือ ข้อมูลในกลุ่ม k

m_k คือ ค่า medoid หรือค่ากลางของกลุ่ม k

ขั้นตอนการทำงานของอัลกอริทึม K-Medoids Clustering จะเป็นไปดังนี้

1) กำหนดจำนวนของกลุ่ม เป็น K กลุ่ม
2) สุ่มวัตถุข้อมูลทั้งหมด K ตัวให้เป็นค่าตำแหน่งกลาง (medoids) ของแต่ละกลุ่ม
3) กำหนดวัตถุข้อมูลที่เหลือที่ไม่ได้เป็นตัวที่แทนตำแหน่งกลางให้เป็นสมาชิกในกลุ่มใดกลุ่มหนึ่งโดยเลือกกลุ่มที่วัตถุข้อมูลใกล้กับตำแหน่งกลางปัจจุบันของกลุ่มมากที่สุด
4) หาค่ากลางค่าใหม่ของกลุ่มภายในแต่ละกลุ่มทดลองสลับค่ากลางปัจจุบันเป็นวัตถุอื่นที่ไม่ใช่ค่าตำแหน่งกลาง (non-medoids) แล้วเปรียบเทียบค่าความหนาแน่น (CPC) เพื่อเลือกวัตถุที่เมื่อทำหน้าที่เป็นค่ากลาง (medoids) แล้วให้ค่า CPC ที่ต่ำที่สุด
5) เมื่อเช็คว่าข้อมูลไม่มีการเปลี่ยนกลุ่มอีกต่อไปแล้ว จะหยุดการทำงาน ถ้าไม่จะวนกลับไปทำต่อที่ขั้นตอนที่ 3

พบว่าขั้นตอนการวัดค่าเงื่อนไขเป้าหมาย (object criterion) ของ K-Medoids นั้นแยกออกต่างหากจากขั้นตอนอื่นๆ ทำให้ง่ายต่อการเปลี่ยนวิธีวัดค่าเงื่อนไขเป้าหมายเป็นแบบอื่นๆ



รูปที่ 2.6 แสดงกลุ่มที่ได้จากการแบ่งโดยอัลกอริทึม K-Medoids Clustering และ อัลกอริทึม K-Means Clustering ซึ่งจะได้ตำแหน่ง medoids จากอัลกอริทึม K-Medoids Clustering คือตำแหน่ง c1 และ c2 ส่วนตำแหน่ง means จากอัลกอริทึม K-Means Clustering คือตำแหน่ง k1 และ k2

2.5 ทฤษฎีราฟเซต (Rough Set Theory)

แนวความคิดเกี่ยวกับเรื่อง ราฟเซต นั้นนำเสนอขึ้นครั้งแรกโดย Zdzislaw Pawlak ในช่วงปี 1982 (Kerdprasop and Kerdprasop, 2007) เป็นแนวคิดใหม่เกี่ยวกับเซตและความไม่แน่นอนของสมาชิกของเซต แตกต่างจาก fuzzy set ตรงที่ความไม่แน่นอนของ ราฟเซต นั้นไม่จำเป็นต้องอาศัยความน่าจะเป็น หรือค่าของความน่าจะเป็นในการจัดการกับความคลุมเคลือ แต่ว่าใช้แนวคิดที่เรียบง่ายกว่าของการประมาณค่า (approximation) ขอบเขตล่าง (lower) และ ขอบเขตบน (upper) ซึ่งถูกนิยามไว้ในรูปแบบพื้นฐานของเซต แนวคิดสำคัญของ ราฟเซต ที่นำมาประยุกต์ใช้กับการจัดกลุ่มข้อมูลคือ พื้นที่โดยประมาณ (approximation space) ซึ่งมีนิยามพื้นฐานต่างๆที่จำเป็นดังนี้ (Pawlak, 2002) และรูปแสดงส่วนต่างๆของ ราฟเซตดังรูปที่ 2.7

- ระบบข้อมูลข่าวสาร (Information System): คือคู่อันดับแทนที่ด้วย S = (U, A) ซึ่ง U แทนเซตของเอกภพสัมพัทธ์ (universe) ของเซตของวัตถุ และ A คือเซตของคุณลักษณะ (attributes) ของวัตถุ ทั้งสองเซตเป็นเซตจำกัด
- ทุกสมาชิกของ A แทนด้วย $a \in A$ และเซต V_a แทนเซตของค่าที่เป็นไปได้ของคุณลักษณะ a
- ให้ B เป็นเซตย่อยของ A และกำหนดให้ I(B) คือเซตของความสัมพันธ์บนเอกภพสัมพัทธ์ U เรียกว่า ความสัมพันธ์ที่ไม่สามารถแยกจากกัน (indiscernibility relation) นิยามไว้ว่า
-

$$I(B) = \{(x, y) \in I(B) \mid a(x) = a(y), \forall a \in B\}$$

a(x) คือค่าคุณสมบัติ a ของวัตถุ x และ a(y) คือค่าคุณสมบัติ a ของวัตถุ y

- I(B) จริงๆแล้วก็เป็น ความสัมพันธ์การเท่ากัน (equivalence relation) ทุกๆระดับความเท่ากัน (equivalence class) ตามความสัมพันธ์ของ I(B) จะแบ่ง U ออกเป็นหลายกลุ่มเขียนแทนกลุ่มที่ของ U ที่ถูกแบ่งด้วยความสัมพันธ์ I(B) ด้วย U/I(B) หรือ U/B ซึ่งเรียกได้ว่าเป็น B-elementary
- ระดับความเท่ากันของ I(B) หรือกลุ่มข้อมูล U/B ที่มีวัตถุ x อยู่ในกลุ่มจะเขียนแทนด้วย B(x)
- ถ้า (x, y) ใดๆขึ้นกับความสัมพันธ์ I(B) แล้ว x และ y จะไม่สามารถแยกความแตกต่างได้ในกลุ่มคุณลักษณะ B (B-indiscernible) นั่นคือ x และ y มีค่าเท่ากันหมดสำหรับทุกคุณลักษณะใน B
- ถ้ากำหนดให้ approximation space คือ (Pawlak, 1982)

approximation space = (U,R)

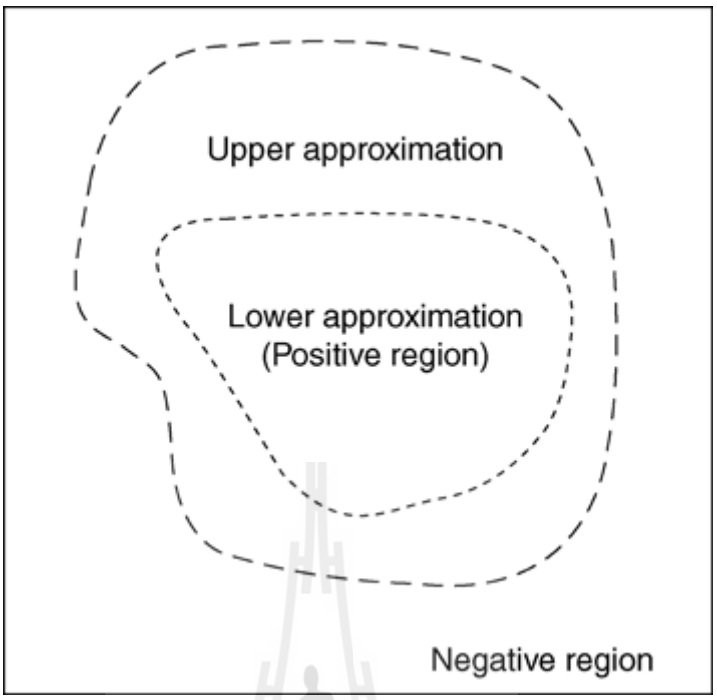
โดยที่ U คือเอกภพสัมพัทธ์ของวัตถุ และ R คือความสัมพันธ์การเท่ากันเช่น I(B)

- ถ้ากำหนด X เป็นเซตย่อยของเอกภพสัมพัทธ์ U และ B เป็นเซตย่อยของเซตคุณลักษณะ A เราจะนิยาม B-lower และ B-upper approximation บน approximation space (U, I(B)) ได้ดังนี้
-

$$B_{lower} = \bigcup_{x \in u} \{B(x) : B(x) \subseteq X\}$$

$$B_{upper} = \bigcup_{x \in u} \{B(x) : B(x) \cap X \neq \varnothing\}$$

- B-lower คือการรวมกันของทุก ๆ กลุ่มข้อมูลที่แบ่งด้วยระดับความเท่ากันโดยความสัมพันธ์ I(B) ที่มี x ที่เป็นซับเซตของ X
- B-upper คือการรวมกันของทุก ๆ กลุ่มข้อมูลที่แบ่งด้วยระดับความเท่ากันโดยความสัมพันธ์ I(B) ที่อินเตอร์เซคกับ X แล้วไม่เท่ากับเซตว่าง
- boundary region คือส่วนที่อยู่ใน upper แต่ไม่อยู่ใน lower
- positive region คือส่วนที่อยู่ใน lower เท่านั้นเท่านั้น
- negative region คือส่วนที่อยู่นอก upper
- ถ้า boundary region เป็นเซตว่างแสดงว่า X นั่นเป็นเซตที่ชัดเจน (crisp) บนเซตคุณลักษณะ B
- ถ้า boundary region ไม่เป็นเซตว่างแสดงว่า X เป็น ราฟเซต บนเซตคุณลักษณะ B



รูปที่ 2.7 แสดงลักษณะของ upper และ lower approximation

2.6 อัลกอริทึม Rough K-Means Clustering

อัลกอริทึม Rough K-Means Clustering เป็นอัลกอริทึมแบ่งกลุ่มข้อมูลโดยในแต่ละกลุ่มจะแบ่งข้อมูลออกเป็น 2 กลุ่มย่อยโดยประมาณนั่นคือ lower approximation และ upper approximation (Zhou, Zhang, Yuan, Lu, 2007) ในส่วนของ lower approximation จะเป็นเซตย่อยของ upper approximation สมาชิกของ lower approximation จะเป็นสมาชิกของกลุ่มอย่างแน่นอน ส่วนวัตถุที่อยู่ในส่วน upper approximation อาจจะอยู่ในกลุ่มแต่ยังมีความไม่แน่นอน โดยที่วัตถุในส่วน boundary region อย่างน้อยต้องเป็นสมาชิกของกลุ่มใดกลุ่มหนึ่ง ในการคิดตำแหน่งกลางใหม่ให้กับกลุ่มด้วยอัลกอริทึม Rough K-Means Clustering ขึ้นอยู่กับค่าพารามิเตอร์ 3 ค่าคือ w_l คือค่าน้ำหนักของส่วน lower approximation , w_b คือค่าน้ำหนักของส่วน boundary region และ ε threshold คือค่าความต่างระยะห่างระหว่างวัตถุที่อยู่ในส่วน boundary region กับค่ากลางของกลุ่มปัจจุบัน กับระยะห่างระหว่างวัตถุที่อยู่ในส่วน boundary region กับกลุ่มอื่นๆ ไม่เกินค่า ε threshold จะถือว่าวัตถุนั้นเป็นสมาชิกของกลุ่มอื่นในส่วน boundary region ด้วย

ในวิธีการของ Lingras (P. Lingras, C. West, 2002) อัลกอริทึม Rough K-Means Clustering สามารถคำนวณค่ากลางของกลุ่มได้ดังสมการต่อไปนี้

$$\vec{m}_k = \begin{cases} w_l \cdot \sum_{x_n \in |\underline{C_k}|} \frac{\vec{X}_n}{|\underline{C_k}|} + w_b \cdot \sum_{X_n \in |C_k^B|} \frac{\vec{X}_n}{|C_k^B|}, \; for C_k^B \neq \varnothing \\ w_l \cdot \sum_{X_n \in |C_k|} \frac{\vec{X}_n}{|\underline{C_k}|}, \; otherwise \end{cases}$$

m_k คือค่า กึ่งกลางกลุ่มค่าใหม่

$|\underline{C_k}|$ คือจำนวนของวัตถุข้อมูลที่อยู่ในส่วนของ lower approximation

$|\underline{C_k^B}| = |\overline{C_k} - \underline{C_k}|$ คือจำนวนของวัตถุข้อมูลที่อยู่ในส่วน boundary region

2.6.1 ขั้นตอนของอัลกอริทึม Rough K-Means Clustering เป็นดังนี้

1) กำหนดค่าพารามิเตอร์เริ่มต้นได้แก่ w_l, w_b, จำนวนของกลุ่ม K และค่า threshold ε
2) สุ่มค่าของวัตถุให้อยู่ในส่วนของ lower approximation ของกลุ่มใดกลุ่มหนึ่ง (ซึ่งตามนิยามจะพบว่าวัตถุถือว่าอยู่ใน upper approximation ด้วย)
3) คำนวณค่ากึ่งกลางกลุ่มค่าใหม่ด้วยวิธีของ Rough K-Means Clustering
4) ในแต่ละวัตถุหาว่าอยู่ใกล้กับค่ากึ่งกลางกลุ่มของกลุ่มไหนมากที่สุด กำหนดให้วัตถุอยู่ในกลุ่มนั้น และถ้าระยะห่างของวัตถุกับกลุ่มที่ใกล้ที่สุด และ ระยะห่างของวัตถุกับกลุ่มอื่น ๆ ต่างกันไม่เกินค่า threshold ε ที่กำหนด จะกำหนดค่าวัตถุเข้าไปในส่วนของ upper approximation ของแต่ละกลุ่มที่ผ่านเงื่อนไขนี้ด้วย ถ้าไม่มีกลุ่มไหนผ่านเงื่อนไขนี้ จะให้ข้อมูลอยู่ในส่วน lower approximation ของกลุ่มที่ใกล้ที่สุดเท่านั้น
5) เมื่อเช็คว่าข้อมูลไม่มีการเปลี่ยนกลุ่มอีกต่อไปแล้ว จะหยุดการทำงาน ถ้าไม่จะวนกลับไปทำต่อที่ขั้นตอนที่ 3

2.6.2 ส่วนขยายและความเปลี่ยนแปลงของ Rough K-Means Clustering

Mitra (2004) ได้เสนอว่า การกำหนดค่าเริ่มต้นของพารามิเตอร์ที่ดีเป็นเรื่องที่ยาก และเป็นความท้าทายหลักในการใช้งานอัลกอริทึม Rough K-Means Clustering เขาจึง ได้ประยุกต์ใช้ genetic algorithm เพื่อหาค่าพารามิเตอร์ที่เหมาะสม โดยใช้การตรวจสอบที่ชื่อว่า Davies-Bouldin cluster validity เป็นเกณฑ์

Davies-Bouldin index เป็นเกณฑ์การวัดคุณภาพการจัดกลุ่มที่เป็นอิสระต่อจำนวนของกลุ่มที่จะวิเคราะห์และไม่ขึ้นอยู่กับวิธีในการแบ่งกลุ่ม ดังนั้นจึงถูกนำมาใช้เป็นเกณฑ์ในการเปรียบเทียบความต่างระหว่างสองอัลกอริทึมที่ใช้ในการวิเคราะห์เพื่อการแบ่งกลุ่มข้อมูล

การคำนวณค่าของ Davies-Bouldin index แบบง่าย ๆ คือเป็นอัตราส่วนระหว่างผลรวมของการกระจายตัวของข้อมูลในกลุ่ม และ ระยะห่างระหว่างกลุ่ม จะเห็นว่าการแบ่งกลุ่มที่ดีนั้น การกระจายตัวในกลุ่มจะต้องน้อย และ ระยะห่างระหว่างแต่ละกลุ่มจะต้องมาก การหาค่าของ Davies-Bouldin index เป็นดังสมการต่อไปนี้

$$\frac{1}{K}\sum_{k=1}^{K}\max_{k\neq l}\left\{\frac{S(U_k)+S(U_l)}{d(U_k,U_l)}\right\}$$

ค่า $S(U_k)+S(U_l)$ เป็นระยะห่างของข้อมูลภายในกลุ่ม k และกลุ่ม l ส่วน $d(U_k, U_l)$ เป็นระยะห่างระหว่างจุดกึ่งกลางกลุ่ม k กับกลุ่ม l ดังนั้นการทำให้ค่าของ Davies-Bouldin index เล็กที่สุดจะทำให้ได้การแบ่งแยกของกลุ่มดีที่สุด

2.7 อัลกอริทึม Rough K-Medoids Clustering

Georg Peters and Martin Lampart (2006) ได้นำเสนอวิธีการนำ ราฟเซต มาประยุกต์ใช้กับการแบ่งกลุ่มข้อมูลแบบ K-Medoids Clustering โดยนิยามตัวแปรดังต่อไปนี้

- X_n คือ เซตของข้อมูล โดยที่ n = 1, …, N
- m_k คือ ค่ากึ่งกลาง (medoid) ของกลุ่ม C_k โดยที่ k = 1, …, K
- $d(X_n, m_k) = ||X_n - m_k||$ คือ ระยะห่างระหว่างวัตถุ X_n และ ค่ากึ่งกลาง m_k
- RCPC (Rough Compactness of Clustering) คือ ค่าของความหนาแน่นกลุ่มข้อมูลแบบราฟ โดยที่

$$\text{RCPC} = \sum_{k=1}^{K} RCPC(C_k)$$

$$\text{RCPC}(C_k) = w_l \sum_{X_n \in C_k} d(X_n, m_k) + w_b \sum_{X_n \in (\overline{C_k} - \underline{C_k})} d(X_n, m_k)$$

ค่าพารามิเตอร์ w_l และ w_b จะมีผลต่อขนาดพื้นที่ของ lower approximation และ boundary region ของกลุ่มในการคำนวณค่า RCPC

ขั้นตอนวิธีของ Rough K-Medoids Clustering มีดังต่อไปนี้

1) สุ่มเลือกข้อมูล K ตัวให้เป็นค่า Medoids: m_k , k = 1, …, K โดยแต่ละตัวจะเป็นสมาชิกของ lower approximation : $m_k \in \underline{C_k}$ ส่วนวัตถุข้อมูลที่เหลือคือที่ไม่ได้ค่ากึ่งกลางคือ X'_m โดยที่ m = 1, …, (N-K)
2) กำหนดวัตถุข้อมูลที่เหลือ (N-K) วัตถุ ของ X'_m ไปใน K กลุ่มโดยทำตาม 2 ขั้นตอนย่อย ในขั้นตอนแรกให้กำหนดข้อมูลอยู่ในพื้นที่ของ upper approximation ของกลุ่มที่อยู่ใกล้ที่สุด ขั้นตอนถัดไปพยายามกำหนดข้อมูลวัตถุที่อยู่ใน upper approximation ของกลุ่มที่ใกล้ที่สุดให้อยู่ใน upper approximation ของกลุ่มอื่นที่ใกล้กันด้วย โดยดูที่เงื่อนไขความต่างของระยะทางว่าน้อยกว่าค่า ε threshold หรือไม่ ถ้าไม่จะอยู่ใน lower approximation ของกลุ่มที่ใกล้ที่สุดเท่านั้น รายละเอียดของทั้งสองขั้นตอนย่อยมีดังนี้
   (i) เช็คว่าข้อมูลวัตถุใน X'_m ใกล้กับ Medoid m_k ตัวไหนที่สุด

   $$d(X'_m, m_k) = \min_{h=1,\ldots K} d(X'_m, m_h)$$

   กำหนดข้อมูล X'_m ให้อยู่ใน upper approximation ของกลุ่ม k: $X'_m \in \overline{C_k}$
   (ii) กำหนดกลุ่ม C_h ที่อยู่ใกล้กับ X'_m และระยะห่างจาก X'_m ต้องไม่เกิน $d(X'_m, m_k) + \varepsilon$ ซึ่ง ε คือค่า threshold ที่กำหนดมา:

   $$T = \{h: d(X'_m, m_h) - d(X'_m, m_k) \leq \varepsilon \wedge h \neq k\}$$

   - T คือเซตของกลุ่มข้อมูลกลุ่มอื่นๆ ที่ระยะห่างของค่ากึ่งกลางกลุ่มนั้นกับวัตถุ X'_m ลบระยะห่างของค่ากึ่งกลางกลุ่มที่ใกล้ที่สุดของวัตถุ X'_m ในปัจจุบันแล้ว น้อยกว่า

หรือเท่ากับค่า ε threshold ถ้า T $\neq \varnothing$ แล้ว X'_m จะอยู่ใกล้กับกลุ่มอื่นๆที่อยู่ข้างๆ กลุ่มที่ใกล้ที่สุดปัจจุบันด้วยอย่างน้อยหนึ่งตัว หมายความว่า X'_m จะเป็นสมาชิกของ upper approximation ในส่วน boundary region ของทุกๆกลุ่มที่เป็นสมาชิกของ T ด้วย คือ

$$X'_m \in \overline{C_h}, \forall h \in T$$

- ไม่เช่นนั้นแล้ว X'_m จะเป็นสมาชิกของ lower approximation ของกลุ่มที่ใกล้ที่สุดปัจจุบันเท่านั้น คือ

$$X'_m \in \underline{C_k}$$

3) คำนวณค่า $RCPC_{current}$ ใหม่สำหรับข้อมูลการจัดกลุ่มปัจจุบัน
4) สลับค่ากึ่งกลาง m_k กับทุกข้อมูลวัตถุ X'_m ทุกตัว และคำนวณค่า $RCPC_{k\leftrightarrow m}$ ของการสลับค่ากึ่งกลางตัวที่ k กับข้อมูลวัตถุตัวที่ m แล้วหาว่าการสลับของ ค่ากึ่งกลางกับวัตถุข้อมูลตัวใดทำให้ค่า RCPC น้อยที่สุด ให้ RCPC ที่น้อยที่สุดคือ

$$RCPC_{k_0\leftrightarrow m_0} = \min_{\forall k,\forall m} RCPC_{k\leftrightarrow m}$$

สำหรับทุกๆ k = 1, …, K และ m = 1, …, (N-K)

- ถ้า $RCPC_{k_0\leftrightarrow m_0} < RCPC_{current}$ แล้ว จะสลับ Medoid m_{k_0} และ วัตถุข้อมูล X_{m_0} และกำหนด $RCPC_{current} = RCPC_{k_0\leftrightarrow m_0}$ แล้ววนกลับไปขั้นตอนที่ 2 จนกว่า ข้อมูลจะไม่มีการเปลี่ยนกลุ่มอีกต่อไปแล้ว จะหยุดการทำงาน ถ้าไม่จะวนกลับไปทำต่อที่ขั้นตอนที่ 2

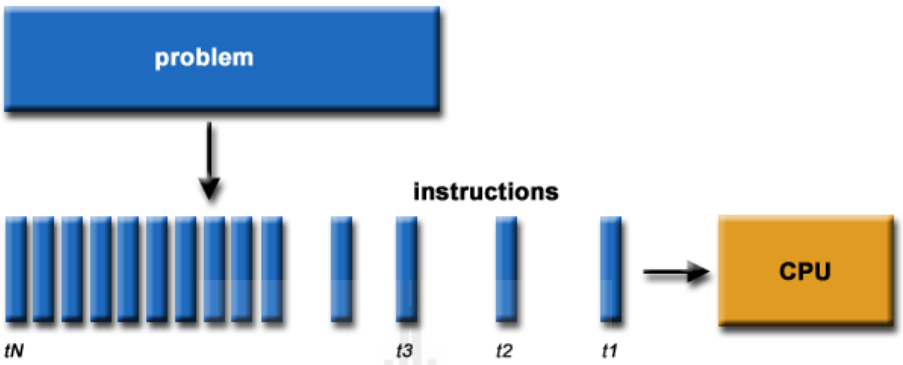
2.8 การเขียนโปรแกรมในแบบขนาน (Parallel Programming)

โปรแกรมโดยทั่วไปแล้วจะทำงานในแบบลำดับ (serial programming) นั่นคือทำการประมวลผลทีละคำสั่งทำให้มีลักษณะการทำงานโดยทั่วไปคือ (B. Barney, 2011)

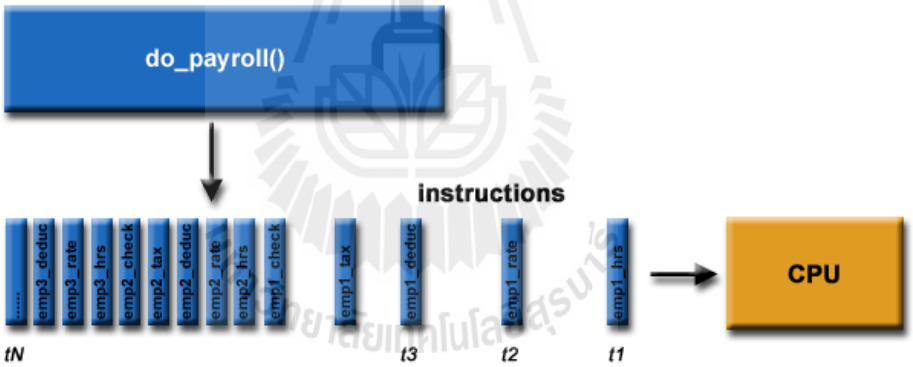
- ทำงานอยู่บนคอมพิวเตอร์เครื่องเดียว ที่มีเพียงหนึ่งหน่วยประมวลผลกลาง
- ต้องแตกปัญหาออกเป็นคำสั่งย่อย เพื่อจัดลำดับการทำงานทีละคำสั่ง
- คำสั่งต้องทำงานต่อจากคำสั่งอื่น

- มีเพียงคำสั่งเดียวเท่านั้นที่จะทำงาน ณ ขณะใดขณะหนึ่ง

ตัวอย่างการทำงานแบบ serial แสดงเป็นแผนภาพได้ดังรูปที่ 2.8 และ 2.9



รูปที่ 2.8 แสดงการแตกปัญหาออกเป็นคำสั่งย่อยๆและทำงานบน 1 หน่วยประมวลผลกลาง (B. Barney, 2011)

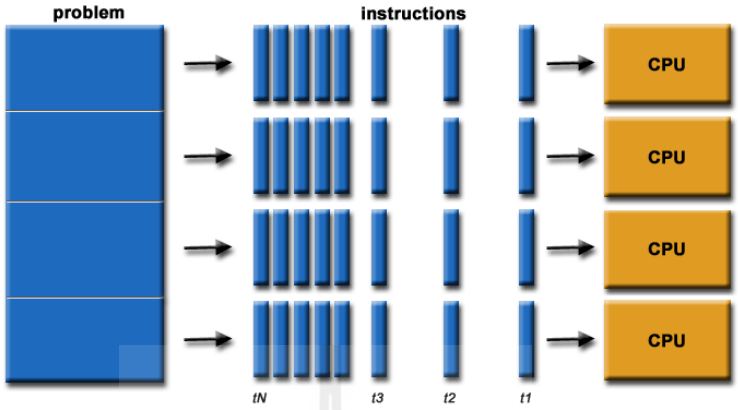


รูปที่ 2.9 แสดงการทำงานของฟังก์ชัน do_payroll() ที่ถูกทำทีละคำสั่งบน 1 หน่วยประมวลผลกลาง (B. Barney, 2011)

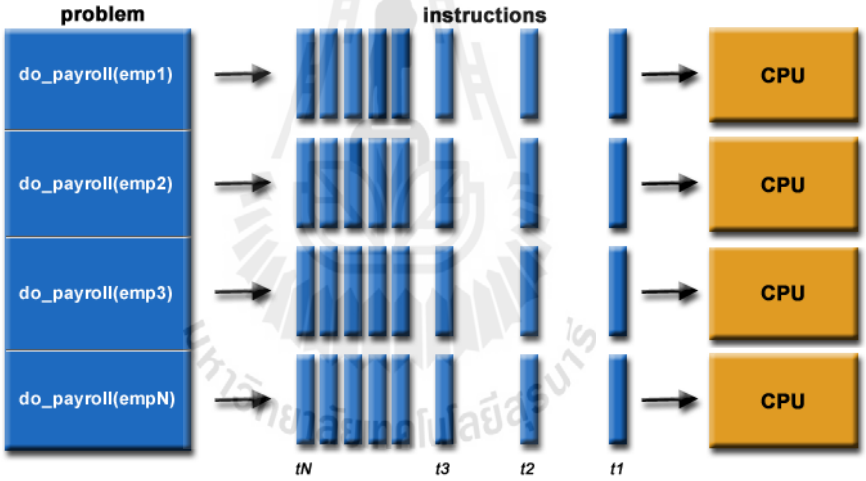
ในการทำงานแบบขนานนั้นจะเป็นการประมวลผลพร้อมกันโดยใช้ทรัพยากรหลาย ๆ ตัว เพื่อแก้ปัญหาอย่างใดอย่างหนึ่ง ลักษณะโดยทั่วไปของการโปรแกรมแบบขนานเป็นดังนี้

- ทำงานบนหลาย ๆ หน่วยประมวลผลกลาง
- ปัญหาถูกแบ่งออกเป็นส่วนย่อยที่สามารถแยกกันทำงานได้
- ในแต่ละส่วนย่อยสามารถแยกเป็นลำดับคำสั่ง
- คำสั่งในแต่ละส่วนย่อยทำงานไปพร้อม ๆ กันบนหน่วยประมวลผลกลางคนละตัว

ตัวอย่างการทำงานแบบขนานแสดงเป็นแผนภาพได้ดังรูปที่ 2.10 และ 2.11



รูปที่ 2.10 แสดงตัวอย่างการแก้ปัญหาด้วยการทำงานแบบขนาน (B. Barney, 2011)



รูปที่ 2.11 แสดงตัวอย่างการทำงานฟังก์ชัน do_payroll() ในแบบขนาน

2.9 ประเภทของการประมวลผลแบบขนานในแบบของ Flynn (Flynn's Classical Taxonomy)

การแบ่งประเภทของการประมวลผลแบบขนานทำได้หลายวิธี แต่วิธีหนึ่งที่ได้รับความนิยมคือการแบ่งในแบบของ Flynn (B. Barney, 2011)

การแบ่งในแบบของ Flynn นั้นจะแบ่งตามวิธีการในการประมวลผลว่าประมวลผลครั้งละกี่คำสั่ง และ ประมวลผลกับข้อมูลหลายชุดพร้อมกัน หรือ ชุดเดียวกัน โดยแบ่งวิธีการประมวลผลออกได้เป็น 4 แบบดังต่อไปนี้

- **SISD (Single Instruction, Single Data)**
  เป็นรูปแบบที่จะมีเพียงหนึ่งลำดับคำสั่งที่ทำงานบนหน่วยประมวลผลกลางเมื่อมีสัญญาณนาฬิกา และ มีเพียงสายข้อมูลเพียงหนึ่งลำดับเท่านั้นที่ถูกใช้เป็นข้อมูลนำเข้าระหว่างการประมวลผลเมื่อมีสัญญาณนาฬิกา ลำดับการทำงานของคำสั่งบ่งบอกได้อย่างชัดเจน ซึ่งการทำงานประเภทนี้เป็นวิธีการที่เก่าที่สุดซึ่งปัจจุบันก็ยังมีใช้กันอยู่ ตัวอย่างเครื่องที่เป็นประเภทนี้ได้แก่เครื่องเมนเฟรม, มินิคอมพิวเตอร์ และ เวิร์กสเตชัน รุ่นเก่า
- **SIMD (Single Instruction, Multiple Data)**
  เป็นรูปแบบที่จะมีเพียงหนึ่งลำดับคำสั่งที่ทำงานบนหน่วยประมวลผลกลางเมื่อมีสัญญาณนาฬิกา และ มีหลายสายลำดับข้อมูลที่ถูกใช้เป็นข้อมูลนำเข้าระหว่างการประมวลผลโดยที่แต่ละหน่วยประมวลผลสามารถประมวลผลข้อมูลที่แตกต่างกันได้ ลักษณะของหน่วยประมวลผลแบบนี้มีอยู่ด้วยกันสองแบบคือ processor arrays และ vector pipelines ตัวอย่างเครื่องคอมพิวเตอร์ที่ประมวลผลลักษณะนี้ได้แก่ CM-2, MasPar MP-12 MP-2, Cray X-MP, Y-MP, Fujitsu VP และเครื่องคอมพิวเตอร์สมัยใหม่โดยเฉพาะหน่วยประมวลผลสำหรับกราฟิกจะเป็นแบบ vector pipelines SIMD
- **MISD (Multiple Instruction, Single Data)**
  ประมวลผลหลายคำสั่งโดยในแต่ละหน่วยประมวลผลทำงานกันคนละลำดับคำสั่งได้ โดยที่มีเพียงลำดับของข้อมูลนำเข้าเพียงหนึ่งลำดับ ตัวอย่างของเครื่องที่ประมวลผลแบบนี้มีอยู่น้อย เช่นเครื่อง Carnegie Mellon C.mmp computer (1971)
- **MIMD (Multiple Instruction, Multiple Data)**
  ประมวลผลหลายคำสั่งโดยในแต่ละหน่วยประมวลผลทำงานกันคนละลำดับคำสั่งได้ และในแต่ละหน่วยประมวลผลสามารถมีลำดับของข้อมูลนำเข้าคนละลำดับแยกกันต่างหาก ลำดับการประมวลผลเป็นได้ทั้งแบบเข้าจังหวะ และ ไม่เข้าจังหวะ ทั้งสามารถกำหนดได้และ ไม่สามารถกำหนดได้ ในปัจจุบันเป็นรูปแบบที่ได้รับความนิยมมากที่สุดสำหรับการประมวลผลแบบขนาน เครื่องซูเปอร์คอมพิวเตอร์สมัยใหม่ส่วนใหญ่จะเป็นประเภทนี้ นอกจากนั้นยังรวมถึงเครื่องที่นำมาเชื่อมต่อโครงข่ายการทำงานกันให้เป็นแบบกริด (grid)

multi-processor SMP และ multi-core PCs ในปัจจุบัน ในสถาปัตยกรรมแบบ MIMD ยังรวมแบบ SIMD เป็นสถาปัตยกรรมย่อย ๆ อยู่ด้วย

นอกจากนี้ยังมี

- **การประมวลผลหลายโปรเซสแบบสมมาตร (SMP, Symmetric Multi Processing)** เครื่องที่ทำการประมวลผลหลายโปรเซสแบบสมมาตรนั้นจะมีจำนวนหน่วยประมวลผลตั้งแต่สองตัวขึ้นไปซึ่งเชื่อมต่อกันกับหน่วยความจำร่วม ซึ่งหน่วยประมวลผลเหล่านี้อาจจะอยู่บนแผ่นวงจรเดียวกันหรือกระจายกันอยู่ในหลายๆแผ่นวงจรหรือมีทั้งสองแบบรวมกัน

2.10 การเขียนโปรแกรมบนหน่วยประมวลผลกลางที่มีหลายแกนหลักด้วยภาษา Erlang

เทคโนโลยีของหน่วยประมวลผลกลางที่มีหลายแกนหลัก ทำให้เราสามารถพัฒนาโปรแกรมแบบขนานได้ง่ายโดยใช้เพียงเครื่องคอมพิวเตอร์ส่วนบุคคลที่หาซื้อได้ทั่วไป หรือแม้กระทั่งปัจจุบันหน่วยประมวลผลกลางสำหรับโทรศัพท์เคลื่อนที่ต่าง ๆ ก็มีหน่วยประมวลผลกลางแบบหลายแกนหลักให้ใช้แล้วเช่นกัน การเขียนโปรแกรมเพื่อให้สามารถทำงานแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลักนั้นมีเครื่องมือที่แตกต่างกันแล้วแต่ภาษา เช่นสำหรับภาษาซี มีไลบรารี่สำหรับพัฒนาโปรแกรมแบบขนานได้แก่ MPI, OpenMP, OpenCL ส่วนหน่วยประมวลผลกลางสำหรับการแสดงผลของบริษัท Nvidia มีเครื่องมือพัฒนาโดยเฉพาะคือ Cuda ซึ่งการพัฒนายังมีความยุ่งยากเพราะยังใช้ภาษาในลักษณะ เช่นภาษาซีในการพัฒนา ในปัจจุบันมีภาษาระดับสูงอื่น ๆ และภาษาเชิงฟังก์ชัน ที่เขียนได้ง่ายกว่าออกมาเช่นภาษา Erlang ที่สามารถเขียนโปรแกรมโดยใช้แนวคิดฟังก์ชัน สามารถกำหนดให้ทำงานแบบหลาย ๆ โปรเซสบนหน่วยประมวลผลกลางที่มีหลายแกนหลักได้โดยอัตโนมัติ ไม่จำเป็นต้องแก้ไขแนวคิดและขั้นตอนคำสั่งในการพัฒนาโปรแกรม ในงานวิจัยนี้จึงเลือกใช้ภาษา Erlang ในการพัฒนา

2.10.1 ประวัติของภาษา Erlang

Erlang เป็นภาษาการเขียนโปรแกรมที่ถูกออกแบบมาเพื่อสนับสนุนการประมวลผลแบบพร้อมกัน (concurrent programming) ที่โปรแกรมหลักต้องทำงานอยู่ตลอดเวลา Erlang ใช้การประมวลผลแบบพร้อมกันในการออกแบบโครงสร้างโปรแกรมโดยที่โปรเซสของ Erlang จะไม่มีการใช้หน่วยความจำร่วมกัน และ ใช้การติดต่อระหว่างโปรเซสแบบอะซิงโครนัส (asynchronous) โปรเซสของ Erlang นั้นเป็นแบบเบา (lightweight) คือไม่ใช้โครงสร้างของโปรเซสของระบบปฏิบัติการแต่เป็นโครงสร้างที่สร้างขึ้นมาใหม่ ใช้ทรัพยากรน้อยกว่าจึงสามารถสร้างโปรเซสได้มากกว่าการใช้โปรเซสของระบบปฏิบัติการ และขึ้นอยู่กับตัวภาษาเอง ไม่ขึ้นอยู่กับระบบปฏิบัติการ Erlang มีกลไกในการที่จะเปลี่ยนแปลงโค้ดของโปรแกรมขณะที่โปรแกรมกำลัง

ทำงาน ด้วยกลไกแบบนี้ทำให้สามารถสร้างโปรแกรมที่ไม่จำเป็นต้องหยุดทำงานเมื่อการแก้ไขเกิดขึ้นที่บางโปรเซส

จุดเริ่มต้นของการพัฒนาภาษา Erlang นั้นเริ่มมาตั้งแต่ปี ค.ศ. 1986 ที่ Ericson Computer Science Laboratory Erlang นั้นถูกออกแบบมาเพื่อจุดประสงค์เฉพาะทางในขณะนั้นคือเพื่อพัฒนาโปรแกรมประยุกต์ทางด้านโทรคมนาคม ต่อมาบริษัทอิริกสันใช้งานภาษา Erlang ในงานประยุกต์ต่าง ๆ ประสบความสำเร็จเป็นอย่างมาก ในช่วงแรก ลิขสิทธิ์ของภาษา Erlang และเครื่องมือในการพัฒนายังไม่ได้อยู่ในรูปแบบโอเพนซอร์ส จนกระทั่ง 2 ธันวาคม ค.ศ. 1998 ได้มีการทำให้ลิขสิทธิ์ของภาษา Erlang และเครื่องมือในการพัฒนาเป็นโอเพนซอร์ส ทำให้มีการร่วมกันพัฒนา Erlang ของนักพัฒนาทั่วไปอย่างกว้างขวางขึ้นจนมีผู้ใช้และมีโปรแกรมที่สร้างจากภาษา Erlang อยู่มากในบริษัทชั้นนำในปัจจุบัน

2.10.2 โครงสร้างของโปรแกรมภาษา Erlang

การเขียนโปรแกรมภาษา Erlang จะเขียนในลักษณะเชิงฟังก์ชัน (functional programming) โดยจะแยกกลุ่มฟังก์ชันออกเป็นโมดูล (Module) และโมดูลของ Erlang จะถูกเรียกใช้งานผ่านเวอร์ชวลแมตชีนของ Erlang

2.10.3 ชนิดข้อมูลในภาษา Erlang

ตัวแปรในภาษา Erlang นั้นไม่จำเป็นต้องประกาศตัวแปรก่อนการใช้งานแต่ว่า ตัวแปรในภาษา Erlang นั้นเมื่อกำหนดค่าแล้ว จะไม่สามารถกำหนดค่าอื่นซ้ำลงไปได้อีก(Immutable) ชนิดข้อมูลพื้นฐานของภาษา Erlang นั้นประกอบไปด้วย เลขจำนวนเต็ม, เลขทศนิยม, ลิสต์, ทูเพิล และ อะตอม ข้อมูลแบบจำนวนเต็มของ Erlang จะไม่จำกัดขนาดความยาวแบบภาษาซีหรือจาวาสามารถเก็บได้ยาวเท่าไหร่ก็ได้จนกว่าหน่วยความจำจะไม่พอ ข้อมูลแบบ ลิสต์จะเป็นการนำข้อมูลแบบอื่น ๆ มารวมอยู่ด้วยกันโดยสมาชิกแต่ละตัวของลิสต์จะอยู่ในเครื่องหมายวงเล็บ [ ] ส่วนทูเพิลนั้นจะเป็นการนำข้อมูลมารวมเป็นกลุ่มแบบลิสต์ แต่แตกต่างจากลิสต์ในด้านการทำแพทเทิร์นแมตชิ่ง โดยทูเพิลจะไม่สามารถทำแพทเทิร์นแมตชิ่งระหว่างข้อมูลตัวแรกกับสมาชิกที่เหลือแบบลิสต์ได้ ส่วนอะตอมนั้นจะเป็นข้อมูลแบบสัญลักษณ์โดยที่อะตอมต้องขึ้นต้นด้วยตัวอักษรตัวเล็กเสมอหรือใช้เครื่องหมาย ‘’ ครอบอะตอมเอาไว้ ตัวแปรในภาษา Erlang ต้องตั้งชื่อขึ้นต้นด้วยตัวอักษรตัวใหญ่หรือใช้ _ เท่านั้น

ตารางที่ 2.2 แสดงชนิดข้อมูลของภาษา Erlang และ ตัวอย่างการใช้งาน

ชนิดข้อมูล	ตัวอย่างการใช้งาน
เลขจำนวนเต็ม	A = 10
เลขทศนิยม	B = 21.11
ลิสต์	L = [1,2,3,4.5], L2 = “Hello World\n”
ทูเพิล	Name = {name, “Weerasak Chongnguluam”}
อะตอม	C = car, R = red

2.10.4 การเขียนฟังก์ชันใน Erlang

Erlang เป็นภาษาเชิงฟังก์ชัน จึงต้องเขียนการทำงานของโปรแกรมอยู่ในรูปของฟังก์ชัน ดังแสดงในรูปที่ 2.12

```
-module(main).
-export(main/0).
main() ->
    io:format(“Hello World~n”).
```

รูปที่ 2.12 แสดงตัวอย่างการสร้างโมดูลและฟังก์ชันในภาษา Erlang

ทุกฟังก์ชันในโปรแกรมจะเขียนไว้ในโมดูล เมื่อเขียนโมดูลเสร็จแล้วจะต้องคอมไพล์แล้วเรียกใช้งานผ่านเวอร์ชวลแมตชีนของตัวภาษา Erlang โดยการเรียกใช้งานฟังก์ชันในโมดูลต่าง ๆ จะใช้ชื่อโมดูล ตามด้วย : และชื่อฟังก์ชันที่ต้องการเรียกใช้งาน ดังรูปที่ 2.13 เป็นการคอมไพล์โมดูล main และเรียกใช้ฟังก์ชัน main() ของโมดูลนี้

```
Eshell V5.8.4  (abort with ^G)
1> c(main).
{ok,main}
2> main:main().
Hello World
ok
```

รูปที่ 2.13 แสดงตัวอย่างการนำโมดูลและฟังก์ชันมาใช้งาน

การเขียนโค้ดหรือคำสั่งในภาษาเชิงฟังก์ชันนั้น ตัวฟังก์ชันถือเป็น first-class object สามารถใช้เป็นข้อมูลให้กับฟังก์ชันอื่นได้ ตัวอย่างเช่น การเรียกใช้งานฟังก์ชัน lists:map/2 ซึ่งเป็นฟังก์ชันในโมดูล lists ที่ทำหน้าที่เรียกใช้งานฟังก์ชันที่ส่งเข้าไปเป็นพารามิเตอร์แรกให้กับฟังก์ชัน map แล้วรับข้อมูลแต่ละตัวของลิสต์ที่ถูกส่งเข้าไปเป็นพารามิเตอร์ที่สองของฟังก์ชันนั้น การเขียนฟังก์ชันนอกจากจะเขียนโดยใช้การตั้งชื่อฟังก์ชันแล้ว เรายังสามารถเขียนฟังก์ชันที่ไม่มีชื่อ (anonymous function)โดยเขียนให้อยู่ในรูปของ fun(Parameter) -> end ดังตัวอย่างในรูปที่ 2.14

```
Eshell V5.8.4  (abort with ^G)
1> lists:map (fun(L) -> L + 2 end, [1,2,3,4,5]).
[3,4,5,6,7]
```

รูปที่ 2.14 แสดงตัวอย่างการใช้ fun(Parameter) -> end และ lists:map

2.10.5 การทำแพตเทิร์นแมตชิ่ง (Pattern Matching)

แพตเทิร์นแมตชิ่งคือ กลไกในการเปรียบเทียบค่าของข้อมูลที่อยู่คนละรูปแบบ (pattern) ว่ามีค่าเท่ากันหรือไม่ Erlang มีความสามารถทำกลไกเรื่องแพตเทิร์นแมตชิ่งได้โดยเมื่อมีการใช้เครื่องหมาย = หรือเมื่อมีการเรียกใช้งานฟังก์ชันแล้วกำหนดค่าพารามิเตอร์ให้กับฟังก์ชัน จะมีการทำ แพตเทิร์นแมตชิ่งข้อมูล และกำหนดค่าให้กับตัวแปรในตำแหน่งที่เทียบเท่ากับค่าที่ส่งไป ตัวอย่างดังรูปที่ 2.15

```
Eshell V5.8.4  (abort with ^G)
1> A = [1,2,3,4].
[1,2,3,4]
2> [H | T] = A.
[1,2,3,4]
3> H.
1
4> T.
[2,3,4]
```

รูปที่ 2.15 ตัวอย่างการทำแพตเทิร์นแมตชิ่งกับข้อมูลลิสต์

2.11 การใช้ภาษา Erlang เพื่อเขียนโปรแกรมบนหน่วยประมวลผลแบบหลายแกนหลัก

ในภาษา Erlang นั้นจะมีฟังก์ชันที่เตรียมไว้สำหรับสร้างโปรเซสย่อย ๆ ให้ทำงานเป็นลักษณะหลาย ๆ โปรเซสพร้อมกันได้ คือฟังก์ชันที่ชื่อว่า spawn(Module, Function, Arguments) โดยฟังก์ชันนี้จะรับค่าพารามิเตอร์สามค่าคือ ชื่อโมดูลของฟังก์ชันที่จะถูกเรียกใช้ ชื่อฟังก์ชันในโมดูลที่จะถูกเรียกใช้ และ ลิสต์ของพารามิเตอร์ที่จะถูกส่งให้ฟังก์ชันนี้เมื่อมีการทำงาน เมื่อเรียกใช้ฟังก์ชัน spawn/3 ในโปรเซสหลัก จะเกิดการสร้างโปรเซสย่อยขึ้นมาใหม่แล้วการทำงานในโปรเซสนั้นจะเรียกใช้ฟังก์ชันตามที่ส่งไปเป็นพารามิเตอร์ให้กับฟังก์ชัน spawn หลังจากที่โปรเซสหลักใช้ฟังก์ชัน spawn แล้วจะได้ค่าที่เรียกว่า process identifier หรือใช้ชื่อย่อว่า PID ขึ้นมาเป็นค่าเฉพาะของแต่ละโปรเซสเพื่อใช้อ้างอิงและเพื่อให้สามารถสื่อสารระหว่าง โปรเซสหลักและโปรเซสย่อย

ในแต่ละโปรเซส จะไม่สามารถใช้หน่วยความจำร่วมกันได้ นั่นหมายถึงตัวแปรที่ถูกสร้างอยู่ในอีกโปรเซสหนึ่งไม่สามารถถูกอ้างถึงหรือเปลี่ยนแปลงค่าใด ๆ ได้จากโปรเซสอื่น แต่สามารถที่จะใช้กลไกการส่งข้อความระหว่างโปรเซส (message passing) เพื่อให้โปรเซสติดต่อและส่งข้อมูลไปมาระหว่างกันได้โดยจะใช้หมายเลข PID ที่ได้จากการเรียกใช้งาน spawn และใช้โอเปอเรเตอร์ส่งข้อมูล ! ตามด้วยข้อมูลที่จะส่งไปให้กับโปรเซสที่มีหมายเลข PID นั้น เช่น PID ! {msg, “hello”}. เพื่อส่งทูเพิล {msg, “hello”} ไปให้กับโปรเซสที่มีหมายเลขโปรเซสตรงกับค่าของตัวแปร

PID ซึ่งระบุไว้ด้านซ้ายของเครื่องหมาย ! ส่วนการรอรับข้อความที่ถูกส่งมาจากโปรเซสอื่น ๆ นั้นต้องใช้คำสั่งในรูปแบบของ receive..end ซึ่งในการประมวลผลคำสั่ง receive นั้นจะเป็นการทำแพตเทิร์นแมตชิ่งของข้อมูลที่ส่งมา ถ้าสามารถแมตชิ่งข้อมูลทางด้านซ้ายของเครื่องหมาย -> ได้ ก็จะทำงานตามโค้ดทางด้านขวาของเครื่องหมาย -> ถ้ามีแพตเทิร์นหลาย ๆ แบบแต่ละแพตเทิร์นจะต้องคั่นด้วยเครื่องหมาย ; ถ้าต้องการให้โปรเซสรอรับคำสั่งตลอดเวลา สามารถทำได้ด้วยการเขียนคำสั่งวนซ้ำในรูปแบบของ recursive function ตัวอย่างการใช้งาน spawn และการส่งและรับข้อมูลระหว่างโปรเซสตามรูปที่ 2.16

ภาษา Erlang ไม่จำเป็นต้องกำหนดว่าส่วนใดต้องทำงานแบบขนานและส่วนใดทำงานแบบเรียงลำดับ เพราะว่าเมื่อมีการทำงานแบบหลาย ๆ โปรเซส และนำโปรแกรมไปทำงานบนหน่วยประมวลผลกลางที่มีหลายแกนหลัก Erlang สามารถกระจายโปรเซสการทำงานไปทำงานบนแกนหลักแต่ละแกนได้เองโดยอัตโนมัติ

```
-module(echo).

-export(main/0).

%% Echo Message


main() ->
      PID = spawn(?MODULE, process_loop, []),

      PID ! {msg, "hello"},

      PID ! exit.

process_loop() ->

      receive

            {msg,Msg} ->

                  io:format("~w~n",[Msg]),

                  process_loop();

            exit -> ok

      end.
```

รูปที่ 2.16 ตัวอย่างการใช้ทำงานแบบหลายๆโปรเซสพร้อมกันของ Erlang และการส่งข้อมูลระหว่างโปรเซสแต่ละโปรเซส เมื่อส่งข้อความ {msg, "hello"} ไปที่โปรเสซหมายเลข PID แล้ว ภายในโปรเซสจะมีคำสั่ง receive รอจำแพทเทิร์น {msg, Msg} ซึ่งจะทำให้ตัวแปร Msg มีค่า "hello" แล้วจึงแสดงผลสตริง"hello" หลังจากนั้นตัวโปรเซสจะเรียกซ้ำฟังก์ชัน process_loop อีก ครั้งเพื่อรอรับข้อความอื่นต่อไป

2.12 สรุปงานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการแบ่งกลุ่มข้อมูล เป็นการนำเสนอวิธีการแบ่งกลุ่มข้อมูลด้วยอัลกอริทึมแบบต่าง ๆ เช่น K-Means Clustering, K-Medoids Clustering และการนำทฤษฎีราฟเซตมาช่วย โดยบางงานวิจัยนำเสนอโดยไม่ได้กล่าวถึงเทคนิคในการพัฒนา บางงานวิจัยเลือกพัฒนาโดยเขียนโปรแกรมแบบขนานด้วยวิธีการที่แตกต่างกัน โดยสรุปเทคนิคของงานวิจัยที่ผ่านมากับสิ่งที่จะทำในวิทยานิพนธ์ฉบับนี้ได้ดังตารางที่ 2.3

Kittisak Kerdprasop และ Nittaya Kerdprasop (2010) ได้ศึกษาวิธีการพัฒนาโปรแกรมแบบขนานที่ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักด้วยภาษา Erlang และนำเสนอวิธีการพัฒนาการจัดกลุ่มข้อมูลด้วยอัลกอริทึม K-Means Clustering ที่ใช้การประมวลผลแบบขนานด้วยภาษา Erlang โดยได้เลือกสถาปัตยกรรมแบบ Single Instruction Multiple Data (SIMD) ในการออกแบบโดยแบ่งข้อมูลออกเป็นกลุ่มย่อย ๆ ขนาดเท่า ๆ กัน แล้วส่งข้อมูลย่อยให้กับแต่ละโปรเซสเพื่อคำนวณระยะห่างและจัดกลุ่มให้กับข้อมูลว่าจะอยู่กลุ่มไหน หลังจากนั้นนำผลลัพธ์ที่ได้มาหาค่า means ของแต่ละกลุ่มอีกครั้งหนึ่ง นอกจากนี้ยังได้นำเสนอวิธีการแบบ Approximate Parallel K-Means Clustering สำหรับการจัดกลุ่มข้อมูลที่มีขนาดใหญ่และมีลักษณะเป็น Streaming การทดลองได้ทำการทดสอบกับจำนวนข้อมูลที่มีขนาดต่าง ๆ กันตั้งแต่ 50 จนถึง 900,000 แล้วเปรียบเทียบเวลาการทำงานของแบบขนานและแบบลำดับ พบว่าเมื่อจำนวนข้อมูล 50,000 เวลาการทำงานในแบบขนานเร็วกว่าแบบลำดับถึง 40.47 เปอร์เซ็น

Honggang Wang, Jide Zhao, Hongguang Li และ Jianguo Wang (2008) ได้นำเสนอการจัดกลุ่มข้อมูลสำหรับงานด้านการประมวลผลภาพ บนหน่วยประมวลผลกลางแบบหลายแกนหลัก โดยได้ศึกษาอัลกอริทึม K-Means Clustering และ Mean-Shift และใช้สถาปัตยกรรมแบบ SIMD ในการประมวลผลแบบขนาน การทดสอบได้เลือกทดสอบกับข้อมูล mailing list สำหรับอัลกอริทึม K-Means clustering และทดสอบกับข้อมูลรูปภาพสำหรับอัลกอริทึม Mean-Shift พบว่าประสิทธิภาพของการพัฒนาบนหน่วยประมวลผลกลางแบบหลายแกนหลักเพิ่มขึ้นตามจำนวนของแกนหลักและหน่วยประมวลผลที่มี เช่นเมื่อใช้ 4 หน่วยประมวลผล ประสิทธิภาพการทำงานเร็วขึ้นถึง 2.89 วินาที

Reza Farivar, Daniel Rebolledo, Ellick Chan และ Roy Campbell (2008) ได้นำเสนอการจัดกลุ่มข้อมูลด้วยอัลกอริทึม K-Means Clustering แบบขนานบนหน่วยประมวลผลกลางสำหรับการแสดงผลกราฟิก ซึ่งใช้เครื่องมือในการพัฒนาที่ชื่อว่า CUDA บนการ์ดประมวลผลกราฟิกยี่ห้อ NVIDIA ตระกูล G80 จากผลการทดสอบกับการ์ดรุ่น NVIDIA 8600GT เมื่อเปรียบเทียบกับการประมวลผลแบบธรรมดาบนหน่วยประมวลผลกลางรุ่น Intel Pentium D 3.0 GHz พบว่าเร็วขึ้นถึง 13.57 เท่า และ เมื่อเปลี่ยนเป็นการ์ดรุ่น NVIDIA 8800 Ultra GTX ประมวลผลได้เร็วขึ้นถึง 68 เท่า

Georg Peters และ Martin Lampart (2006) ได้นำเสนอวิธีการนำทฤษฎีราฟเซต มาประยุกต์ใช้กับอัลกอริทึม K-Medoids Clustering โดยงานวิจัยได้กล่าวถึงอัลกอริทึม K-Means Clustering อัลกอริทึม Rough K-Means Clustering วิธีวัดประสิทธิภาพของการจัดกลุ่มข้อมูลที่ชื่อว่า Davies-Bouldin index การจัดกลุ่มข้อมูลแบบ K-Means Clustering และ การจัดกลุ่มข้อมูลด้วยอัลกอริทึมแบบ K-Medoid Clustering และนำเสนอวิธีการ Rough K-Medoids Clustering ในการ

ทดลองได้ทำการทดสอบกับข้อมูล 4 ชุดด้วยกันได้แก่ ข้อมูล ที่สังเคราะห์ขึ้นเอง, ข้อมูล Colon Cancer, ข้อมูล Forest และ ข้อมูล Control Chart โดยใช้ Rough K-Means Clustering เปรียบเทียบกับ Rough K-Medoids Clustering และวัดประสิทธิภาพโดยใช้วิธีการ Davies-Bouldin index ซึ่งพบว่า Rough K-Medoids Clustering ทำได้ดีกว่า โดยมีบางกรณีที่ Rough K-Means Clustering ทำได้ดีกว่า แต่เมื่อเปลี่ยนวิธีการเช็คค่า Compactness ของ Rough K-Medoids Clustering ไปเป็น Davies-Bouldin Index พบว่า K-Medoids Clustering มีประสิทธิภาพดีกว่าทุกการทดสอบ

Gregor von Laszewski และ Douglas Roberts (2009) ได้นำเสนอวิธีการจัดกลุ่มข้อมูลด้วยอัลกอริทึม K-Medoids Clustering โดยนำเทคนิคการพัฒนาโปรแกรมแบบขนานบนหน่วยกลางสำหรับการประมวลผลกราฟิกที่เรียกกว่า CUDA มาใช้โดยทำการแบ่งข้อมูลออกเป็น block ซึ่งในแต่ละ block จะประกอบด้วยหลาย ๆ thread ในแต่ละ thread จะเป็นการคำนวณระยะห่างของข้อมูลกับตำแหน่ง medoid ของแต่ละกลุ่ม โดยการทดสอบได้ทดสอบกับข้อมูล 1 มิติ ซึ่งเวลาการทำงานในแบบขนานเมื่อจำนวนกลุ่มเป็น 64 กลุ่มทำงานได้เร็วกว่าแบบลำดับถึง 7.76 เท่า

ตารางที่ 2.3 สรุปเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับการจัดกลุ่มข้อมูลและพัฒนาด้วยการเขียนโปรแกรมแบบขนานบทความวิจัยที่เกี่ยวข้องประกอบด้วย 1 = Kittisak Kerdprasop และ Nittaya Kerdprasop (2010), 2 = Reza Farivar, Daniel Rebolledo, Ellick Chan และ Roy Campbel (2008), 3 = Gregor von Laszewski และ Douglas Roberts (2009), 4 = Honggang Wang, Jide Zhao, Hongguang Li และ Jianguo Wang (2008), 5 = Georg Peters และ Martin Lampart, 6 = การจัดกลุ่มข้อมูลด้วยเทคนิคราฟเคมีดอยส์แบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก (งานวิจัยของวิทยานิพนธ์ฉบับนี้)

กระบวนการทำงาน	งานวิจัยที่เกี่ยวข้อง					
	1	2	3	4	5	6
อัลกอริทึมที่ใช้						
K-Means	/	/		/		
K-Medoids			/			
Mean-Shift				/		
Approximate K-Means	/					
Rough K-Means					/	
Rough K-Medoids					/	/
เทคนิคการประมวลผลแบบขนาน						
Erlang	/					/
CUDA		/	/			
OpenMP				/		
Objective criterion						
Rough Compactness of the Clustering					/	/
Davies-Bouldin Index					/	
PAM			/			

บทที่ 3
วิธีดำเนินการวิจัย

ในบทนี้จะนำเสนอถึง วิธีการวิจัย เครื่องมือที่ใช้ในการวิจัย และกระบวนการต่าง ๆ ของการวิจัย โดยมีรายละเอียดดังนี้

3.1 วิธีวิจัย

ในการวิจัยนี้ จะมีวิธีการในการดำเนินการแยกออกเป็นขั้นตอนต่าง ๆ ดังนี้

3.1.1 ศึกษาวิธีการจัดกลุ่มข้อมูลด้วยอัลกอริทึมแบบต่าง ๆ

ในการจัดกลุ่มข้อมูลนั้นสามารถทำได้หลายวิธีด้วยกัน แต่ละแบบนั้นมีข้อดีข้อเสียแตกต่างกันขึ้นอยู่กับลักษณะของข้อมูลที่นำมาวิเคราะห์ ในงานวิจัยนี้ได้ศึกษาวิธีการจัดกลุ่มข้อมูลจากงานวิจัยที่ผ่านมา โดยเลือกศึกษาอัลกอริทึมที่เกี่ยวข้องกับงานวิจัยของวิทยานิพนธ์ดังต่อไปนี้

1) K-Means clustering

เป็นอัลกอริทึมในการจัดกลุ่มข้อมูลที่แบ่งข้อมูลออกเป็นส่วน ๆ อย่างชัดเจน โดยอาศัยการวัดระยะห่างระหว่างข้อมูลกับตำแหน่ง means ของแต่ละกลุ่ม โดยที่ค่า means คือค่าเฉลี่ยของข้อมูลที่อยู่ในกลุ่ม ข้อดีของ K-Means clustering คือง่ายในการพัฒนาโปรแกรม แต่ผลลัพธ์ที่ได้นั้นสามารถเกิดผลกระทบจากการมี outliers ได้ง่าย

2) K-Medoids clustering

เป็นการจัดกลุ่มข้อมูลโดยวัดระยะห่างข้อมูลเทียบกับตำแหน่ง medoids ซึ่งวิธีการนี้จะใช้ตำแหน่งของข้อมูลจริง ๆ เป็นค่ากลางหรือ medoids ของกลุ่ม ในแต่ละกลุ่มยังคงแยกข้อมูลจากกันอย่างชัดเจนเช่นเดียวกับ K-Means clustering ข้อมูลไม่สามารถอยู่ในหลายกลุ่มได้ ข้อดีของ K-Medoids clustering คือได้รับผลกระทบจาก outliers ได้น้อยกว่าเพราะใช้ข้อมูลจริง

แทนตำแหน่งกลางของกลุ่ม ข้อเสียคือใช้เวลาคำนวณระยะห่างนานและพัฒนาได้ยากกว่า K-Means clustering เพราะว่าขั้นตอนในการหาข้อมูลที่จะเป็น medoids ใหม่ต้องวนรอบเทียบกับข้อมูลทุก ๆ ตัวในกลุ่ม

3) Rough K-Means clustering

ในอัลกอริทึมนี้จะนำทฤษฎี ราฟเซต มาประยุกต์ใช้กับการจัดกลุ่มแบบ K-Means clustering โดยมองว่าการจัดกลุ่มที่กำหนดให้แต่ละข้อมูลอยู่ได้เพียงหนึ่งกลุ่มนั้นตามความเป็นจริงของข้อมูลอาจจะสามารถมีลักษณะหลาย ๆ อย่างที่สามารถอยู่ได้มากกว่าหนึ่งกลุ่ม วิธีการนี้จึงได้นำหลักการของ ราฟเซต มาใช้โดยทำการแบ่งกลุ่มออกเป็นสองพื้นที่ด้วยกันเรียกว่า lower approximation และ upper approximation ซึ่งข้อมูลที่สามารถแบ่งเข้ากลุ่มได้อย่างชัดเจนจะอยู่ในส่วน lower approximation ส่วนข้อมูลที่สามารถอยู่ได้หลาย ๆ กลุ่มจะอยู่ในส่วนขอบของกลุ่มซึ่งคือพื้นที่ขอบด้านนอก lower approximation แต่ไม่เกิน upper approximation

4) Rough K-Medoids clustering

วิธีการนี้นำทฤษฎี ราฟเซต มาประยุกต์ใช้กับการแบ่งข้อมูลแบบ K-Medoids clustering ซึ่งจะแบ่งข้อมูลออกเป็นสองส่วนเช่นเดียวกันกับ Rough K-Means clustering เพียงแต่ว่าในการคำนวณจะใช้ตำแหน่ง medoids เป็นค่ากลางแทนที่จะใช้ค่า mean เป็นค่ากลาง

3.1.2 ศึกษาการประมวลผลแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก

ศึกษาการประมวลผลแบบขนานเพื่อนำมาออกแบบอัลกอริทึมสำหรับการจัดกลุ่มข้อมูลแบบ Rough K-Medoids clustering จากการศึกษางานวิจัยที่ผ่านมาพบว่ารูปแบบสถาปัตยกรรมในการทำงานแบบขนานที่นิยมใช้กันก็คือแบบ Single Instruction Multiple Data (SIMD) ที่มีการแบ่งข้อมูลออกเป็นหลาย ๆ กลุ่มแล้วส่งการทำงานหนึ่งคำสั่งที่สามารถนำไปทำไปพร้อม ๆ กันได้บนหน่วยประมวลผลหลายๆตัว ซึ่งเทคโนโลยีทางด้านอุปกรณ์คอมพิวเตอร์สามารถนำหน่วยประมวลผลมาต่อกันเพื่อให้ทำงานตามสถาปัตยกรรมนี้ได้ โดยใช้หลาย ๆ เครื่องติดต่อกันผ่านทางระบบเน็ตเวิร์ก เพราะว่าในอดีตยังไม่ได้มีการพัฒนาให้วงจรของหน่วยประมวลผลกลางมีแกนหลักในการประมวลผลหลายแกนแบบในปัจจุบัน เมื่อมีการพัฒนาหน่วยประมวลผลกลางแบบที่มีหลายแกนหลักขึ้นมา ทำให้เกิดการพัฒนาโปรแกรมแบบขนานประมวลผลได้โดยเครื่องคอมพิวเตอร์ส่วนบุคคลทั่วไป ไม่จำเป็นต้องเชื่อมต่อคอมพิวเตอร์หลายเครื่องแบบในอดีต การพัฒนาโปรแกรมแบบขนานจึงช่วยให้สามารถใช้งานหน่วยประมวลผลกลางแบบหลายแกนหลักได้

อย่างมีประสิทธิภาพมากที่สุดเท่าที่จะเป็นไปได้ ซึ่งวิธีการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลักที่ทำการศึกษาพบว่าใช้เทคโนโลยีดังต่อไปนี้

1) Erlang

Erlang เป็นภาษาแบบ functional language ที่สามารถแยกการประมวลผลออกเป็นหลายๆโปรเซส การทำงานของโปรแกรมภาษา Erlang จะอาศัยเครื่องจักรเสมือนหรือ Virtual Machine(VM) ในการประมวลผลซึ่ง VM ของ Erlang นั้นสามารถจะส่งการทำงานของโปรเซสที่แยกกันทำงานให้กระจายไปบนแกนหลักของหน่วยประมวลผลแบบหลายแกนได้เองอัตโนมัติ ทำให้ง่ายในการพัฒนา ไม่จำเป็นต้องเขียนเจาะจงว่าจะให้ทำงานกี่แกนหลักหรือไม่จำเป็นต้องมีคำสั่งพิเศษ หรือ library พิเศษเพิ่มเข้ามาในภาษา

2) OpenMP

เป็นเทคโนโลยีในการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนโดยใช้ภาษาซีในการพัฒนาซึ่งต้องเพิ่ม library ของ OpenMP เขามาด้วยในขั้นตอนการแปลโค้ดโปรแกรม วิธีการใช้งานของ OpenMP จะแยกการทำงานออกเป็นเทรด(thread)ย่อย ๆ โดยโปรแกรมเมอร์ไม่จำเป็นต้องเขียนโค้ดกำหนดว่าจะให้การทำงานเกิดขึ้นบนเทรดไหน แต่จะต้องกำหนดมาโครเพื่อบอกว่าการทำงานส่วนไหนที่ต้องการให้ทำงานแบบขนาน การกระจายการทำงานบนหลายเทรดจะจะเป็นหน้าที่ของส่วน OpenMP ซึ่งทำงานโดยอัตโนมัติ

3) CUDA

CUDA เป็นเทคโนโลยีในการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางสำหรับการประมวลผลกราฟิกที่มีหลายแกนหลัก ซึ่งเป็นเทคโนโลยีของบริษัท NVIDIA โดยสถาปัตยกรรมของการ์ดกราฟิกนี้จะแบ่งข้อมูลออกเป็น block ในแต่ละ block จะประกอบด้วย เทรดสำหรับการทำงานแบบขนานและเมื่อเรากระจายข้อมูลลงไปบน block ต่าง ๆ เทรดแต่ละเทรดก็จะประมวลผลโดยเลือกข้อมูลใน block นั้น ๆ มาประมวลผล

3.1.3 ออกแบบอัลกอริทึม Rough K-Medoids clustering สำหรับการประมวลผลแบบขนาน

ในงานวิจัยนี้พัฒนาอัลกอริทึม Rough K-Medoids clustering ที่ประมวลผลได้ทั้งในแบบขนานและแบบลำดับ เพื่อให้สามารถทดสอบประสิทธิภาพการทำงานบนเครื่องคอมพิวเตอร์ที่มีหลายแกนหลัก รายละเอียดของอัลกอริทึมปรากฏในหัวข้อ 3.2

3.1.4 ทดสอบการทำงานและประเมินผล

ในการทดสอบและประเมินผลจะทดสอบกับข้อมูลทดสอบจำนวนสี่ชุดด้วยกัน ได้แก่

- ข้อมูลที่สุ่มขึ้นมา
- ข้อมูล Conlon Cancer
- ข้อมูล Forest Data
- ข้อมูล Control Chart Data

โดยทำการทดสอบระหว่างอัลกอริทึมที่เป็นแบบลำดับทำงานแค่ภายในหนึ่งโปรเซส กับอัลกอริทึมแบบขนานทำงานแบบหลายโปรเซสแยกกันบนหน่วยประมวลผลกลางแบบหลายแกนหลัก

3.2 การออกแบบอัลกอริทึม Parallel Rough K-Medoids clustering

3.2.1 ส่วนทำงานหลัก

การประมวลผลแบบขนานเป็นการพยายามที่จะลดเวลาการทำงานของโปรแกรม โดยการแยกส่วนการทำงานที่สามารถประมวลผลไปพร้อมกัน ภายในหนึ่งโปรแกรมอาจจะมีทั้งส่วนที่สามารถแยกประมวลผลแบบขนานได้ และบางส่วนยังจำเป็นต้องประมวลผลแบบลำดับ เนื่องจากลำดับการทำงานมีผลต่อผลลัพธ์ที่ต้องการ

ในส่วนอัลกอริทึม Parallel Rough K-Medoids clustering เป็นการประยุกต์ใช้การประมวลผลแบบขนานกับอัลกอริทึม Rough K-Medoids clustering ซึ่งเมื่อวิเคราะห์จุดที่สามารถทำการประมวลผลแบบขนานได้คือ ขั้นตอนการสลับค่าระหว่างค่ากลางปัจจุบันกับวัตถุอื่นที่ไม่ใช่ค่ากึ่งกลางปัจจุบันเพื่อหาค่า RCPC ที่น้อยที่สุดว่าเกิดจากการสลับค่ากลางกลุ่มไหนกับวัตถุตัวใด

จากรูปที่ 3.1 แสดงอัลกอริทึม Parallel Rough K-Medoids clustering แบ่งออกเป็น 4 ขั้นตอนหลักขั้นตอนที่ 1 ถึง 3 จะทำงานแบบลำดับ ส่วนขั้นตอนที่ 4 จะประยุกต์ใช้การทำงานแบบขนาน แผนภาพแสดงลำดับการทำงานเป็นไปดังรูปที่ 3.2

3.2.2 ส่วนรับไฟล์อินพุต

ส่วนประกอบย่อยส่วนแรกจะเป็นส่วนรับไฟล์อินพุต โดยค่าที่อ่านได้จากไฟล์อินพุต และจากพารามิเตอร์มีค่าต่าง ๆ ดังนี้

- ค่าของจุดข้อมูลที่อ่านจากไฟล์ข้อมูล กำหนดรูปแบบของข้อมูลในไฟล์ให้แต่ละบรรทัดแทนข้อมูลหนึ่งเรคคอร์ด และ ข้อมูลของแต่ละมิติคั่นด้วยเครื่องหมาย "," ตัวอย่างดังรูปที่ 3.3 ส่วนกระบวนการในการอ่านไฟล์แสดงได้ดังรูปที่ 3.4
- จำนวนของกลุ่มที่ต้องการแบ่ง
- ค่าคงที่ w_l
- ค่าคงที่ w_b
- ค่าคงที่ ε
- จำนวนของโปรเซสที่ต้องการแบ่ง

หลังจากอ่านอินพุตทั้งหมดแล้วจะทำการสุ่มเลือกข้อมูลมาเป็นค่า medoids ตามจำนวนกลุ่มที่ต้องการ

3.2.3 ส่วนการแบ่งข้อมูลทดสอบ

ส่วนนี้จะทำหน้าที่นำข้อมูลที่อ่านจากไฟล์เรียบร้อยแล้วและแยกส่วนที่เป็น medoids เริ่มต้น กับส่วนที่เหลือเรียบร้อยแล้ว มาจัดกลุ่มโดยสำหรับขบวนการแบบ Rough K-Medoids clustering นั้นจะมี 2 ขั้นตอนย่อย ๆ คือ

- ขั้นตอนจัดเข้ากลุ่มที่ใกล้ที่สุดกำหนดให้เป็นส่วนของ lower approximation ของกลุ่มนั้น (ตามนิยามจะเป็น upper ด้วยเพราะ lower เป็นเซตย่อยของ upper)
- หลังจากนั้นจะนำข้อมูลของแต่ละกลุ่มที่แบ่งในขั้นตอนแรก ไปเปรียบเทียบระยะกับ medoids ของกลุ่มอื่นอีกครั้ง ถ้าระยะทางระหว่างข้อมูลกับ medoid ของกลุ่มที่ใกล้ที่สุด กับ medoid ของกลุ่มอื่นห่างกันไม่เกินค่า ε ที่กำหนด จะกำหนดใช้ข้อมูลนี้เป็นข้อมูลในส่วนของ upper approximation ของทุกกลุ่มที่เข้าเงื่อนไขนี้และไม่ได้อยู่ในส่วนของ lower approximation ของกลุ่มใด ๆ

กระบวนการแบ่งข้อมูลทดสอบมีลำดับการทำงานดังรูปที่ 3.5

3.2.4 ส่วนการหาตำแหน่ง Medoids ใหม่

ขั้นตอนวิธีของ Rough K-Medoids clustering ปกตินั้นจะเสียเวลากับขั้นตอนนี้เป็นอย่างมากเนื่องจากต้องทำการสลับค่าระหว่างข้อมูลที่เป็นตำแหน่ง medoids ของแต่ละกลุ่มกับข้อมูลที่ไม่ใช่ตำแหน่ง medoids ทั้งหมดเพื่อที่จะเปรียบเทียบว่าค่า RCPC ของก่อนและหลังสลับอันไหนดีกว่ากัน และ เมื่อสลับทั้งหมดแล้วอันไหนดีที่สุด ซึ่งงานวิจัยนี้ได้เลือกที่จะออกแบบกระบวนการตรงนี้ให้ทำงานแบบขนานโดยทำการแบ่งข้อมูลที่ไม่ใช่ medoids ที่ต้องการสลับออกเป็นกลุ่ม ๆ ตามจำนวนโปรเซสที่ต้องการ หลังจากนั้นให้แต่ละโปรเซสทำการสลับค่าของข้อมูลที่ไม่ใช่ medoids ที่ได้รับการแบ่งแล้วหาตำแหน่งที่ใช้ค่า RCPC ดีที่สุดภายในกลุ่มข้อมูลที่ได้รับ แผนภาพแสดงขั้นตอนการเปรียบเทียบหาตำแหน่งที่ให้ค่า RCPC ดีที่สุดแสดงดังรูปที่ 3.6 แผนภาพการแบ่งโปรเซสแสดงดังได้รูปที่ 3.7 เมื่อแต่ละโปรเซสทำงานจบแล้วเราจะนำข้อมูลที่ดีที่สุดของแต่ละกลุ่มมาทำการเทียบกันอีกครั้งเพื่อหาตำแหน่งที่สลับที่ดีที่สุด ถ้าทำการสลับจนครบทุกตัวแล้วได้ค่า medoids เป็นตำแหน่งเดิมทั้งหมด จะถือว่าสิ้นสุดการแบ่งกลุ่มแบบ Rough K-Medoids clustering

ตัวอย่างการทำงานของขั้นตอนนี้เช่น สมมติมีข้อมูล 2 มิติอยู่ 6 ตัวด้วยกันได้แก่ {(1, 2), (27, 32), (30, 40), (25, 50), (45, 10), (10, 9)} กำหนดจำนวนกลุ่มที่ต้องการแบ่งสองกลุ่ม (K = 2), ค่าคงที่ ε = 0.5, ค่าคงที่ w_l = 0.8, ค่าคงที่ w_b = 0.2 และจำนวนโปรเซสที่ต้องการแบ่งในขั้นตอนนี้คือ 2 โปรเซส (P = 2) ลำดับการทำงานตามอัลกอริทึมเป็นดังต่อไปนี้

- เลือก (1, 2) กับ (27, 32) เป็นค่ากึ่งกลางเริ่มต้นของกลุ่มที่ 1 และ กลุ่มที่ 2 แทนด้วย m_1 และ m_2 ตามลำดับ
- ทำการจัดกลุ่มโดยดูจากระยะห่างของวัตถุที่เหลือกับค่ากึ่งกลาง m1 และ m2 โดยแสดงค่าระยะห่างตามตารางที่ 3.1 จะได้
  กลุ่มที่ 1 มีสมาชิกคือ {(1, 2), (10, 9)}
  กลุ่มที่ 2 มีสมาชิกคือ {(27, 32), (30, 40), (25, 50), (45, 10)}
  โดยทั้งสองกลุ่มมีเฉพาะส่วนที่เป็น lower approximation ไม่มีส่วน boundary region
- ทำการคำนวณค่าปัจจุบันได้ $RCPC_{current}$ = 53.18
- สร้างโปรเซสย่อย 2 โปรเซส คือ P_1 และ P_2 ตามลำดับโดยแบ่งข้อมูลให้ P_1 และ P_2 ดังนี้
  ข้อมูลสำหรับ P_1 คือ {(30, 40), (25, 50)}
  ข้อมูลสำหรับ P_2 คือ {(45, 10), (10, 9)}

- กระบวนการสลับตำแหน่ง medoids ใน P_1 และค่า RCPC ที่คำนวณได้

  m_1 สลับกับ (30, 40) ได้ค่า RCPC = 73.58

  m_1 สลับกับ (25, 50) ได้ค่า RCPC = 71.62

  m_2 สลับกับ (30, 40) ได้ค่า RCPC = 44.90

  m_2 สลับกับ (25, 50) ได้ค่า RCPC = 52.00

- กระบวนการสลับตำแหน่ง medoids ใน P_2 และค่า RCPC ที่คำนวณได้

  m_1 สลับกับ (45, 10) ได้ค่า RCPC = 87.19

  m_1 สลับกับ (10, 9) ได้ค่า RCPC = 44.06

  m_2 สลับกับ (45, 10) ได้ค่า RCPC = 71.73

  m_2 สลับกับ (10, 9) ได้ค่า RCPC = 101.57

- เมื่อนำค่าที่น้อยที่สุดของแต่ละโปรเซสมาเปรียบเทียบกันจะพบว่าการสลับระหว่าง m_1 และข้อมูล (10, 9) จะได้ค่า RCPC น้อยที่สุดจึงทำการเปลี่ยนค่ากึ่งกลางของกลุ่มที่หนึ่งเป็น m_1 = (10, 9) และ ค่ากึ่งกลางของกลุ่มที่สองยังคงเดิมคือ m_2 = (27, 32)

- ทำการจัดกลุ่มอีกครั้งโดยดูจากระยะห่างของวัตถุที่เหลือกับค่ากึ่งกลาง m1 และ m2 โดยแสดงค่าระยะห่างตามตารางที่ 3.2 จะได้

  กลุ่มที่ 1 มีสมาชิกคือ {(10, 9), (1, 2)}

  กลุ่มที่ 2 มีสมาชิกคือ {(27, 32), (30, 40), (25, 50), (45, 10)}

- พบว่าสมาชิกในแต่ละกลุ่มไม่เปลี่ยนแปลง จึงจบการทำงาน

ตารางที่ 3.1 ระยะห่างระหว่างค่ากึ่งกลางกับวัตถุในรอบแรกของการทำงาน

วัตถุ	ระยะห่างกับ $m_1 = (1, 2)$	ระยะห่างกับ $m_2 = (27, 32)$	ใกล้กับ	ความต่างระยะห่างของ m_1 และ m_2
(30, 40)	47.80	8.54	m_2	39.26
(25, 50)	53.67	18.11	m_2	35.56
(45, 10)	44.72	28.43	m_2	16.29
(10,9)	11.40	28.60	m_1	17.20

ตารางที่ 3.2 ระยะห่างระหว่างค่ากึ่งกลางกับวัตถุในรอบที่สองของการทำงาน

วัตถุ	ระยะห่างกับ $m_1 = (10, 9)$	ระยะห่างกับ $m_2 = (27, 32)$	ใกล้กับ	ความต่างระยะห่างของ m_1 และ m_2
(30, 40)	36.89	8.54	m_2	28.35
(25, 50)	43.66	18.11	m_2	25.5
(45, 10)	30.01	28.43	m_2	1.58
(1,2)	11.40	39.70	m_1	28.30

Algorithm: Parallel Rough K-Medoids clustering

//Input : Data in file, K(number of clusters), w_l, w_b, ε, P(number of processes)

//Output: Set of cluster

1. Select K data objects randomly as medoids: m_k, k = 1, …, K. They belong to the lower approximation of the set they are medoids of: $m_k \in \underline{C_k}$. The remaining data objects are denoted as X'_m, m = 1, …, (N - K).

2. Assign the remaining (N – K) data objects X'_m to the K clusters in two step process. In the first step a data object is assigned to the upper approximation of the cluster to which it is closest. In the second step the data object is assigned to the upper approximation of further reasonably close clusters or it is assigned to the lower approximation of the closest cluster. The detains are as follows:

(i) For a given data object X'_m determine its closest medoid m_k:

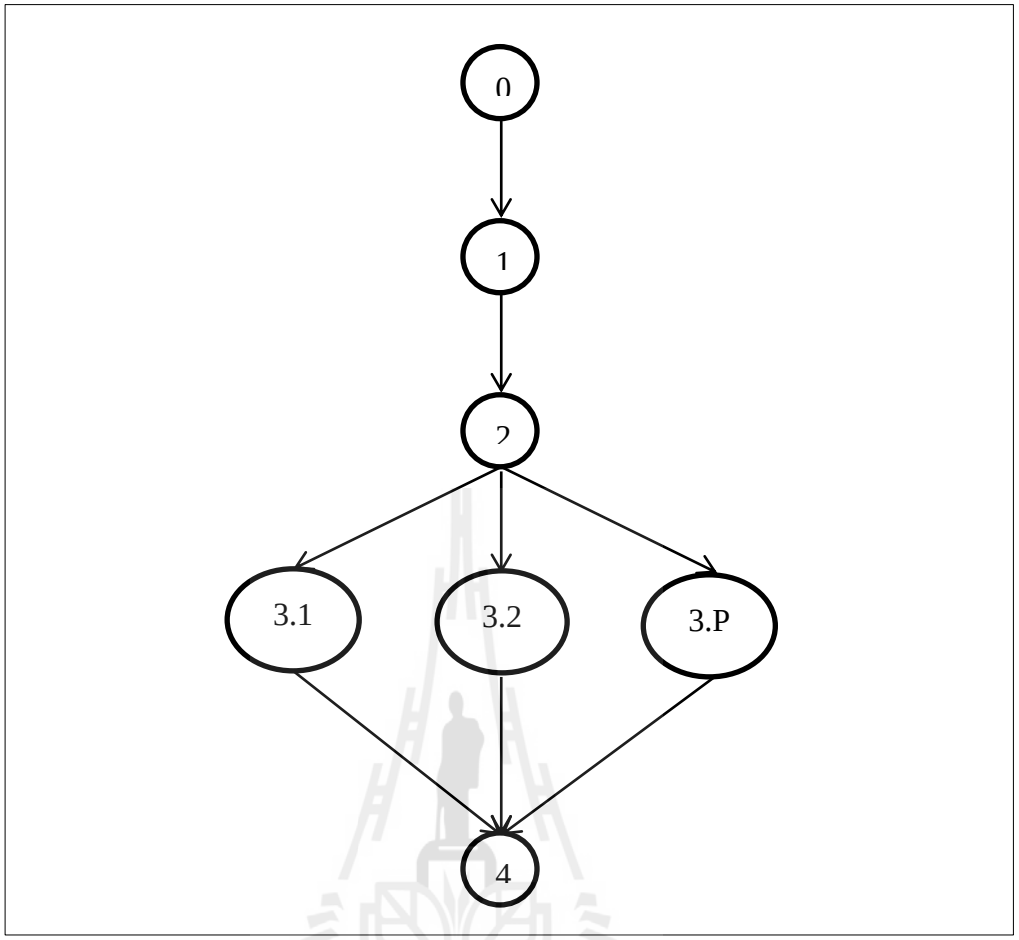
$d(X'_m, m_k) = \min_{h=1,\ldots,K} d(X'_m, m_h)$. Assign X'_m to the upper approximation of the cluster k: $X'_m \in \overline{C_k}$

(ii) Determine the clusters C_h that are also close to X'_m they are not farther away from X'_m than $d(X'_m, m_k) + \varepsilon$ where ε is a given threshold:
$T = \{h : d(X'_m, m_h) - d(X'_m, m_k) \le \varepsilon \wedge h \ne k\}$

3. create P processes which each process swap every medoid m_k with N/P data object and return data object and $RCPC_{k_0<->m_0}$ minimum. After all processes finish job calculate minimum of all process

4. swap medoid m_{k_0} and data object X_{m_0} if convergence STOP else go back to step 2

รูปที่ 3.1 อัลกอริทึม Parallel Rough K-Medoids clustering



รูปที่ 3.2 กราฟแสดงลำดับการทำงาน ส่วนที่ทำแบบลำดับ และ ส่วนที่ทำแบบขนาน ซึ่งขั้นตอนที่ 1, 2, 4 ตามรูปที่ 3.1 เป็นการทำงานแบบลำดับ ส่วนขั้นตอนที่ 3 เป็นการทำงานแบบขนานโดยแบ่งออกเป็น P โปรเซส

3841.780730,4164.162069

7504.314753,7909.953555

4847.181458,4750.496176

3186.546748,8721.028747

4584.364095,9156.071246

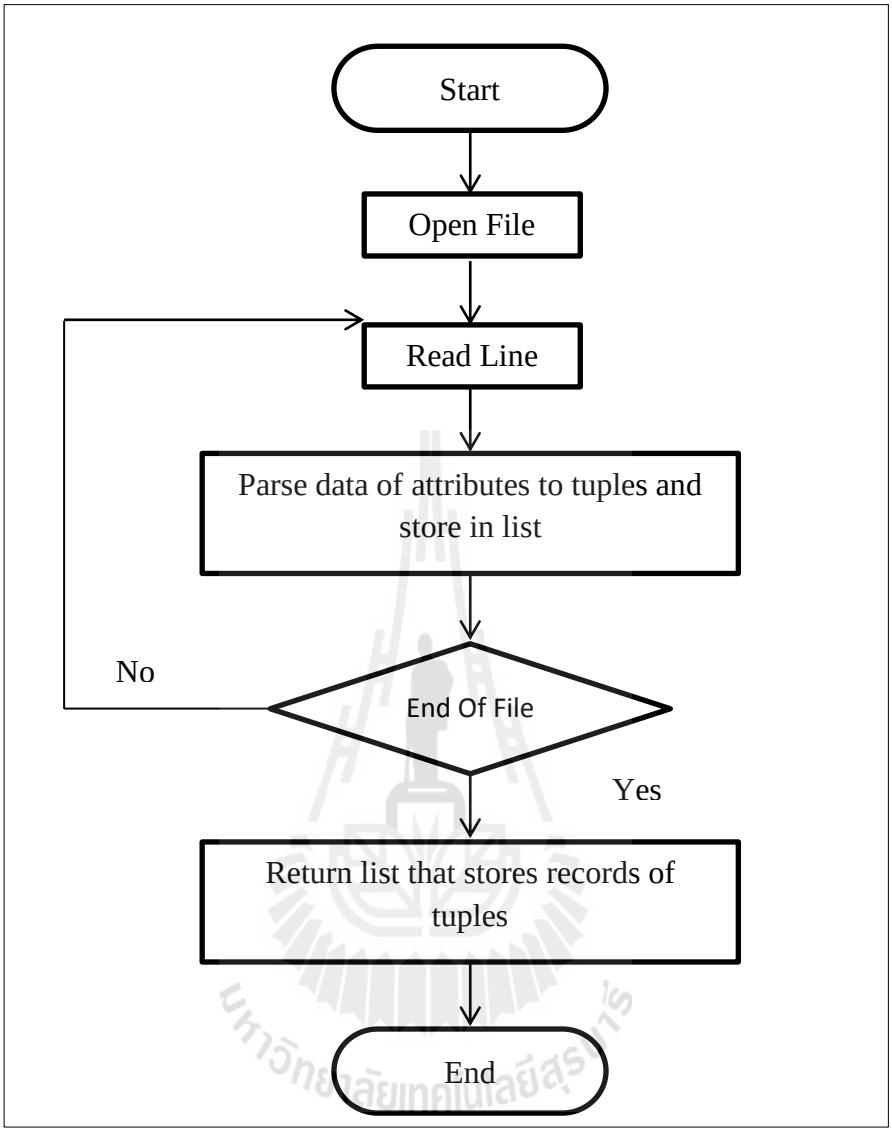
2862.519347,5055.664578

1185.839238,5491.177889

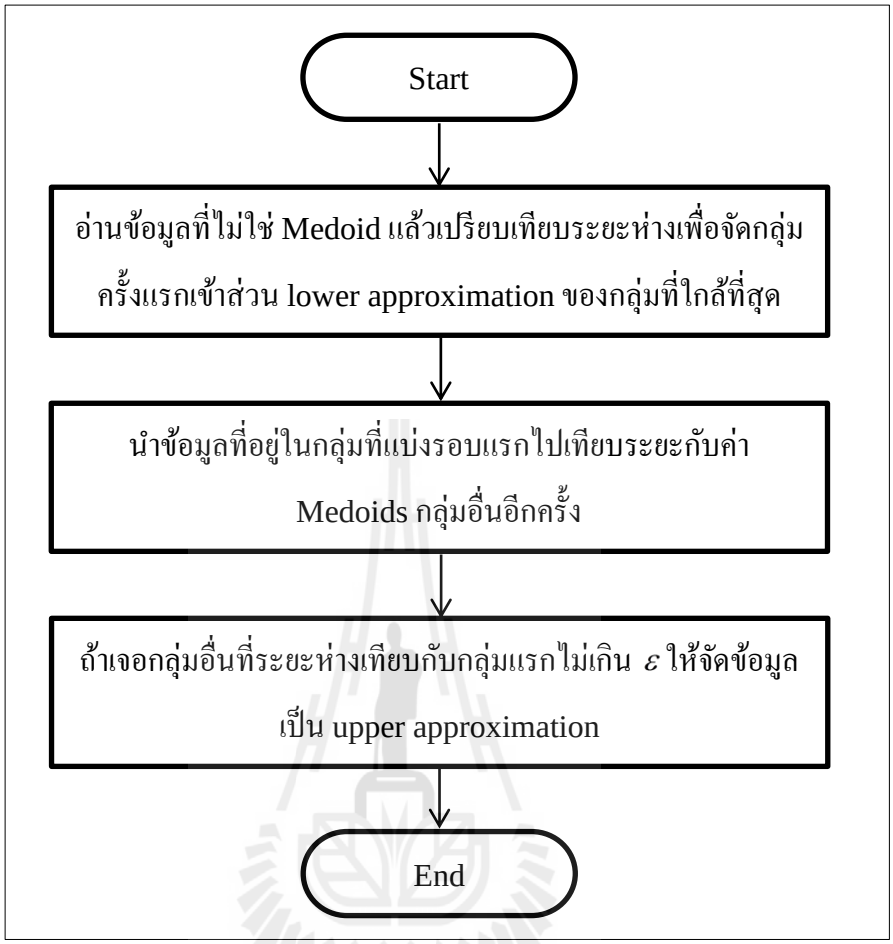
4861.185547,5336.554382

7910.086225,7975.786519

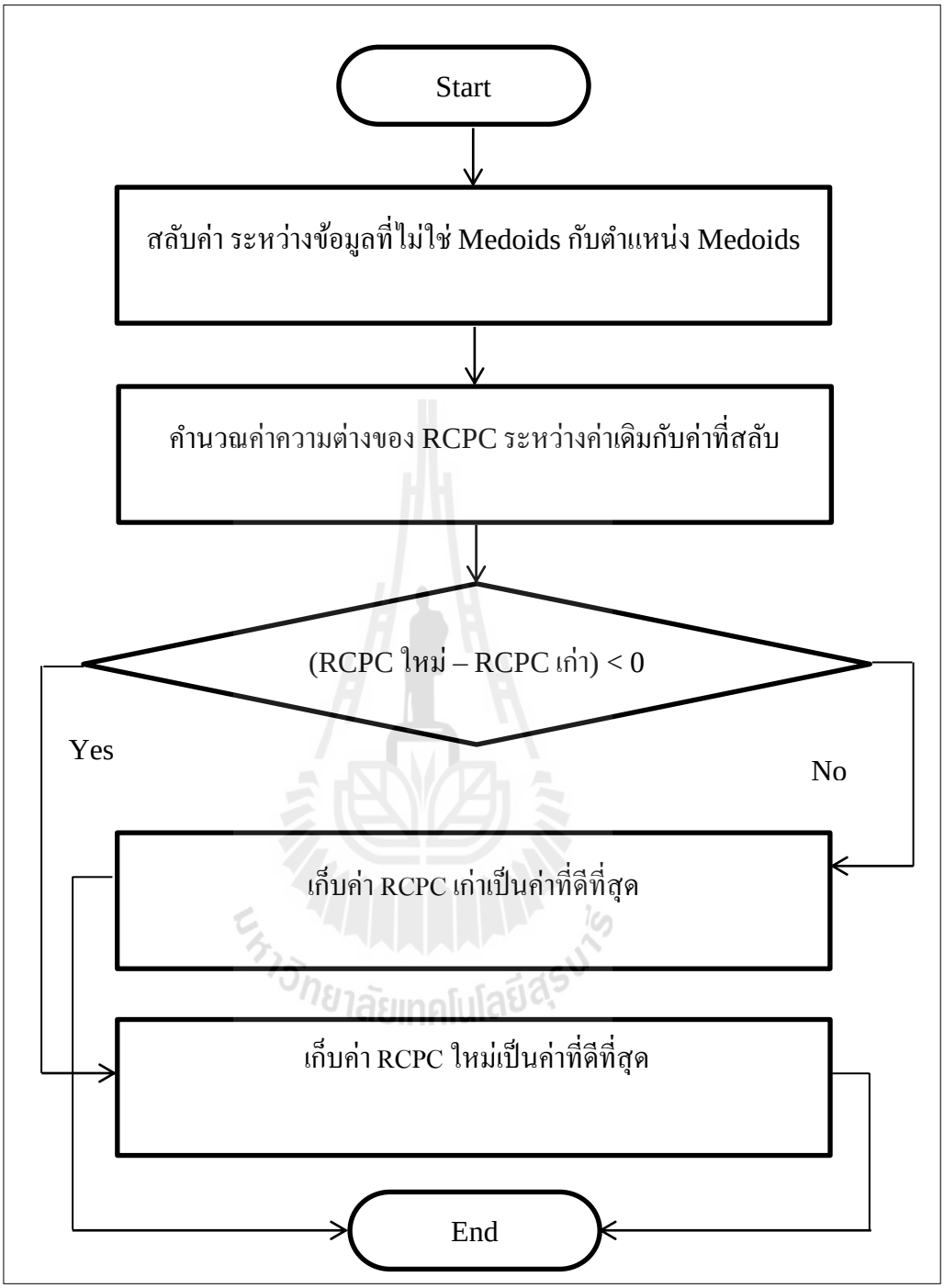
รูปที่ 3.3 แสดงตัวอย่างของข้อมูลสองมิติจำนวน 9 เรคคอร์ด



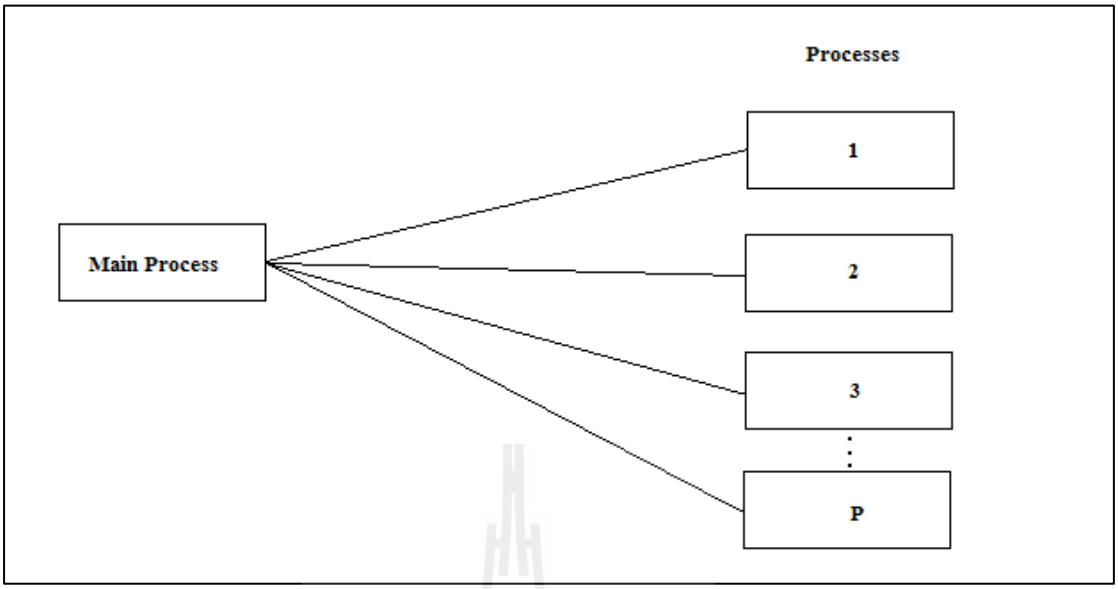
รูปที่ 3.4 แสดงกระบวนการในการอ่านไฟล์ข้อมูล



รูปที่ 3.5 แสดงกระบวนการในการแบ่งกลุ่มข้อมูลแบบ Rough K-Medoids



รูปที่ 3.6 แสดงการทำงานของแต่ละโปรเซสในการหาตำแหน่งที่จะมาแทน medoids ตัวเก่า



รูปที่ 3.7 แสดงกระบวนการแบ่งโปรเซสและแบ่งข้อมูลที่ไม่ใช่ medoids เพื่อทำการสลับค่า เป็นจำนวน P โปรเซส

3.3 เครื่องมือที่ใช้ในการวิจัย

เครื่องมือที่ใช้ในการพัฒนาโปรแกรม Parallel Rough K-Medoids clustering ประกอบด้วย

1) เครื่องคอมพิวเตอร์สำหรับการพัฒนา โดยมีรายละเอียดดังนี้
   - หน่วยประมวลผลกลาง : Intel® Core™ i3 CPU U380 @1.33GHz 1.33Gz 4 core
   - หน่วยความจำหลัก : 4.00 GB
   - หน่วยความจำสำรอง: 300 GB
   - อุปกรณ์เสริมอื่น ๆ เช่น เมาส์ แป้นพิมพ์ เป็นต้น
2) ระบบปฏิบัติการและโปรแกรมประยุกต์สำหรับการพัฒนา ประกอบไปด้วย
   - ระบบปฏิบัติการ : Ubuntu 10.10, Windows 7 Ultimate 64 bit
   - เครื่องมือที่ใช้ในการพัฒนา : Erlang OTP R14B04 (ดาวน์โหลดได้จาก http://www.erlang.org/download_release/12)

บทที่ 4
การทดสอบและการอภิปรายผล

การทดสอบประสิทธิภาพของอัลกอริทึม Parallel Rough K-Medoids clustering ที่ออกแบบด้วยเทคนิค message passing และพัฒนาด้วยภาษา Erlang เป็นการทดสอบความเร็วในการประมวลผลกับทั้งข้อมูลสังเคราะห์ และข้อมูลมาตรฐาน รวมสี่ชุดข้อมูล

4.1 วิธีการทดสอบ

การทดสอบประสิทธิภาพของระบบนั้นจะเป็นการทดสอบความเร็วในการประมวลผลของอัลกอริทึม Parallel Rough K-Medoids clustering อัลกอริทึมในแง่มุมต่าง ๆ ได้แก่

- จำนวนโปรเซส เปรียบเทียบการทำงานของ 1 โปรเซส ซึ่งแทนลักษณะของการทำงานแบบลำดับ และการทำงานแบบขนานที่มีจำนวนโปรเซสแปรผันตั้งแต่ 2 จนถึง 256 โปรเซส
- จำนวนข้อมูล เปรียบเทียบการทำงานของ 1 โปรเซส ซึ่งแทนลักษณะของการทำงานแบบลำดับ และการทำงานแบบขนานที่มีจำนวนโปรเซส 2 โปรเซส และ 4 โปรเซส กับจำนวนข้อมูลที่แปรผันตั้งแต่ 1000 จนถึง 5000 ข้อมูล
- จำนวนกลุ่มที่ต้องแบ่ง เปรียบเทียบการทำงานของ 1 โปรเซส ซึ่งแทนลักษณะของการทำงานแบบลำดับ และการทำงานแบบขนานที่มีจำนวนโปรเซส 2 โปรเซส และ 4 โปรเซส กับจำนวนข้อมูล 1000 ข้อมูลที่จำนวนกลุ่มที่ต้องการแบ่งแปรผันตั้งแต่ 2 จนถึง 32 กลุ่ม
- จำนวนมิติของข้อมูล เปรียบเทียบการทำงานของ 1 โปรเซส ซึ่งแทนลักษณะของการทำงานแบบลำดับ และการทำงานแบบขนานที่มีจำนวนโปรเซส 2 โปรเซส และ 4 โปรเซสกับจำนวนข้อมูล 5000 ข้อมูล ที่จำนวนมิติของข้อมูลแปรผันจาก 1 จนถึง 3 โดยแบ่ง 2 กลุ่ม

ทั้งนี้เมื่อต้องการเปรียบเทียบในเรื่องใดจะกำหนดให้ตัวแปรอื่น ๆ ที่เกี่ยวข้องมีค่าคงที่เท่ากัน โดยในการทดสอบจะมีข้อมูลทดสอบทั้งหมดด้วยกัน 4 แบบได้แก่

- ข้อมูลที่สร้างขึ้น เรียกว่า ข้อมูลสังเคราะห์
- ข้อมูล Colon Cancer (http://molbio.princeton.edu/colondata)
- ข้อมูล Forest Data (http://kdd.ics.uci.edu)
- ข้อมูล Control Chart Data (http://kdd.ics.uci.edu)

โดยที่เราจะใช้ข้อมูลสังเคราะห์ในการทดสอบทั้ง 4 แง่มุมดังที่ได้กล่าวไปแล้ว แต่สำหรับข้อมูลทดสอบอีก 3 แบบที่เหลือจะเป็นการวัดประสิทธิภาพของการทำงานเมื่อกำหนดจำนวนโปรเซสระหว่าง 1 โปรเซส เพื่อแทนการประมวลผลแบบลำดับ กับ 2 โปรเซส และ 4 โปรเซส เพื่อแทนการประมวลผลแบบขนาน

4.2 การทดสอบกับข้อมูลสังเคราะห์

ในการทดสอบกับข้อมูลที่สร้างขึ้นมานั้น จะทำการทดสอบการเพิ่มจำนวนโปรเซสก่อน เพื่อหาว่าเมื่อมีหลายโปรเซสแล้ว ควรกำหนดจำนวนโปรเซสเป็นเท่าไรจึงจะดีที่สุด และเพื่อยืนยันประสิทธิภาพของอัลกอริทึมว่าสามารถทำให้การประมวลผลเร็วขึ้นได้เมื่อทำงานหลายโปรเซสบนหน่วยประมวลผลกลางแบบหลายแกนหลัก หลังจากนั้นเมื่อได้จำนวนโปรเซสที่เหมาะสมแล้วจะทำการทดสอบการเพิ่มจำนวนข้อมูลโดยเทียบกันอีกครั้งระหว่าง 1 โปรเซส กับหลายโปรเซสโดยจำนวนโปรเซสกำหนดให้เท่ากับ 2 และ 4 โปรเซสเนื่องจากจำนวนแกนหลักของเครื่องที่ใช้ในการทดสอบมีจำนวน 4 แกนหลักดังนั้นถ้าใช้มากกว่า 4 ประสิทธิภาพจะแตกต่างจาก 4 ไม่มากนัก จากนั้นทำการทดสอบการเพิ่มจำนวนกลุ่มและสุดท้ายเป็นการทดสอบการเพิ่มจำนวนมิติข้อมูล

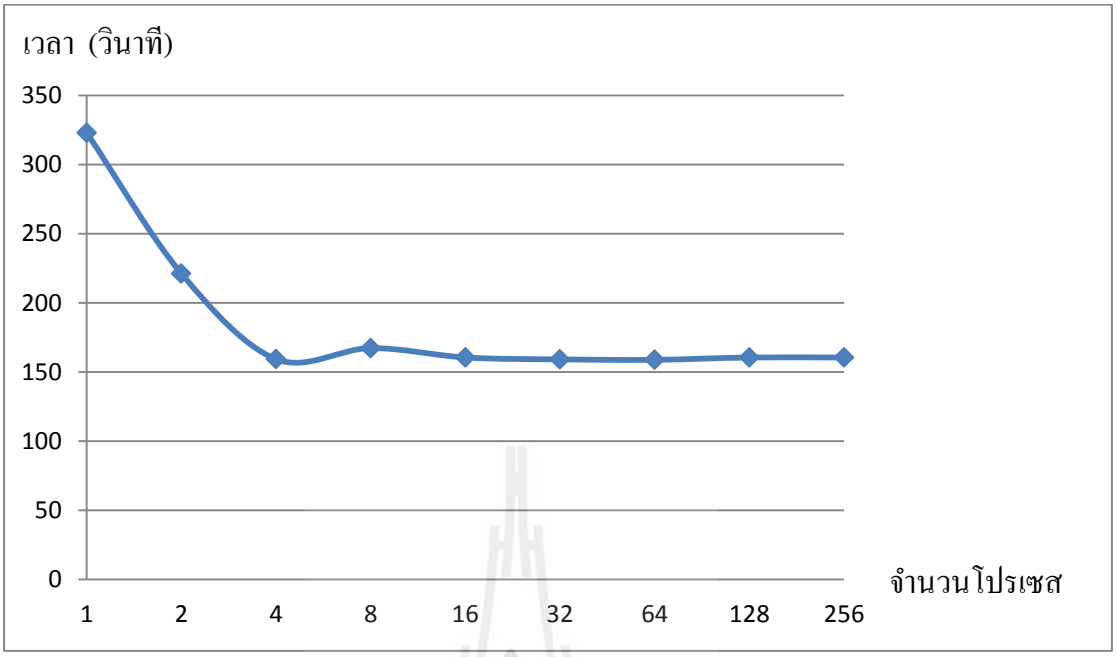
4.2.1 การทดสอบเพิ่มจำนวนโปรเซส

ในการทดสอบนี้จะเพิ่มจำนวนโปรเซส โดยทดสอบทั้งหมด 9 ครั้งแต่ละครั้งเพิ่มจำนวนโปรเซสเป็นสองเท่า จาก 1 จนถึง 256 โปรเซส โดยข้อมูลที่นำมาทดสอบนั้นเป็นข้อมูลที่สุ่มขึ้นมา มีจำนวน 5000 ข้อมูล เป็นข้อมูล 2 มิติ ทำการแบ่งข้อมูลออกเป็น 2 กลุ่ม กำหนดค่าของ w_l = 0.75 , w_b = 0.25 และค่า threshold ε = 2000.0 เนื่องจากข้อมูลที่สุ่มนี้อยู่ในช่วง 0 ถึง 10000 จึงกำหนดค่าประมาณนี้เพื่อให้ผลลัพธ์มีทั้งส่วนที่เป็น lower approximation และ upper approximation ผลการจับเวลาการทำงานเป็นไปดังตารางที่ 4.1 และ จากผลการทดลองจับเวลาที่ได้เมื่อนำมาสร้างเป็นกราฟจะได้กราฟดังรูปที่ 4.2 จากผลการทดลองพบว่าเมื่อเพิ่มจำนวนโปรเซสเวลาการทำงานจะลดลงโดยที่เมื่อจำนวนโปรเซสมีจำนวนตั้งแต่ 4 ขึ้นไปเวลาการทำงานจะลดลง

ไม่มาก เพราะว่าจำนวนแกนหลักของเครื่องที่ใช้ทดสอบมีจำนวน 4 แกนหลัก ดังนั้นจึงเลือกทดสอบการทดลองถัดไปกับจำนวนโปรเซส 1 , 2 และ 4 โปรเซสตามลำดับ เพื่อศึกษาผลที่เกิดขึ้นในกรณีของการเปลี่ยนแปลงค่าอื่น ๆ ต่อไป

ตารางที่ 4.1 ผลการทดลองจับเวลาการทำงานเมื่อจำนวนโปรเซสต่างกัน

จำนวนโปรเซส	เวลาการทำงาน (วินาที)
1	323.04
2	221.28
4	159.37
8	167.32
16	160.55
32	159.17
64	158.85
128	160.53
256	160.50



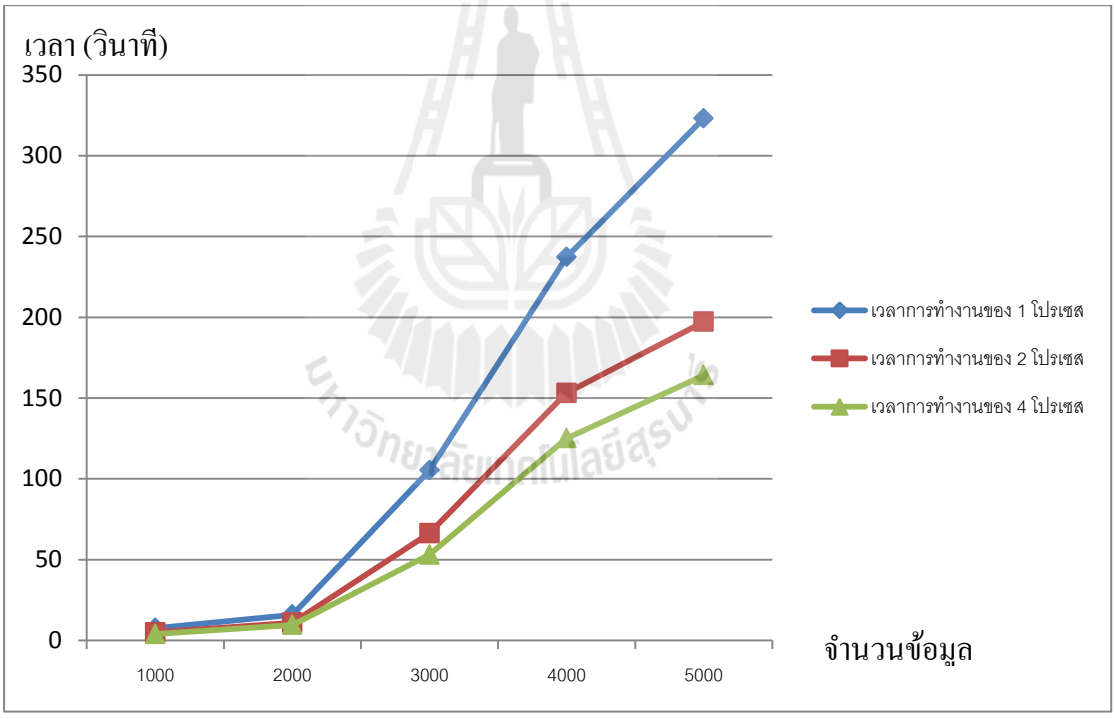
รูปที่ 4.1 กราฟแสดงเวลาการทำงานเมื่อจำนวนโปรเซสเพิ่มขึ้น

4.2.2 การทดสอบเพิ่มจำนวนข้อมูล

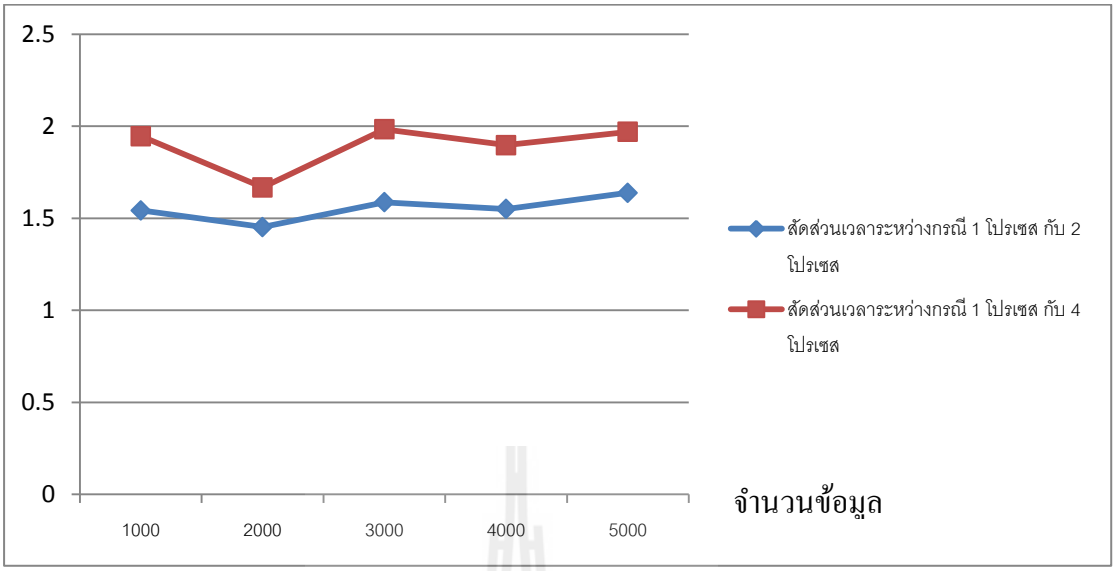
ในการทดสอบนี้จะทดลองกับข้อมูลสังเคราะห์ที่มีจำนวนข้อมูลต่างกัน โดยเริ่มจาก 1000 ข้อมูล แล้วเพิ่มขึ้นทีละ 1000 จนถึง 5000 ข้อมูล โดยที่จำนวนมิติของข้อมูลเป็นสองมิติ และทำการแบ่งข้อมูลออกเป็น 2 กลุ่ม กำหนดค่าของ $w_l = 0.75$, $w_b = 0.25$ และค่า threshold ε = 2000.0 เนื่องจากข้อมูลที่สุ่มนี้อยู่ในช่วง 0 ถึง 10000 จึงกำหนดค่าประมาณนี้เพื่อให้ผลลัพธ์มีทั้งส่วนที่เป็น lower approximation และ upper approximation ทำการทดสอบกับจำนวนโปรเซส 1, 2 และ 4 โปรเซสเพื่อเปรียบเทียบเวลาในการทำงาน ผลการจับเวลาการทำงานเป็นไปดังตารางที่ 4.2 แผนภาพเปรียบเทียบเวลาการทำงานแสดงดังรูปที่ 4.2 และแผนภาพเปรียบเทียบสัดส่วนเวลาที่ลดลงระหว่าง 1 โปรเซสกับ 2 โปรเซส และ ระหว่าง 1 โปรเซส กับ 4 โปรเซสดังรูปที่ 4.3 จากผลการทดลองสังเกตได้ว่า ถึงแม้ว่าจะใช้จำนวนข้อมูลที่เพิ่มขึ้นตามลำดับ แต่ว่าการทำงานของโปรแกรมที่ใช้จำนวนโปรเซส 2 โปรเซส และ 4 โปรเซส ที่ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักก็ยังมีแนวโน้นทำงานได้เร็วกว่าโปรแกรมที่ทำงานแบบ 1 โปรเซส โดยความเร็วเพิ่มขึ้นไปตามจำนวนแกนหลักที่เพิ่มขึ้น และ การทำงานของ 2 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 1.55 เท่า ส่วนการทำงานของ 4 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 1.89 เท่า

ตารางที่ 4.2 แสดงผลการจับเวลาเมื่อเพิ่มจำนวนข้อมูล

จำนวนข้อมูล	เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
1000	7.59	4.92	3.90
2000	15.91	10.96	9.54
3000	105.31	66.38	53.11
4000	237.26	153.08	125.09
5000	323.09	197.22	164.10



รูปที่ 4.2 แผนภาพเปรียบเทียบเวลาการทำงานของกรณี 1 โปรเซส 2 โปรเซส และ 4 โปรเซส เมื่อเพิ่มจำนวนข้อมูล



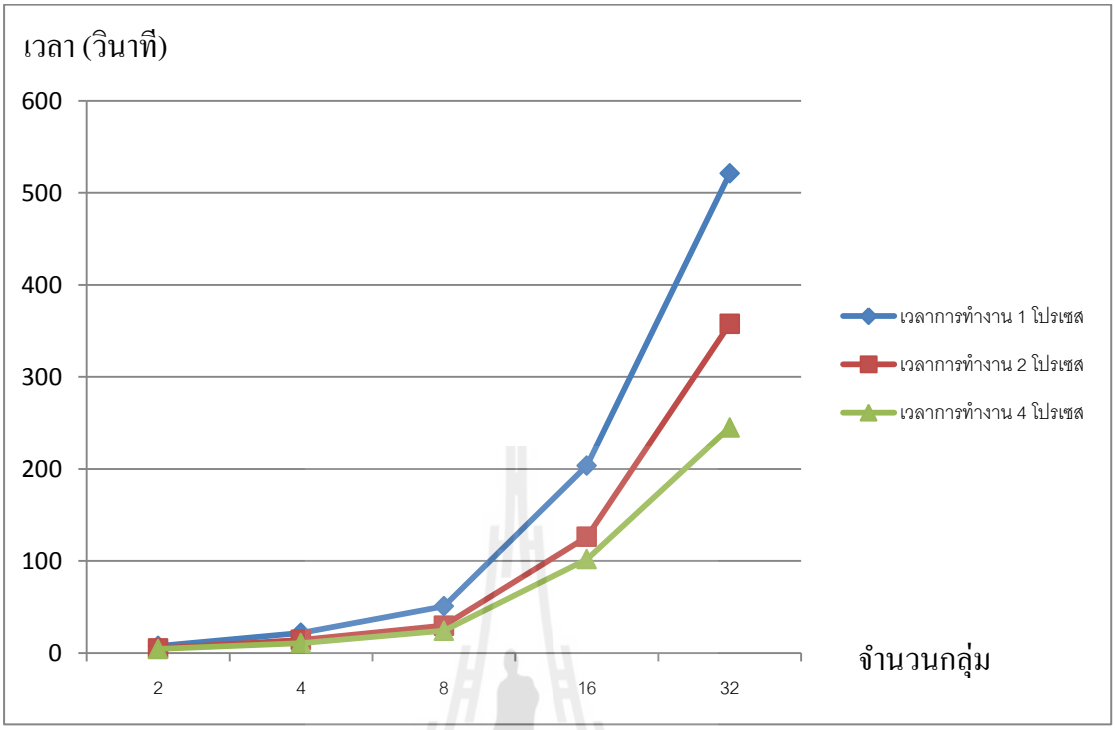
รูปที่ 4.3 แผนภาพแสดงสัดส่วนเวลาระหว่างกรณี 1 โปรเซส กับ 2 โปรเซส และ ระหว่างกรณี 1 โปรเซส กับ 4 โปรเซส เมื่อเพิ่มจำนวนข้อมูล

4.2.3 การทดสอบการเพิ่มจำนวนกลุ่ม

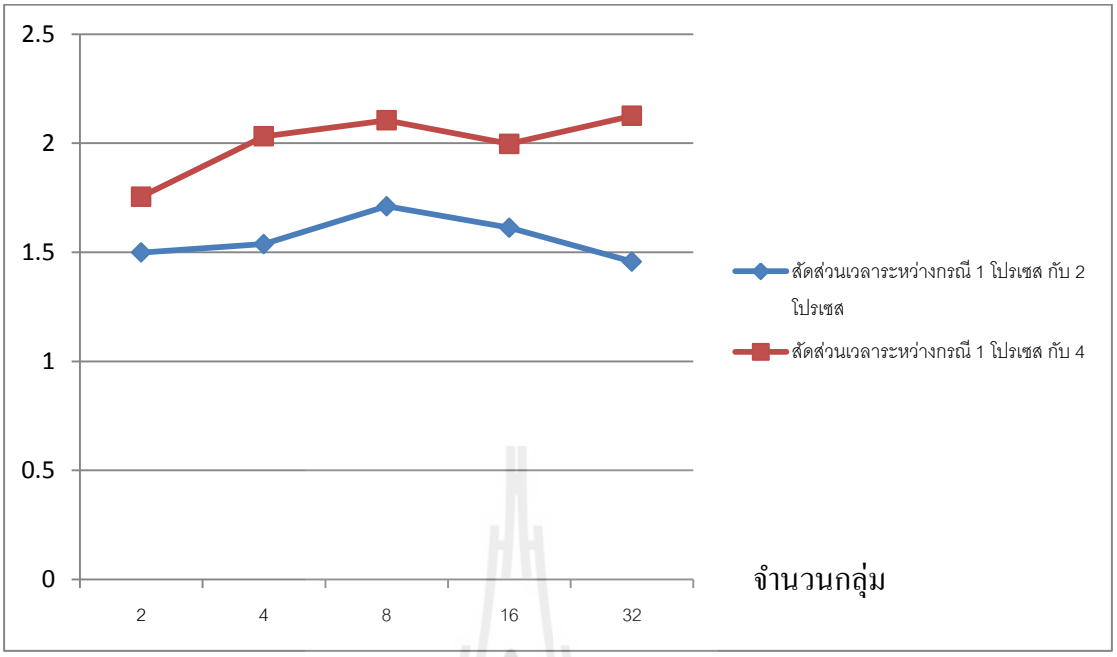
ในการทดสอบนี้จะทดลองเพิ่มจำนวนกลุ่มที่ต้องการแบ่ง โดยเริ่มจากจำนวน 2 กลุ่ม แล้วเพิ่มครั้งละสองเท่าจนถึง 32 กลุ่ม ทดลองกับข้อมูลจำนวน 1000 ข้อมูล โดยที่จำนวนมิติของข้อมูลเท่ากับ 2 กำหนดค่าของ $w_l = 0.75$, $w_b = 0.25$ และค่า threshold $\varepsilon = 2000.0$ เนื่องจากข้อมูลที่สุ่มนี้อยู่ในช่วง 0 ถึง 10000 จึงกำหนดค่าประมาณนี้เพื่อให้ผลลัพธ์มีทั้งส่วนที่เป็น lower approximation และ upper approximation และทำการทดสอบกับจำนวนโปรเซส 1, 2 และ 4 โปรเซสเพื่อเปรียบเทียบเวลาในการทำงาน ผลการจับเวลาการทำงานเป็นไปดังตารางที่ 4.3 และแผนภาพเปรียบเทียบเวลาการทำงานแสดงได้ดังรูปที่ 4.4 และแผนภาพเปรียบเทียบสัดส่วนเวลาที่ลดลงระหว่าง 1 โปรเซสกับ 2 โปรเซส และ ระหว่าง 1 โปรเซส กับ 4 โปรเซสดังรูปที่ 4.5จากผลการทดลองสังเกตได้ว่า ถึงแม้ว่าจะเพิ่มจำนวนกลุ่มข้อมูลที่ต้องการแบ่ง แต่ว่าการทำงานของโปรแกรมที่ใช้จำนวนโปรเซส 2 โปรเซส และ 4 โปรเซส ที่ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักก็ยังมีแนวโน้นทำงานได้เร็วกว่าโปรแกรมที่ทำงานแบบ 1 โปรเซส โดยความเร็วเพิ่มขึ้นไปตามจำนวนแกนหลักที่เพิ่มขึ้น และ การทำงานของ 2 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 1.56 เท่า ส่วนการทำงานของ 4 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 2.00 เท่า

ตารางที่ 4.3 แสดงผลการจับเวลาเมื่อเพิ่มจำนวนกลุ่ม

จำนวนกลุ่มข้อมูล	เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
2	7.60	5.07	4.33
4	21.96	14.28	10.81
8	50.77	29.68	24.12
16	203.59	126.23	101.94
32	521.10	357.64	245.13



รูปที่ 4.4 แผนภาพเปรียบเทียบเวลาการทำงานของกรณี 1 โปรเซส 2 โปรเซส และ 4 โปรเซส เมื่อเพิ่มจำนวนกลุ่มที่ต้องการแบ่ง



รูปที่ 4.5 แผนภาพแสดงสัดส่วนเวลาระหว่างกรณี 1 โปรเซส กับ 2 โปรเซส และ ระหว่างกรณี 1 โปรเซส กับ 4 โปรเซส เมื่อเพิ่มจำนวนกลุ่มที่ต้องการแบ่ง

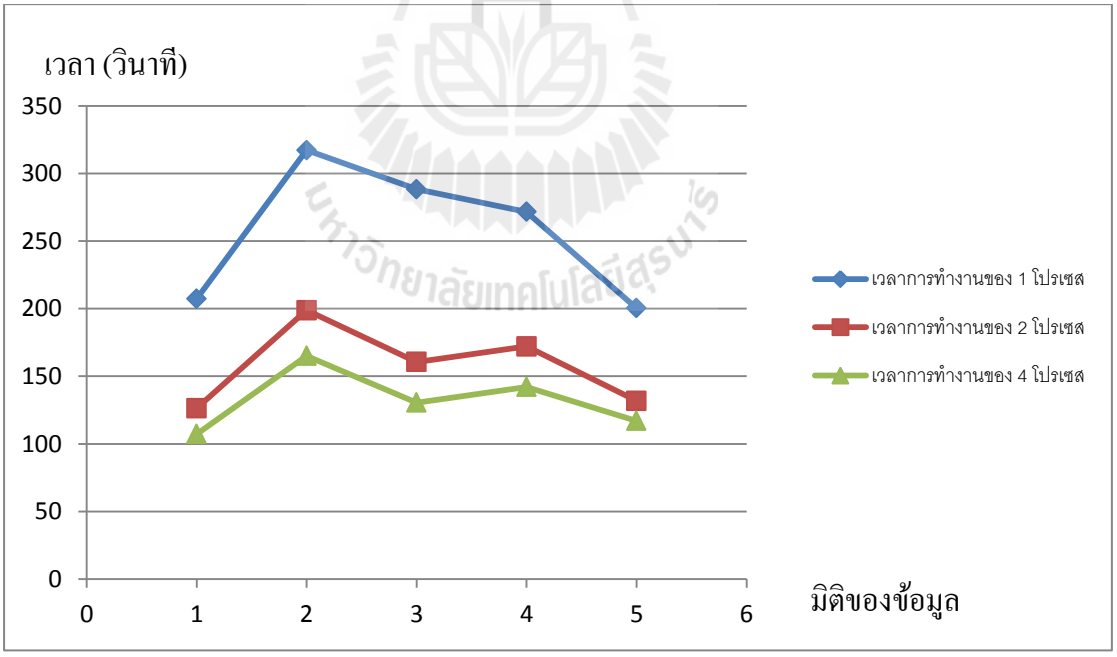
4.2.4 การทดสอบเพิ่มจำนวนมิติข้อมูล

ในการทดสอบนี้จะทดลองเพิ่มจำนวนมิติของข้อมูล โดยทดสอบกับข้อมูล 1 มิติ 2 มิติ 3 มิติ 4 มิติ และ 5 มิติ ทดลองกับข้อมูลจำนวน 5000 ข้อมูล และแบ่งข้อมูลออกเป็น 2 กลุ่ม กำหนดค่าของ $w_l = 0.75$, $w_b = 0.25$ และค่า threshold $\varepsilon = 2000.0$ เนื่องจากข้อมูลที่สุ่มนี้อยู่ในช่วง 0 ถึง 10000 จึงกำหนดค่าประมาณนี้เพื่อให้ผลลัพธ์มีทั้งส่วนที่เป็น lower approximation และ upper approximation และทำการทดสอบกับจำนวนโปรเซส 1, 2 และ 4 โปรเซสเพื่อเปรียบเทียบเวลาในการทำงาน ผลการจับเวลาการทำงานเป็นไปดังตารางที่ 4.4 และแผนภาพเปรียบเทียบเวลาการทำงานแสดงได้ดังรูปที่ 4.6 และแผนภาพเปรียบเทียบสัดส่วนเวลาที่ลดลงระหว่าง 1 โปรเซสกับ 2 โปรเซส และ ระหว่าง 1 โปรเซส กับ 4 โปรเซสดังรูปที่ 4.7 จากผลการทดลองสังเกตได้ว่า ถึงแม้ว่าจะเพิ่มจำนวนมิติที่ต้องการแบ่ง แต่ว่าการทำงานของโปรแกรมที่ใช้จำนวนโปรเซส 2 โปรเซส และ 4 โปรเซส ที่ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักก็ยังมีแนวโน้นทำงานได้เร็วกว่าโปรแกรมที่ทำงานแบบ 1 โปรเซส โดยความเร็วเพิ่มขึ้นไปตามจำนวนแกนหลักที่เพิ่มขึ้น และ การทำงานของ 2 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 1.63เท่า ส่วนการทำงานของ 4 โปรเซสเร็วขึ้นเมื่อเทียบกับ 1 โปรเซสเฉลี่ยประมาณ 1.94 เท่า แต่ว่าเมื่อมองในมุมการเพิ่มขึ้นของมิติข้อมูลต่อเวลาในการประมวลผลพบว่าเมื่อเพิ่มจำนวนมิติข้อมูล บางครั้งทำงานได้เร็วกว่าจำนวน

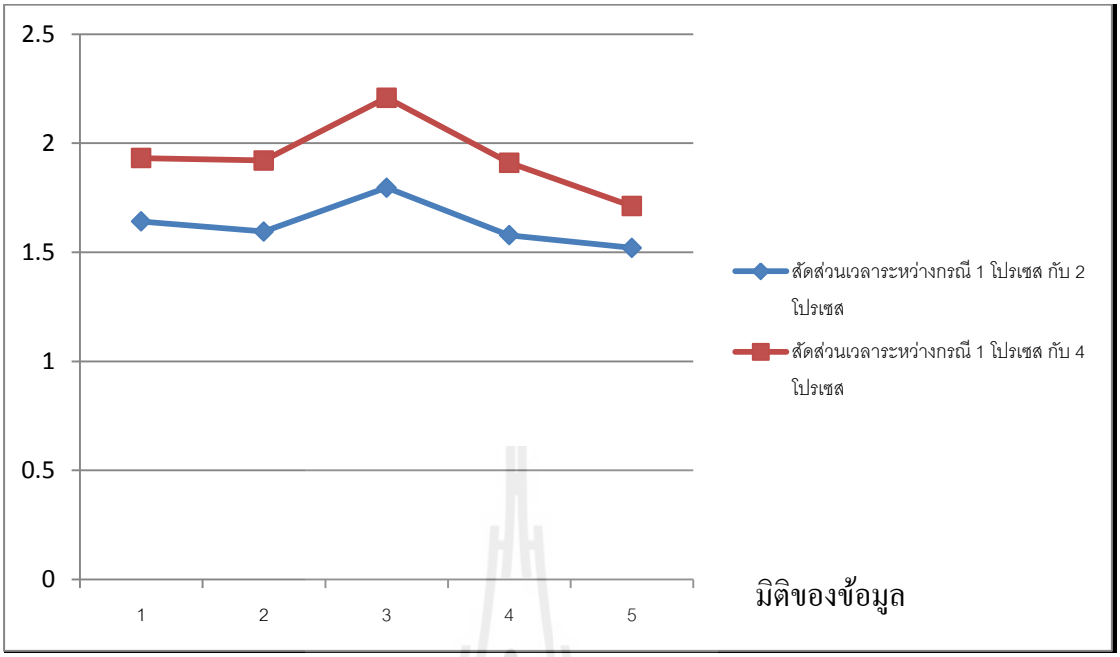
มิติที่น้อยกว่า เพราะว่าการเพิ่มมิติทำให้การกระจายตัวของข้อมูลแตกต่างไปจากเดิมการแบ่งกลุ่มจึงเป็นไปได้ทั้งเร็วขึ้นและช้าลงในงานวิจัยนี้มองในมุมมองความต่างของจำนวนแกนหลักของหน่วยประมวลผลเท่านั้น

ตารางที่ 4.4 แสดงผลการจับเวลาเมื่อเพิ่มจำนวนมิติข้อมูล

จำนวนมิติข้อมูล	เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
1	207.37	126.38	107.35
2	317.11	198.79	165.11
3	288.26	160.53	130.52
4	271.54	172.03	142.14
5	200.39	131.82	117.08



รูปที่ 4.6 แผนภาพเปรียบเทียบเวลาการทำงานของกรณี 1 โปรเซส 2 โปรเซส และ 4 โปรเซส เมื่อเพิ่มจำนวนมิติข้อมูล



รูปที่ 4.7 แผนภาพแสดงสัดส่วนเวลาระหว่างกรณี 1 โปรเซส กับ 2 โปรเซส และ ระหว่างกรณี 1 โปรเซส กับ 4 โปรเซส เมื่อเพิ่มจำนวนมิติข้อมูล

4.3 การทดสอบกับข้อมูล Conlon Cancer

ข้อมูลของ colon cancer ที่นำมาทดสอบเป็นข้อมูลทั้งหมด 62 เรคคอร์ด เป็นข้อมูลที่มีทั้งหมด 21 มิติ ในการทดสอบได้กำหนดค่าคงพารามิเตอร์ต่าง ๆ ดังต่อไปนี้

- $w_l = 0.75$
- $w_b = 0.25$
- threshold ε = 0.2 เนื่องจากข้อมูลมีการเกาะกลุ่มกันทำให้ค่าระยะห่างที่ต่างกันมีค่าน้อย ดังนั้นจึงเลือกค่า ε น้อยถ้าเลือกค่า ε สูงเกินไปจะทำให้ข้อมูลอยู่ภายในส่วน boundary region เกือบทั้งหมด
- ทำการแบ่งข้อมูลออกเป็น 2 กลุ่ม

เมื่อนำมาทดสอบกับโปรแกรมที่พัฒนาโดยเปรียบเทียบการทำงานระหว่าง 1 โปรเซสกับ 2 โปรเซส และ 4 โปรเซส ได้เวลาการทำงานดังตารางที่ 4.5

ตารางที่ 4.5 เวลาการทำงานของการแบ่งกลุ่มข้อมูล colon cancer ระหว่าง 1 โปรเซส กับ 2 โปรเซส และ 4 โปรเซส

เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
0.06	0.05	0.04

4.4 การทดสอบกับข้อมูล Forest Data

ข้อมูลของ forest data ที่นำมาทดสอบเป็นข้อมูลทั้งหมด 241 เรคคอร์ด เป็นข้อมูลที่มีทั้งหมด 10 มิติ ในการทดสอบได้กำหนดค่าคงพารามิเตอร์ต่างๆดังต่อไปนี้

- $w_l = 0.75$
- $w_b = 0.25$
- threshold $\varepsilon = 200$
- ทำการแบ่งข้อมูลออกเป็น 3 กลุ่ม

เมื่อนำมาทดสอบกับโปรแกรมที่พัฒนาโดยเปรียบเทียบการทำงานระหว่าง 1 โปรเซสกับ 2 โปรเซส และ 4 โปรเซส ได้เวลาการทำงานดังตารางที่ 4.6

ตารางที่ 4.6 เวลาการทำงานของการแบ่งกลุ่มข้อมูล forest data ระหว่าง 1 โปรเซส กับ 2 โปรเซส และ 4 โปรเซส

เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
1.55	1.13	0.76

4.5 การทดสอบกับข้อมูล Control Chart Data

ข้อมูลของ control chart data ที่นำมาทดสอบเป็นข้อมูลทั้งหมด 21 เรคคอร์ด เป็นข้อมูลที่มีทั้งหมด 60 มิติ ในการทดสอบได้กำหนดค่าคงพารามิเตอร์ต่างๆดังต่อไปนี้

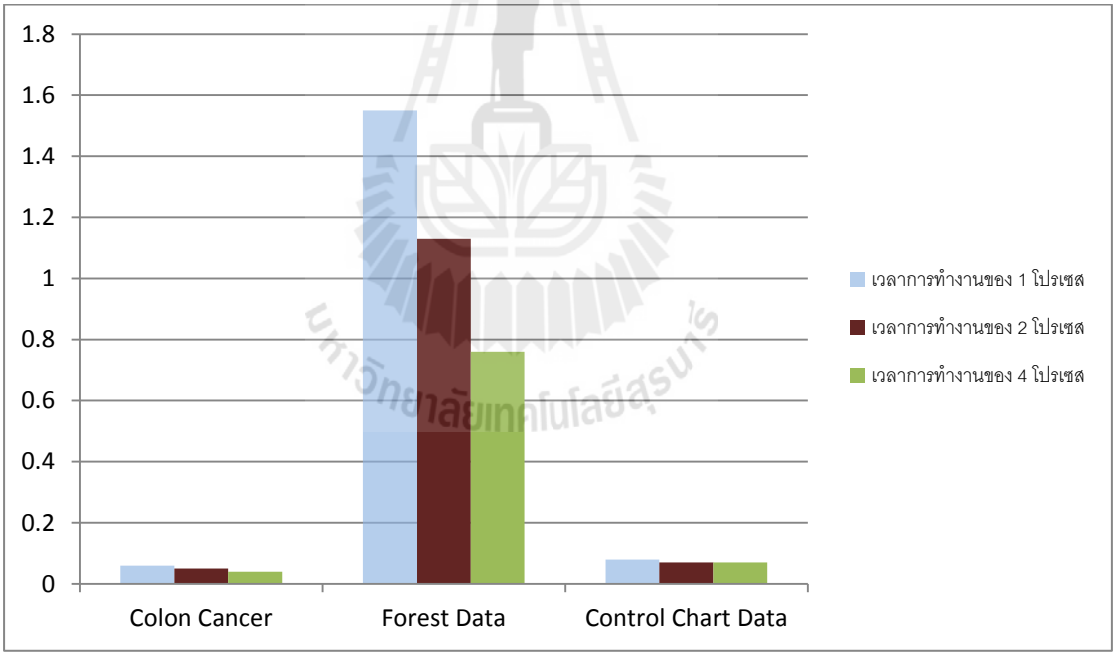
- $w_l = 0.75$
- $w_b = 0.25$
- threshold $\varepsilon = 20$
- ทำการแบ่งข้อมูลออกเป็น 6 กลุ่ม

เมื่อนำมาทดสอบกับโปรแกรมที่พัฒนาโดยเปรียบเทียบการทำงานระหว่าง 1 โปรเซสกับ 2 โปรเซส และ 4 โปรเซส ได้เวลาการทำงานดังตารางที่ 4.7

ตารางที่ 4.7 เวลาการทำงานของการแบ่งกลุ่มข้อมูล control chart data ระหว่าง 1 โปรเซส กับ 2 โปรเซส และ 4 โปรเซส

เวลาการทำงาน 1 โปรเซส (วินาที)	เวลาการทำงาน 2 โปรเซส (วินาที)	เวลาการทำงาน 4 โปรเซส (วินาที)
0.08	0.07	0.07

จากการทดสอบกับข้อมูลจริงทั้ง 3 แบบคือ colon cancer, forest data, control chart data สามารถนำผลการทดสอบ แบบมาสร้างเป็นกราฟแท่งเปรียบเทียบเวลาการทำงานได้ดังรูปที่ 4.8



รูปที่ 4.8 กราฟแท่งเปรียบเทียบเวลาการทำงานของ 1 โปรเซสกับ 2 โปรเซส และ 4 โปรเซส สำหรับข้อมูลทดสอบ colon cancer, forest data, control chart data

จากกราฟพบว่าการทำงานของโปรแกรมที่ประมวลผล 2 โปรเซสและ 4 โปรเซสที่ทำงานบนหน่วย ประมวลผลแบบหลายแกนทำงานได้ดีกว่าการประมวลผล 1 โปรเซสทุก ๆ การทดสอบ

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

ในปัจจุบันการทำเหมืองข้อมูลได้รับความนิยมเป็นอย่างมาก มีการศึกษาวิจัย และ นำไปใช้งานจริงอย่างต่อเนื่องไม่ว่าจะเป็นงานทางด้านธุรกิจ ด้านวิทยาศาสตร์ ด้านสังคม และ การศึกษา เนื่องจากวิวัฒนาการของเครื่องคอมพิวเตอร์และโครงข่ายคอมพิวเตอร์ในปัจจุบันทำให้เกิดการสะสมของข้อมูลสารสนเทศเพิ่มขึ้นอย่างรวดเร็วทุก ๆ ปี ดังนั้นการที่จะให้ได้มาซึ่งประโยชน์จากข้อมูลเหล่านั้นต้องมีเครื่องมือช่วยในการขุดค้น และวิเคราะห์ข้อมูล ซึ่งเทคนิคทางด้านการทำเหมืองข้อมูลเป็นเทคนิคที่จะช่วยให้เราพัฒนาโปรแกรมในการขุดค้นและวิเคราะห์ข้อมูลโดยอัตโนมัติ

งานประเภทหนึ่งของการทำเหมืองข้อมูลที่ได้รับความนิยมได้แก่ การแบ่งกลุ่มข้อมูล เพื่อศึกษารูปแบบ จากกลุ่มข้อมูลที่มีความคล้ายกัน ข้อดีของวิธีการนี้คือสามารถประมวลผลได้โดยไม่ต้องมีข้อมูลอื่นมาช่วยในการแนะนำการวิเคราะห์ (Unsupervise)

อัลกอริทึมเพื่องานแบ่งกลุ่มข้อมูลที่ได้รับความนิยมคือ K-Means clustering เนื่องจากเข้าใจได้ง่ายและพัฒนาโปรแกรมได้ง่าย แต่ว่าข้อเสียของ K-Means clustering คือต้องประมวลผลข้อมูลที่เป็นเชิงตัวเลข (numerical) เนื่องจากต้องคำนวนหาค่าเฉลี่ยเพื่อมาเป็นตัวแทนตำแหน่งกลางของกลุ่มข้อมูล และ เป็นการจัดกลุ่มที่แบ่งกลุ่มข้อมูลออกอย่างชัดเจน ข้อมูลต้องอยู่กลุ่มใดกลุ่มหนึ่งเท่านั้นซึ่งอาจขัดแย้งกับข้อมูลจริงบางประเภทที่สามารถมีลักษณะร่วมกัน ไม่สามารถแยกได้ชัดเจนว่าอยู่กลุ่มได้

จากข้อจำกัดดังกล่าวจึงได้มีการพัฒนาอัลกอริทึมที่ชื่อว่า Rough K-Medoids clustering ขึ้นมาเพื่อจัดการกับข้อมูลที่เป็นเชิงกลุ่ม (categorical) โดยใช้ข้อมูลจริงแทนตำแหน่งค่ากลางของกลุ่ม และได้นำเอาทฤษฎีราฟเซต มาช่วยเพื่อให้สามารถแบ่งกลุ่มข้อมูลที่ไม่สามารถแยกได้ชัดเจนออกเป็นอีกระดับได้คือในระดับ upper approximation กับระดับ lower approximation

ซึ่งถ้าข้อมูลใดแยกได้ไม่ชัดเจนจะอยู่ในระดับของ upper approximation ถ้าแยกได้ชัดเจนจะอยู่ในระดับ lower approximation

เนื่องจากอัลกอริทึม Rough K-Medoids clustering นั้นยังมีข้อเสียคือใช้เวลาในการประมวลผลนานกว่า K-Means clustering เลยยังไม่เหมาะกับการวิเคราะห์ข้อมูลขนาดใหญ่ งานวิจัยนี้จึงได้นำเสนอวิธีการที่จะพัฒนาอัลกอริทึมแบบขนานสำหรับอัลกอริทึม Rough K-Medoids clustering เพื่อประมวลผลบนหน่วยประมวลผลกลางแบบหลายแกนที่มีใช้แพร่หลายในปัจจุบัน โดยเลือกใช้ภาษา Erlang ในการพัฒนาเนื่องจากมีข้อดีคือเข้าใจได้ง่าย สามารถเขียนโปรแกรมให้ทำงานหลายโปรเซสได้ง่าย เป็นภาษาเชิงฟังก์ชันที่ช่วยให้การเขียนโปรแกรมกระชับ ไม่ซับซ้อน และสามารถทำงานบนหน่วยประมวลผลแบบหลายแกนหลักได้อย่างมีประสิทธิภาพ งานวิจัยนี้ทำการทดสอบประสิทธิการของโปรแกรมที่พัฒนากับข้อมูลทดสอบในหลายประเด็น ทั้งเปรียบเทียบเวลาการทำงานเมื่อเพิ่มจำนวนโปรเซส , เปรียบเทียบเวลาการทำงานเมื่อเพิ่มจำนวนข้อมูล, เปรียบเทียบเวลาการทำงานเมื่อเพิ่มกลุ่มที่ต้องการแบ่ง, เปรียบเทียบการทำงานเมื่อเพิ่มมิติข้อมูล และ เปรียบเทียบการทำงานเมื่อนำไปวิเคราะห์ข้อมูลตัวอย่างที่เป็นข้อมูลจริงไม่ใช่ข้อมูลที่ได้จากการสังเคราะห์ได้แก่ colon cancer, forest data, control chart data

5.1 สรุปผลการวิจัย

จากการทดสอบการทำงานกับข้อมูลสังเคราะห์และข้อมูลจริงทั้ง 3 ชุดคือ colon cancer, forest data, control chart data พบว่าประสิทธิภาพการทำงานของโปรแกรมแบบขนานที่ทำงานบนหน่วยประมวลผลกลางแบบหลายแกน และ อัลกอริทึมที่ได้ออกแบบไว้อย่างเหมาะสมสำหรับการแบ่งกลุ่มด้วยอัลกอริทึม Parallel Rough K-Medoids clustering ใช้เวลาการทำงานน้อยกว่าอัลกอริทึมแบบเดิมที่อยู่ในลักษณะการทำงานแบบลำดับ (serial) ในทุกการทดสอบ ไม่ว่าจะเป็นการเพิ่มจำนวนข้อมูล, เพิ่มจำนวนกลุ่มที่ต้องการแบ่ง และเพิ่มจำนวนมิติของข้อมูล

5.2 ปัญหาและข้อเสนอแนะ

เนื่องจาก Erlang เป็นภาษาเชิงฟังก์ชันที่ทำงานแบบขนานโดยอาศัยการส่งข้อความระหว่างโปรเซส ดังนั้นจึงต้องระวังการส่งผ่านข้อมูลระหว่างโปรเซสที่มากจนเกินไป และ ต้องเลือกงานส่วนที่จะประมวลผลแบบขนานเฉพาะส่วนที่มีการประมวลผลมากจึงจะคุ้มกับเวลาที่ต้องเสียไปในการส่งผ่านข้อมูล นอกจากนั้นถ้าเกิดมีส่วนใดของภาษา Erlang ที่ทำงานช้าจนเกินไปเราสามารถแยกส่วนนั้นไปพัฒนาด้วยภาษาระดับต่ำกว่าเช่นภาษา C แล้วจึงนำมาเรียกใช้ผ่านภาษา Erlang จะทำให้ประสิทธิภาพการทำงานโดยรวมดีขึ้นมาก

ปัจจุบันยังต้องการการพัฒนาประสิทธิภาพของอัลกอริทึมสำหรับการทำเหมืองข้อมูลอีกหลายอัลกอริทึม การนำเอาเทคนิคการประมวลผลแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนมาใช้จึงช่วยให้ประมวลผลได้เร็วขึ้นมาก ดังนั้นความรู้ที่ได้จากงานวิจัยนี้สามารถใช้เป็นแนวทางในการนำไปพัฒนาอัลกอริทึมอื่น ๆ ของการทำเหมืองข้อมูลได้อีกต่อไปในอนาคต

รายการอ้างอิง

กิตติศักดิ์ เกิดประสพ. (2552). **การพัฒนาการทำเหมืองข้อมูลแบบจัดกลุ่ม**. นครราชสีมา: มหาวิทยาลัยเทคโนโลยีสุรนารี.

B. Barney. (2011). **Introduction to Parallel Computing.** [ออนไลน์] Lawrence Livermore National Laboratory. ได้จาก https://computing.llnl.gov/tutorials/parallel_comp/

C.G. Bell, W. Broadley, W. Wulf, A. Newel, C. Pierson, R. Reddy, and S. Rege. (1971). **Cmmp: The CMU Multi-mini-processor Computer.** Tech Report, Computer Science Department, Carnegie-Mellon University.

E. Deza and M. M. Deza. (2009). **Encyclopedia of Distances**. Springer. p 94.

R. Farivar, D.Rebolledo, E. Chan, and R. Campbell. (2008). **A parallel implementation of k-means clustering on GPUs.** Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp. 340-345.

R.A Johnson and D.W Wichern, (1992). **Applied Multivariate Statistical Analysis**. Prentice-Hall, Englewood Cliffs, NJ.

J. Kaufman and P.J. Rousseeuw. (1990). **Finding Groups in Data: An Introduction to Cluster Analysis**. John Wiley, New York.

K. Kerdprasop and N. Kerdprasop. (2010). **Parallelization of K-means Clustering on Multi-core Processors**.. Proceedings of 10th WSEAS International Conference on Applied Computer Science, Japan, pp. 472-477.

N. Kerdprasop and K. Kerdprasop. (2007). **Semantic Knowledge Integration to Support Inductive Query Optimization**. LNCS, vol. 4654, pp. 157-169.

E. F. Krause. (1987). **Taxicab Geometry.** Dover.

G. Laszewski and D. Roberts. (2009). **K-Medoids: CUDA implementation**. Service Oriented Cyberinfrastructure Lab, Rochester Institute of Technology.

P. Lingras and C. West. (2002). **Interval set clustering of web users with rough k-means.** Technical Report 2002-002, Department of Mathematics and Computer Science, St. Mary's University, Halifax, Canada.

S. Mitra. (2004). **An evolutionary rough partitive clustering.** Pattern Recognition Letters, pp. 1439-1449.

P. Murphy and D. Aha. (1994). **Uci repository o machine learning databases.** [ออนไลน์]. ได้จาก http://archive.ics.uci.edu/ml/datasets/zoo

G. J. Myatt and W. P. Johnson. (2009). **Making Sense of Data II: A Practical Guide to Data Visualization, Advanced Data Mining Methods, and Applications**. Wiley.

O.J. Oyelade, O.O. Oladipupo, and I.C. Obagbuwa. (2010). **Application of k-Means clustering algorithm for prediction of students' academic performance.** International Journal of Computer Science and Information Security, 7(1):292-295.

Z. Pawlak. (1982). **Rough sets**. International Journal of Computer and Information Science, 11:341—356.

Z.Pawlak. (2002). **Rough set theory and its applications.** Journal of Telecommunications and Information Technology.

G. Peters and M. Lampart. (2006). **A partitive rough clustering algorithm**. Proceedings of RSCTC 2006-Fifth International Conference on Rough Sets and Current Trends in Computing, Lecture Notes in Artificial Intelligence, pp. 657–666.

H. Wang, J. Zhao, H. Li, and J. Wang. (2008). **Parallel clustering algorithms for image processing on multi-core CPUs**. Proceedings of International Conference on Computer Science and Software Engineering (CSSE), pp 450-53.

T. Zhou, Y.M. Zhang, H.E.J. Yuan, H.L. Lu.(2007). **Rough k-means cluster with adaptive parameters**. Proceedings of 6th. International Conference on Machine Learning and Cybernetics(ICMLC'07), pp. 3063–3068.

ภาคผนวก ก

บทความทางวิชาการที่ได้รับการตีพิมพ์เผยแพร่

รายชื่อบทความที่ได้รับการตีพิมพ์เผยแพร่

วีรศักดิ์ ช่องงูเหลือม (2012). **การจัดกลุ่มข้อมูลแบบ ราฟ เค-มีดอยส์ แบบขนานบนหน่วยประมวลกลางแบบหลายแกนหลัก**. การประชุมวิชาการ “ศรีนครินทรวิโรฒวิชาการ” ครั้งที่ 6. 29-30 พฤษภาคม, 2012.,หน้า 84

การประชุมวิชาการ "ศรีนครินทรวิโรฒวิชาการ" ครั้งที่ 6
29 – 30 พฤษภาคม 2555 มหาวิทยาลัยศรีนครินทรวิโรฒ

SWU6-1022: การจัดกลุ่มข้อมูลแบบ ราฟ เค-มีดอยส์ แบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก

PARALLELIZED ROUGH K-MEDOIDS CLUSTERING ON MULTICORE PROCESSORS

วีรศักดิ์ ช่องงูเหลือม*, กิตติศักดิ์ เกิดประสพ
Weerasak Chongnguluam*, Kittisak Kerdprasop

สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
School of Computer Engineering, Institute of Engineering Suranaree University of Technology, Thailand.
***Corresponding author, E-mail:** singpor@gmail.com

บทคัดย่อ

หน่วยประมวลผลกลางแบบหลายแกนหลัก (Multi-core processors) นั้นปัจจุบันมีใช้งานกันแพร่หลาย ทั้งบนเครื่องคอมพิวเตอร์ส่วนบุคคล เครื่องคอมพิวเตอร์แบบพกพา รวมทั้งหน่วยประมวลผลสำหรับโทรศัพท์พกพาทั่วไป เพื่อให้ใช้ประโยชน์สูงสุดกับหน่วยประมวลผลเหล่านี้ อัลกอริทึมเดิมจำเป็นต้องได้รับการออกแบบใหม่ ในงานวิจัยนี้ได้นำเสนอ การประมวลผลแบบขนานสำหรับอัลกอริทึมการจัดกลุ่มที่เรียกว่า ราฟ เค-มีดอยส์ (Rough K-Medoids) ซึ่งเป็นอัลกอริทึมที่นำเอาทฤษฎีทางด้าน ราฟเซต มาประยุกต์กับการจัดกลุ่มแบบ เค-มีดอยส์ วิธีการนี้ทำให้การจัดข้อมูลในแต่ละกลุ่มแบ่งออกเป็น 2 แบบคือ แบบที่ขึ้นอยู่กับกลุ่มใดกลุ่มหนึ่งเท่านั้น (lower approximation) และ แบบที่ขึ้นอยู่กับหลายๆ กลุ่มได้ (upper approximation) โดยนำอัลกอริทึมที่ออกแบบมาพัฒนาด้วยวิธีการโปรแกรมเชิงฟังก์ชันด้วยภาษา เออร์แลง (Erlang) ผลการทดลองแสดงให้เห็นว่าความเร็วของอัลกอริทึมแบบขนานสำหรับ ราฟ เค-มีดอยส์ นั้นดีขึ้น เมื่อเทียบกับอัลกอริทึมของ ราฟ เค-มีดอยส์แบบทำงานเป็นลำดับ

คำสำคัญ: หน่วยประมวลผลกลางแบบหลายแกน การประมวลผลแบบขนาน เหมืองข้อมูล การแบ่งกลุ่ม ราฟเค-มีดอยส์ ภาษาเออร์แลง

Abstract

Multi-core processors have recently been available on most personal computers, laptop computers and also smart phones. To get the maximum benefit of computational power from the multi-core architecture, we need a new design on existing algorithms. In this paper, we propose the parallelization of Rough K-Medoids clustering algorithms. In the Rough K-Medoids clustering, each cluster has been formed regarding the two approximations, a lower and an upper approximation. To make Rough K-Medoids clustering be better parallelized, we employ Erlang as a language for concurrent programming with functional paradigm. The experimental results demonstrate considerable speedup rate of the proposed parallel Rough K-Medoids clustering method, compared to the serial Rough K-Medoids approach.

Keywords: Multi-Core Processors, Parallel Computing, Data Mining, Clustering Rough K-Medoids, Erlang

บทนำ

ในปัจจุบัน เป็นยุคของเทคโนโลยีสารสนเทศและการสื่อสาร ไม่ว่าจะเป็นการดำเนินกิจกรรมใด ๆ ทั้งในทางธุรกิจ การศึกษา วิจัย หรือ การดำเนินงานของหน่วยงานภาครัฐ จำเป็นต้องยุ่งเกี่ยวกับปริมาณข้อมูลมหาศาล จึงเป็นไปได้ยากที่จะทำการแยกแยะ จัดกลุ่ม หาความสัมพันธ์ และ วิเคราะห์ข้อมูลเพื่อการจัดการข้อมูลได้อย่างมีประสิทธิภาพและนำมาใช้ประโยชน์ได้เหมาะสมที่สุด จึงต้องมีการนำเทคโนโลยีสมัยใหม่ ทางด้านเทคโนโลยีสารสนเทศและการสื่อสารเข้ามาช่วย วิธีการที่กำลังได้รับความนิยมในปัจจุบัน ในการแยกแยะ จัดกลุ่ม หาความสัมพันธ์และวิเคราะห์ข้อมูลคือ การทำเหมืองข้อมูล (Data Mining)

การทำเหมืองข้อมูลเป็นกระบวนการในการสกัดข้อมูลจากฐานข้อมูลขนาดใหญ่ เพื่อให้ได้สารสนเทศที่ยังไม่ทราบมาก่อน แล้วนำสารสนเทศที่ค้นพบมาสร้างแบบจำลองเพื่อการตัดสินใจทางธุรกิจหรือการคาดการทางด้านวิทยาศาสตร์ การทำเหมืองข้อมูลเป็นขั้นตอนหนึ่งของ "การค้นหาความรู้จากฐานข้อมูล (Knowledge Discovery in Database) แต่ปัจจุบันเวลากล่าวถึงการทำเหมืองข้อมูลจะหมายถึง การค้นหาความรู้จากฐานข้อมูลด้วยเช่นกัน

เทคนิคอย่างหนึ่งของการทำเหมืองข้อมูลเพื่อทำการแบ่งข้อมูลออกเป็นกลุ่มตามคุณลักษณะที่ใกล้เคียงกันเรียกว่า การทำ clustering การจัดกลุ่มข้อมูลก็เพื่อให้เห็นรูปแบบความคล้ายกันของข้อมูล ตัวอย่างเช่นธุรกิจค้าปลีกอาศัยการจัดกลุ่มของข้อมูลการซื้อสินค้าของลูกค้า เพื่อนำมาวิเคราะห์พฤติกรรมการซื้อสินค้าของลูกค้า หรือ การจัดกลุ่มผู้เข้าใช้งานเว็บไซต์โดยนำข้อมูลมาจากแฟ้มการเข้าใช้งานเว็บ (web log file) ที่ถูกจัดเก็บโดยโปรแกรม web server

อัลกอริทึมที่นิยมใช้ในการจัดกลุ่มข้อมูลคืออัลกอริทึม เค-มีน (K-Means clustering) [1-4] โดยทำการแบ่งข้อมูลออกเป็นจำนวน K กลุ่ม เริ่มจากเลือกค่าเริ่มต้นของค่าเฉลี่ยให้กับแต่ละกลุ่ม จากนั้นแบ่งข้อมูลเข้ากลุ่มที่ใกล้เคียงกับค่าเฉลี่ยปัจจุบันของแต่ละกลุ่มที่สุด หลังจากแบ่งเสร็จแล้วทำการหาค่าเฉลี่ยใหม่อีกครั้ง เมื่อได้ค่าเฉลี่ยใหม่จะทำการจัดกลุ่มใหม่ไปเรื่อย ๆ จนกว่าจะได้ตำแหน่งที่เสถียร K-Means นั้นง่ายต่อการเขียนโปรแกรมขึ้นมาใช้งาน แต่ว่าอาจจะทำให้เกิดข้อมูลที่แตกต่างไปจากกลุ่มอื่นๆ (Outliers) ได้ง่าย ที่จะเป็นผลลบอย่างมากต่อการวิเคราะห์ข้อมูล

อัลกอริทึมที่ได้รับการพัฒนาสืบเนื่องจาก K-Means คือ K-Medoids [5-7] ทั้งนี้เนื่องจากข้อจำกัดของ K-Means ที่ค่าคุณลักษณะต่าง ๆ ต้องเป็นตัวเลขเท่านั้น เมื่อนำมาใช้กับข้อมูลวัตถุที่ไม่เป็นตัวเลข จึงไม่สามารถคำนวณค่าเฉลี่ยได้ จึงได้เกิดอัลกอริทึม K-Medoids สำหรับจัดกลุ่มข้อมูลวัตถุ งานวิจัยนี้ใช้แนวทางการจัดกลุ่มข้อมูลด้วยอัลกอริทึม K-Medoids และเพิ่มความยืดหยุ่นให้มากขึ้นด้วยหลักการของ Rough Set

ได้มีการนำเสนอทฤษฎี Rough Set (Rough Set Theory) ขึ้นมาโดย Zdzislaw Pawlak [8] และเมื่อไม่นานมานี้ Lingras [3] ได้นำเสนออัลกอริทึมสำหรับการจัดกลุ่มข้อมูลที่เรียกว่า Rough K-Means clustering [9] ซึ่งทำการแบ่งข้อมูลออกเป็นกลุ่มที่ต่ำกว่ากับกลุ่มที่สูงกว่าโดยประมาณ เพื่อใช้จัดกลุ่มและดูความสัมพันธ์ของข้อมูลที่ในบางครั้งไม่อาจแบ่งกลุ่มได้อย่างชัดเจนแบบอัลกอริทึม K-Means

ต่อมา Georg Peters และ Martin Lampart [6] ได้นำเสนออัลกอริทึมในการจัดกลุ่มที่เรียกว่า Rough K-Medoids clustering ซึ่งได้นำทฤษฎี Rough Set มาประยุกต์กับ K-Medoids สำหรับจัดกลุ่มข้อมูลวัตถุออกเป็นกลุ่มที่ต่ำกว่าและสูงกว่าโดยประมาณ

การทำเหมืองข้อมูลด้วยวิธีการจัดกลุ่มนั้นต้องใช้เวลานานเพราะการจัดกลุ่มทำได้หลายรูปแบบ ขึ้นอยู่กับว่าจะเลือกคุณลักษณะใดเป็นเป้าหมายหลักของแต่ละกลุ่ม และ ต้องมีการวนซ้ำหลาย ๆ ครั้งเพื่อคำนวณว่าข้อมูลนั้นควรจะอยู่ในกลุ่มใดมากที่สุด

ในปัจจุบันเทคโนโลยีของหน่วยประมวลผลกลางได้พัฒนาให้มีประสิทธิภาพมากยิ่งขึ้น นอกจากจะเพิ่ม

ความเร็วของสัญญาณนาฬิกาแล้วปัจจุบันยังได้พัฒนาให้ภายในหนึ่งแผ่นของวงจรมีหลายแกนหลัก (multicore) [1, 4-5, 10-11] และราคาไม่แพงมาก ผู้ใช้คอมพิวเตอร์ส่วนบุคคลทั่วไปสามารถหาซื้อมาใช้ได้ แต่ยังไม่มีการพัฒนาโปรแกรมให้ใช้งานประสิทธิภาพของแกนหลักที่มีหลายๆ แกนนี้ได้อย่างเต็มที่เท่าที่ควร

งานวิจัยนี้จึงมองเห็นว่าควรที่จะนำเอาเทคโนโลยีทางการพัฒนาโปรแกรมสำหรับใช้งานหน่วยประมวลผลกลางแบบมีหลายแกนหลักภายในแผ่นวงจรหนึ่งแผ่นมาพัฒนาโปรแกรมสำหรับใช้งานทางด้านการทำเหมืองข้อมูล โดยเลือกอัลกอริทึมที่ใช้จัดกลุ่มข้อมูลที่เรียกว่า Rough K-Medoids clustering ที่ทำงานในแบบ sequential เพื่อให้สามารถแบ่งกลุ่มข้อมูลได้อย่างรวดเร็วและมีประสิทธิภาพกว่าการพัฒนาโปรแกรมแบบเดิม โดยงานวิจัยฉบับนี้จะเน้นการหาขั้นตอนของ Rough K-Medoids ที่เหมาะต่อการทำงานในแบบขนาน แล้วนำขั้นตอนนี้มาพัฒนาโปรแกรมบนหน่วยประมวลผลที่มีหลายแกนหลัก เพื่อให้เกิดการกระจายการทำงานให้ทำพร้อมๆ กัน แล้วนำผลลัพธ์การทำงาน เวลาที่ใช้ มาเปรียบเทียบกับการพัฒนาโปรแกรมแบบเดิม

วัตถุประสงค์ของการวิจัย

1. เพื่อศึกษาค้นคว้าและวิเคราะห์อัลกอริทึม Rough K-Medoids clustering ที่ใช้ในการจัดกลุ่มข้อมูลและลดเวลาการจัดแบ่งโดยการพัฒนาโปรแกรมให้ทำงานบนหน่วยประมวลผลกลางแบบหลายแกนหลักบนแผ่นวงจรเดียว
2. เพื่อศึกษาวิธีการพัฒนาโปรแกรมบนหน่วยประมวลผลกลางแบบหลายแกนหลัก แล้วนำมาประยุกต์ใช้กับการจัดกลุ่มข้อมูลด้วยอัลกอริทึม Rough K-Medoids ได้
3. หาเทคนิคในการที่จะลดเวลาของอัลกอริทึม Rough K-Medoids จากวิธีการเดิม ด้วยการแบ่งการคำนวณเป็นแบบขนานแล้วให้ทำงานบนหน่วยประมวลผลแบบหลายแกนหลักได้
4. เพื่อสรุปเทคนิคที่ใช้ในการพัฒนาโปรแกรมบนหน่วยประผลกลางแบบมีหลายแกนหลักเพื่อใช้จัดกลุ่มข้อมูล เพื่อเป็นแนวทางในการพัฒนาโปรแกรมทางด้านการทำเหมืองข้อมูลต่อไปได้ในอนาคต

วิธีดำเนินการวิจัย

ในการวิจัยนี้ จะมีวิธีการในการดำเนินการแยกออกเป็นขั้นตอนต่างๆ ดังนี้

ขั้นตอนที่ 1 ศึกษาวิธีการจัดกลุ่มข้อมูลด้วยอัลกอริทึมแบบต่าง ๆ

ในการจัดกลุ่มข้อมูลนั้นสามารถทำได้หลายวิธีด้วยกัน แต่ละแบบนั้นมีข้อดีข้อเสียแตกต่างกันขึ้นอยู่กับลักษณะของข้อมูลที่นำมาวิเคราะห์ในงานวิจัยนี้ได้ศึกษาวิธีการจัดกลุ่มของข้อมูลจากงานวิจัยที่ผ่านมา โดยเลือกศึกษาอัลกอริทึมดังต่อไปนี้

1) K-Means เป็นอัลกอริทึมในการจัดกลุ่มข้อมูลออกเป็นส่วนๆ อย่างชัดเจนโดยอาศัยการวัดระยะห่างระหว่างข้อมูลกับตำแหน่ง Means ของแต่ละกลุ่มโดยที่ค่า Means คือ ค่าเฉลี่ยของข้อมูลที่อยู่ในกลุ่ม ข้อดีของ K-Means คือ ง่ายในการพัฒนาโปรแกรม แต่ผลลัพธ์ที่ได้นั้นสามารถเกิด Outliers ได้ง่าย

2) K-Medoids เป็นการจัดกลุ่มข้อมูลโดยวัดระยะห่างข้อมูลเทียบกับตำแหน่ง Medoids ซึ่งวิธีการนี้จะใช้ตำแหน่งของข้อมูลจริงๆ เป็นค่า Medoids ของกลุ่มในแต่ละกลุ่มยังคงแยกข้อมูลจากกันอย่างชัดเจนเช่นเดียวกับ K-Means ข้อมูลไม่สามารถอยู่ได้หลายกลุ่มได้ ข้อดีของ K-Medoids คือ เกิด Outliers ได้น้อยกว่าเพราะใช้ข้อมูลจริงของแทนตำแหน่งกลางของกลุ่ม ข้อเสียคือ ใช้เวลานานและพัฒนาได้ยากกว่า K-Means เพราะว่าขั้นตอนในการหาตำแหน่ง Medoids ใหม่ต้องวนรอบเทียบกับข้อมูลทุกๆ ตัวในกลุ่ม

3) Rough K-Means ในอัลกอริทึมนี้จะนำทฤษฎี Rough Set มาประยุกต์ใช้กับการจัดกลุ่มแบบ K-Means โดยมองว่าการจัดกลุ่มโดยให้แต่ละข้อมูลอยู่ได้เพียงหนึ่งกลุ่มนั้นตามความเป็นจริงของข้อมูลอาจจะ

สามารถมีลักษณะหลาย ๆ อย่างที่สามารถอยู่ได้มากกว่าหนึ่งกลุ่ม วิธีการนี้จึงได้ใน Rough Set มาใช้โดยทำการแบ่งกลุ่มออกเป็นสองพื้นที่ด้วยกันเรียกว่า lower approximation และก็ upper approximation ซึ่งข้อมูลที่สามารถแบ่งเข้ากลุ่มได้อย่างชัดเจนจะอยู่ในส่วน lower approximation ส่วนข้อมูลที่สามารถอยู่ได้หลาย ๆ กลุ่มจะอยู่ในส่วนขอบของกลุ่มซึ่งคือพื้นที่ขอบด้านนอก lower approximation แต่ไม่เกิน upper approximation

4) Rough K-Medoids วิธีการนี้นำทฤษฎี Rough Set มาประยุกต์ใช้กับการแบ่งข้อมูลแบบ K-Medoids ซึ่งจะแบ่งข้อมูลออกเป็นสองส่วนเช่นเดียวกันกับ Rough K-Means แต่ว่าในการคำนวณจะใช้ตำแหน่ง Medoids เป็นค่ากลางเหมือนกับ K-Medoids ธรรมดา

ขั้นตอนที่ 2 ศึกษาการประมวลผลแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกน

ศึกษาการประมวลผลแบบขนานเพื่อนำมาออกแบบอัลกอริทึมสำหรับการจัดกลุ่มข้อมูลแบบ Rough K-Medoids จากการศึกษางานวิจัยที่ผ่านมาก็จะพบว่า รูปแบบสถาปัตยกรรมในการทำงานแบบขนานที่นิยมใช้กันก็คือแบบ Single Instruction Multiple Data (SIMD) คือ การแบ่งข้อมูลออกเป็นหลาย ๆ กลุ่มแล้วส่งให้ทำงานหนึ่งคำสั่งที่สามารถทำไปพร้อม ๆ กันได้บนหน่วยประมวลผลคำสั่งหลาย ๆ ตัว ซึ่งเทคโนโลยีทางด้านอุปกรณ์คอมพิวเตอร์สามารถนำมาต่อกันเพื่อให้ทำงานตามสถาปัตยกรรมนี้ได้โดยใช้หลาย ๆ เครื่องติดต่อกันผ่านทางระบบเน็ตเวิร์ก เพราะว่าในอดีตยังไม่ได้มีการพัฒนาให้วงจรของหน่วยประมวลผลกลางมีแกนหลักในการประมวลผลหลายแกนแบบในปัจจุบัน ในปัจจุบันเทคโนโลยีปัจจุบันมีการพัฒนาหน่วยประมวลผลกลางแบบที่มีหลายแกนหลักขึ้นมา ทำให้เราสามารถพัฒนาโปรแกรมแบบขนานได้โดยเครื่องคอมพิวเตอร์ส่วนบุคคลทั่วไป ไม่จำเป็นต้องต่อหลายเครื่องแบบในอดีต การพัฒนาโปรแกรมแบบขนานจึงจะทำให้สามารถใช้งานหน่วยประมวลผลกลางแบบหลายแกนหลักได้อย่างมีประสิทธิภาพมากที่สุดเท่าที่จะเป็นไปได้ ซึ่งวิธีการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนที่ทำการศึกษาพบว่าใช้เทคโนโลยีดังต่อไปนี้

1) Erlang เป็นภาษาแบบ fuctional language ที่สามารถแยกการประมวลผลออกเป็นหลายๆ โปรเซส การทำงานของโปรแกรมภาษา Erlang จะอาศัย Virtual Machine(VM) ในการประมวลผลซึ่ง VM ของ Erlang นั้นสามารถจะส่งการทำงานของโปรเซสที่แยกกันทำงานให้กระจายไปบนแกนหลักของหน่วยประมวลผลแบบหลายแกนได้เองอัตโนมัติ ทำให้ง่ายในการพัฒนา ไม่จำเป็นต้องเขียนเจาะจงว่าจะให้ทำงานกี่แกนหลัก หรือไม่จำเป็นต้องมีคำสั่งพิเศษ หรือ library พิเศษเพิ่มเข้ามาในภาษา

2) OpenMP เป็นเทคโนโยลีในการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนโดยใช้ภาษาซีในการพัฒนาซึ่งต้องเพิ่ม library ของ OpenMP เขามาด้วยในการแปลโค้ดโปรแกรม วิธีการใช้งานของ OpenMP จะแยกการทำงานออกเป็นเทรดย่อยๆ โดยเราไม่จำเป็นต้องเขียนโค้ดกำหนดว่าจะให้การทำงานทำงานบนเทรดไหน แต่จะต้องกำหนดมาโครเพื่อบอกว่าการทำงานส่วนไหนที่ต้องการให้ทำงานแบบขนาน แล้วตัว OpenMP จะกระจายไปทำงานบนหลาย ๆ เทรดให้เอง

3) CUDA เป็นเทคโนโลยีในการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกลางสำหรับการประมวลผลกราฟฟิคที่มีหลายแกนหลัก ซึ่งเป็นเทคโนโลยีของบริษัท NVIDIA โดยสถาปัตยกรรมของการ์ดกราฟฟิคนี้จะแบ่งข้อมูลออกเป็น block ในแต่ละ block จะประกอบด้วย เทรดสำหรับการทำงานแบบขนานและเมื่อเรากระจายข้อมูลลงไปบน block ต่าง ๆ เทรดแต่ละเทรดก็จะประมวลผลโดยเลือกข้อมูลใน block นั้น ๆ มาประมวลผล

ขั้นตอนที่ 3 ออกแบบอัลกอริทึมสำหรับ Rough K-Medoids สำหรับการประมวลผลแบบขนาน แบ่งออกเป็นส่วนต่าง ๆ ได้ดังต่อไปนี้

1) ส่วนรับข้อมูลเริ่มต้นสำหรับโปรแกรม

ส่วนแรกนี้เป็นส่วนรับค่าเริ่มต้นต่าง ๆ ของโปรแกรมซึ่งมีค่าที่ต้องการดังต่อไปนี้

- แฟ้มเก็บข้อมูลที่ต้องการแบ่ง

- จำนวนของข้อมูลที่ต้องการแบ่ง (N)
- จำนวนของกลุ่มที่ต้องการแบ่ง (K)
- ค่าคงที่น้ำหนักของส่วน lower approximation (w_l)
- ค่าคงที่น้ำหนักของส่วน upper approximation (w_b)
- ค่าคงที่ threshold (ε)
- จำนวนของโปรเซสที่ต้องการแบ่ง

2) ส่วนการแบ่งข้อมูล

ส่วนนี้จะทำหน้าที่นำข้อมูลที่ได้แบ่งออกข้อมูลออกเป็นส่วนโดยแบ่งให้แต่ละโปรเซสเพื่อคำนวณระยะทางและจัดกลุ่มข้อมูลให้แต่ละกลุ่ม เมื่อแต่ละโปรเซสทำงานเสร็จจะรวบรวมส่งมาให้โปรเซสหลัก

3) ส่วนการหาตำแหน่ง Medoids ใหม่

หลังจากจัดกลุ่มเสร็จแล้วจะทำการแบ่งโปรเซสตามจำนวนของกลุ่มเพื่อส่งข้อมูลของแต่ละกลุ่มไปหาตำแหน่ง Medoids ใหม่เมื่อเสร็จแล้วส่งค่าของ Medoids ใหม่ให้กับโปรเซสหลัก

ขั้นตอนที่ 4 ทดสอบการทำงานและประเมินผล

ในการทดสอบและประเมินผลจะสุ่มสร้างข้อมูลสังเคราะห์ขึ้นมาทดสอบ โดยวิธีการทดสอบเปรียบเทียบเวลาในการทำงานระหว่างวิธีการแบบขนานและวิธีการแบบลำดับ ซึ่งทำการทดลอง 3 แบบดังต่อไปนี้

- ทดสอบเทียบเวลาเมื่อ เพิ่มจำนวนโปรเซส เพื่อหาว่าจำนวนเท่าไหร่ให้ประสิทธิภาพดีที่สุดโดยกำหนดจำนวนข้อมูล 5000 และจำนวนกลุ่มที่ต้องการแบ่ง 2 กลุ่มเท่ากัน เพื่อนำจำนวนที่ได้ไปใช้ในการทดลองถัดไป
- ทดสอบเทียบเวลาระหว่างการทำงานแบบลำดับ 1 โปรเซส กับการทำงานแบบขนานด้วยจำนวนโปรเซสที่ดีที่สุดจากการทดลองแรก โดยทำการเพิ่มข้อมูลจาก 1000, 2000, 3000, 4000 ถึง 5000 และจำนวนกลุ่มเท่ากันที่ 2 กลุ่ม
- ทดสอบเทียบเวลาระหว่างการทำงานแบบลำดับ 1 โปรเซส กับการทำงานแบบขนานด้วยจำนวนโปรเซสที่ดีที่สุดจากการทดลองแรก โดยทำการเพิ่มมิติของข้อมูลจาก 1, 2 ถึง 3 โดยจำนวนข้อมูลคงที่ 1000 ข้อมูล

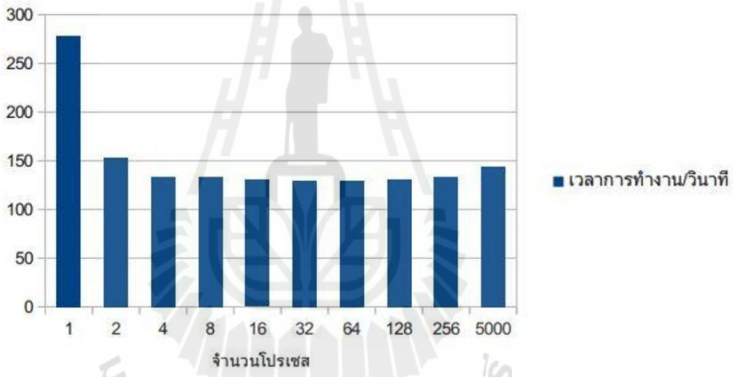
ผลการวิจัย

ผลการวิจัยในแต่ละการทดลองเป็นดังต่อไปนี้

1) เวลาในการทำงานของการทดลอง ทดสอบเทียบเวลาเมื่อ เพิ่มจำนวนโปรเซส เพื่อหาว่าจำนวนเท่าไหร่ให้ประสิทธิภาพดีที่สุดโดยกำหนดจำนวนข้อมูล 5000 และจำนวนกลุ่มที่ต้องการแบ่ง 2 กลุ่มเท่ากัน ได้ค่าออกมาดัง ตารางที่ 1 และเมื่อนำมาสร้างกราฟเปรียบเทียบจะได้ดังภาพที่ 1

ตารางที่ 1 ผลการทดลองจับเวลาเมื่อเพิ่มจำนวนโปรเซส

จำนวนโปรเซส	เวลาการทำงาน (วินาที)
1	278.24
2	153.27
4	133.22
8	133.93
16	130.21
32	129.29
64	129.58
128	130.69
256	133.17
5000	144.57



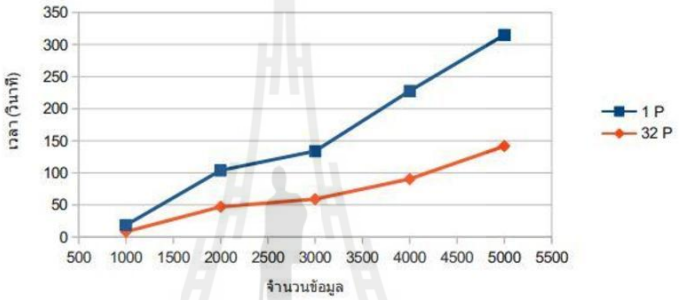
ภาพที่ 1 กราฟแสดงเวลาที่ใช้เมื่อจำนวนโปรเซสต่างกัน

จากกราฟแสดงให้เห็นว่า เมื่อทำการประมวลผลแบบขนานโดยเพิ่มจำนวนโปรเซส จะเห็นว่ามีการทำงานที่เร็วกว่าแบบลำดับที่ใช้เพียง 1 โปรเซสอยู่มาก โดยในการทดลองนี้จำนวนโปรเซสที่ให้ผลดีสุดคือ 32 โปรเซส ซึ่งเร็วกว่าแบบลำดับอยู่ถึง 2.15 เท่า

2) เวลาในการทำงานของการทดลอง ทดสอบเทียบเวลาระหว่างการทำงานแบบลำดับ 1 โปรเซส กับการทำงานแบบขนานด้วยจำนวนโปรเซสที่ดีที่สุดจากการทดลองแรก โดยทำการเพิ่มข้อมูลจาก 1000, 2000, 3000, 4000 ถึง 5000 และจำนวนกลุ่มเท่ากันที่ 2 กลุ่ม ได้ค่าออกมาดัง ตารางที่ 2 และเมื่อนำมาสร้างกราฟเปรียบเทียบจะได้ดังภาพที่ 2

ตารางที่ 2 ผลการจับเวลาเมื่อเพิ่มจำนวนข้อมูล

จำนวนข้อมูล	เวลาของแบบลำดับ 1 โปรเซส (วินาที)	เวลาของแบบขนาน 32 โปรเซส (วินาที)
1000	18.56	8.27
2000	103.82	47.09
3000	133.73	58.97
4000	227.65	90.41
5000	314.94	141.71



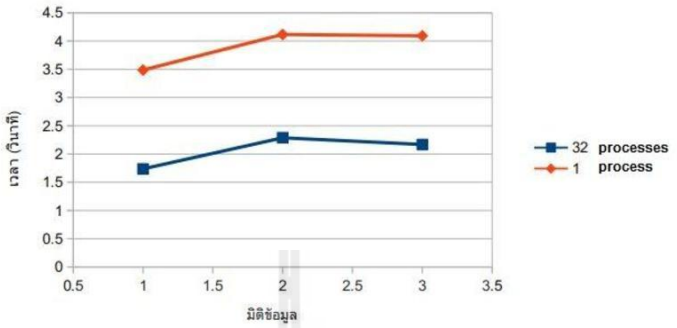
ภาพที่ 2 กราฟแสดงเวลาเปรียบเทียบระหว่าง เมื่อจำนวนข้อมูลต่างกันของแบบลำดับและขนาน

จากกราฟแสดงให้เห็นว่า เมื่อเพิ่มจำนวนข้อมูลแล้วเวลาของการทำงานแบบลำดับจะเพิ่มขึ้นเรื่อย ๆ แต่เวลาการทำงานของแบบขนานที่ 32 โปรเซสก็เพิ่มขึ้น ก็ก็ยังใช้เวลาน้อยกว่าแบบลำดับตลอดช่วงการทดลองของจำนวนข้อมูลที่ 1000 ถึง 5000

3) เวลาในการทำงานของการทดลอง ทดสอบเทียบเวลาระหว่างการทำงานแบบลำดับ 1 โปรเซส กับการทำงานแบบขนานด้วยจำนวนโปรเซสที่ดีที่สุดจากการทดลองแรก โดยทำการเพิ่มมิติของข้อมูลจาก 1, 2 ถึง 3 โดยจำนวนข้อมูลคงที่ 1000 ข้อมูล ได้ค่าออกมาดัง ตารางที่ 3 และเมื่อนำมาสร้างกราฟเปรียบเทียบจะได้ดังภาพที่ 3

ตารางที่ 3 ผลการจับเวลาเมื่อเพิ่มจำนวนมิติข้อมูล

มิติข้อมูล	เวลาของแบบลำดับ 1 โปรเซส (วินาที)	เวลาของแบบขนาน 32 โปรเซส (วินาที)
1	3.49	1.74
2	4.12	2.29
3	4.09	2.17



ภาพที่ 3 กราฟแสดงเวลาเปรียบเทียบระหว่างจำนวนมิติที่ต่างกันของแบบลำดับและขนาน

จากกราฟแสดงให้เห็นว่า เมื่อเพิ่มจำนวนมิติข้อมูลการทำงานของแบบขนานก็ยังเร็วกว่าแบบลำดับโดยรูปร่างของเส้รจกราฟเป็นไปในลักษณะเดียวกันแต่ว่าแบบขนานใช้เวลาน้อยกว่าเสมอ

สรุปและอภิปรายผล

งานวิจัยนี้เป็นการนำเอาวิธีการแบ่งกลุ่มแบบ Rough K-Medoids ซึ่งเป็นวิธีการแบ่งกลุ่มที่มีประสิทธิภาพและใช้กับข้อมูลที่อาจจะขึ้นกับหลายกลุ่มได้ แต่ว่าใช้การคำนวณที่มากกว่าแบบที่นิยมกันคือ K-Means มาออกแบบกระบวนการทำงานใหม่โดยใช้การประมวลผลแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนเพื่อเพิ่มประสิทธิภาพในการคำนวณให้เวลาลดลงจากเดิม

ผลการวิจัยพบว่า สามารถลดเวลาการทำงานของการแบ่งกลุ่มแบบ Rough K-Medoids ลงไปได้ประมาณ 2.15 เท่า เมื่อจำนวนโปรเซสที่ใช้เท่ากับ 32 และแม้ว่าจะทำการเพิ่มจำนวนข้อมูล หรือเพิ่มจำนวนมิติของข้อมูลการทำงานแบบขนานก็ยังให้ประสิทธิภาพที่ดีกว่า

ในงานวิจัยนี้ผู้เขียนได้ใช้ภาษา เออร์แลงเป็นภาษาเชิงฟังก์ชันในการเขียนโปรแกรมเพื่อทดสอบ เนื่องจากมีความง่ายในการเขียนโปรแกรมเพื่อใช้งานแบบขนานบนหน่วยประมวลผลกลางแบบหลายแกน และเขียนได้สั้นกระชับ อ่านง่ายกว่าการเขียนด้วยภาษาอื่นให้ทำงานบนหน่วยประมวลผลกลางแบบหลายแกน แต่ว่าถ้าต้องการประสิทธิภาพความเร็วสูงกว่านี้ เราสามารถแปลงจากภาษาเออร์แลงไปเป็นภาษาอื่นที่ประสิทธิภาพดีกว่าเช่น ภาษาซี ได้โดยง่าย ง่ายกว่าการจะเริ่มต้นที่ภาษาซีตั้งแต่ต้น

กิตติกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนจากมหาวิทยาลัยเทคโนโลยีสุรนารี

เอกสารอ้างอิง

[1] Kerdprasop, K.; & Kerdprasop, N. (2010). Parallelization of K-means Clustering on Multi-core Processors. In *Proceedings of 10th WSEAS International Conference on Applied Computer Science, Japan*. pp. 472-477. n.p.

[2] Oyelade, O.J.; Oladipupo, O.O.; & Obagbuwa, I.C. (2010). Application of k-Means Clustering algorithm for prediction of Students' Academic Performance. *International Journal of Computer Science and Information Security.* 7(1): 292-295.

[3] Lingras, P.; & West, C. (2002). Interval set clustering of web users with rough k-means. In *Technical Report 2002-002, Department of Mathematics and Computer Science, St. Mary's University, Halifax, Canada.* Canada: St. Mary's University, Halifax.

[4] Farivar, R.; et al. (2008). A parallel implementation of k-means clustering on GPUs. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA).* pp. 340-345. n.p.

[5] Laszewski, G.; & Roberts, D. (2009). *K-Medoids: CUDA Implementation. Service Oriented Cyberinfrastructure Lab, Rochester Institute of Technology.* n.p.

[6] Peters, G.; & Lampart, M. (2006). A partitive rough clustering algorithm. In *Proc RSCTC 2006-Fifth Int Conf on Rough Sets and Current Trends in Computing. Lecture Notes in Artificial Intelligence.* pp. 657-666. n.p.

[7] Mitra, S. (2004). An evolutionary rough partitive clustering. In *Pattern Recognition Letters.* pp. 1439-1449. n.p.

[8] Pawlak, Z. (1982). Rough Sets. *International Journal of Computer and Information Science.* 11: 341-356.

[9] Zhou, T.; et al. (2007). Rough k-means cluster with adaptive parameters. In *Proceedings of 6th. International Conference on Machine Learning and Cybernetics, (ICMLC'07).* pp. 3063–3068. n.p.

[10] Barney, B. (2011). *Introduction to Parallel Computing.* from https://computing.llnl.gov/tutorials/parallel_comp/

[11] Wang, H.; et al. (2008). Parallel Clustering Algorithms for Image Processing on Multi-core CPUs. In *Proceedings of International Conference on Computer Science and Software Engineering (CSSE).* pp. 450-53. n.p.

ภาคผนวก ข

รหัสต้นฉบับของโปรแกรม Parallel Rough K-Medoids clustering

```
-module(parallel_rough_k_medoids).
-compile(export_all).

distance(Object1,Object2) ->
    math:sqrt(
      lists:foldl(
       fun({X,Y},S) ->
              S + (X-Y)*(X-Y)
       end,
       0,
       lists:zip(
         tuple_to_list(Object1),
         tuple_to_list(Object2)
        )
       )
     ).

near_medoid(Medoids, Object) ->
    {NearMedoid,_} = lists:foldl(
                     fun(Medoid,{MinMedoid,DistMin}) ->
                            Dist = distance(Object, Medoid),
                            if Dist < DistMin ->
                                   {Medoid, Dist};
                              true ->
                                   {MinMedoid, DistMin}
                            end
                     end,
                     {null,999999999.0},
                     Medoids),
    NearMedoid.


objective_criteria(Objects, Medoid, OldMedoid) ->
    lists:foldl(
      fun(Object,Acc) ->
             Acc + distance(Medoid,Object) - distance(OldMedoid,Object)
      end,
      0,
      Objects).
```

```
rough_objective_criteria(Objects, Medoid, OldMedoid, Wl, Wb) ->
    lists:foldl(
      fun({Approx,Object}, Acc) ->
            if Approx =:= lower ->
                  Acc + (Wl*(distance(Medoid,Object) -
distance(OldMedoid,Object)));
               true ->
                  Acc + (Wb*(distance(Medoid,Object) -
distance(OldMedoid,Object)))
            end
      end,
      0,
      Objects).

read_file(File,Result) ->
    case file:read_line(File) of
        eof ->
            Result;
        {ok, Line} ->
            Data = lists:map(fun(X) ->
                                case string:to_float(X) of
                                   {error,no_float} ->
                                      {F,_} = string:to_float(X++".0"),
                                      F;
                                   {F,_} -> F
                                end
                             end,
                             string:tokens(string:strip(Line,right,$\n),",")
                            ),
            read_file(File,[list_to_tuple(Data) | Result])
    end.
```

```
getRandomMedoid(K, Data, _Acc) ->
    InitMedoid = lists:sublist(Data,K),
    RestData = Data -- InitMedoid,
    {InitMedoid, RestData}.

getInitMedoid(K,Data) ->
    {MedoidList,NoneMedoidList} = getRandomMedoid(K,Data,[]),
    MedoidTable = ets:new(medoids,[bag]),
    {MedoidTable,MedoidList,NoneMedoidList}.
```

```
%% cluster member
clustering(MedoidTable,MedoidList,NoneMedoids, Threshold) ->
    %% closest medoid belong to upper & lower approximation
    lists:foreach(
      fun(D) ->
             Near = near_medoid(MedoidList,D),
             ets:insert(MedoidTable,{Near,{lower,D}})
      end,
      NoneMedoids),

    %% check belong to upper approximation of another cluster
    lists:foreach(
      fun(M) ->
             UpperApprox = case ets:member(MedoidTable,M) of
                           true ->
                               [{lower,M} |
ets:lookup_element(MedoidTable,M,2)];
                           false ->
                               [{lower,M}]
                       end,
             lists:foreach(
              fun(K) ->
                     {R,Km} = K,
                     if R == lower ->
                           T = lists:foldr(
                             fun(MH,Acc) ->
                                   DH = abs(distance(M,Km) -
distance(MH,Km)),
                                   if DH < Threshold ->
                                        [MH|Acc];
                                    true ->
                                        Acc
                                   end

                             end,
                             [],
                             MedoidList -- [M]),
                           if T /= [] ->
                              ets:delete_object(MedoidTable,{M,K}),

       ets:insert(MedoidTable,{M,{upper,Km}}),

                                lists:foreach(
                                  fun(Tm) ->

ets:insert(MedoidTable,{Tm,{upper,Km}})
                                  end,
                                  T);

                              true -> pass
                           end;
                       true ->
                           pass
                     end
              end,
              UpperApprox
              )
      end,
      MedoidList),
    MedoidTable.
```

```
clusteringAll(MedoidTable,MedoidList,NoneMedoidList,NumOfProcess,Wl,Thre
shold) ->
    Wb = 1.0 - Wl,
    %% fill table
    ets:delete_all_objects(MedoidTable),
    {Time,_} =
timer:tc(?MODULE,clustering,[MedoidTable,MedoidList,NoneMedoidList,Thres
hold]),
    io:format("~p~n",[MedoidList]),
    %%io:format("~p , Time = ~p ~n",[MedoidList,Time / 1000000]),
    %% swap medoids
    {Time2,{NewMedoidList,NewNoneMedoidList}} =
timer:tc(?MODULE,calculateNewMinMedoid,[MedoidTable, MedoidList,
NoneMedoidList,NumOfProcess,Wl,Wb]),
    io:format("Time2 = ~p ~n",[Time2 / 1000000]),

    if ((NewMedoidList == MedoidList) and (NewNoneMedoidList ==
NoneMedoidList)) ->
           %% fill table
           ets:delete_all_objects(MedoidTable),
           clustering(MedoidTable,MedoidList,NoneMedoidList,Threshold),
           {MedoidTable,MedoidList,NoneMedoidList};
       true ->

clusteringAll(MedoidTable,NewMedoidList,NewNoneMedoidList,NumOfProcess,W
l,Threshold)
    end.
```

```
calculateNewMinMedoidProcess(Parent, MedoidTable, MedoidList,
NoneMedoidList,Start,End,Wl,Wb) ->
    AllDistance = lists:map(fun(Index) ->
                              NM = lists:nth(Index,NoneMedoidList),
                              lists:map(fun(MM) ->
                                            Member = case
ets:member(MedoidTable,MM) of
                                                       true ->

[{lower,MM}|ets:lookup_element(MedoidTable,MM,2)];
                                                       false ->

[{lower,MM}]
                                                   end,
                                            DiffNewDistance =
rough_objective_criteria(Member,NM,MM,Wl,Wb),
                                            {NM,MM,DiffNewDistance}
                                        end,

                                        MedoidList)
                        end,
                        lists:seq(Start,End)),
    BestMinDistance =
lists:foldl(fun({NM,MM,NewDistance},{NM2,MM2,NewDistance2}) ->
                                if NewDistance < NewDistance2 ->
                                     {NM,MM,NewDistance};
                                   true ->
                                     {NM2,MM2,NewDistance2}
                                end
                            end,
                            {-1,-
1,9999999999},lists:flatten(AllDistance)),

    Parent ! BestMinDistance.
```

```
receiveAllDistance(0,Acc) ->
    lists:reverse(Acc);
receiveAllDistance(Div,Acc) ->
    receive
       AllDistance ->
           receiveAllDistance(Div - 1, [AllDistance|Acc])
    end.
calculateNewMinMedoid(MedoidTable, MedoidList, NoneMedoidList,Div,Wl,Wb)
->
    LengthNoneMedoidList = length(NoneMedoidList),
    {TimeSpawn,_} = timer:tc(lists,map,[fun(Index) ->
                                        StartList = 1 +
((LengthNoneMedoidList div Div)*Index),
                                        EndList = (LengthNoneMedoidList
div Div)*(Index+1),

       spawn(?MODULE,calculateNewMinMedoidProcess,[self(),MedoidTable,
MedoidList, NoneMedoidList,StartList,EndList,Wl,Wb])
                                    end,
                                    lists:seq(0,Div-1)]),

    AllDistance2 = receiveAllDistance(Div,[]),

    BestMinDistance =
lists:foldl(fun({NM,MM,NewDistance},{NM2,MM2,NewDistance2}) ->
                                      if NewDistance < NewDistance2 ->
                                            {NM,MM,NewDistance};
                                         true ->
                                            {NM2,MM2,NewDistance2}
                                      end
                              end,
                              {-1,-1,9999999999},AllDistance2),
    {NM,MM,BestDistance} = BestMinDistance,

    %% swap best non medoid and medoid that best RCPC
    if BestDistance < 0 ->
           {KeepLeft,[_|KeepRight]} = lists:splitwith(fun(E) -> E =/= NM
end, NoneMedoidList),
           {KeepMLeft,[_|KeepMRight]} = lists:splitwith(fun(E) -> E =/=
MM end, MedoidList),
           NewNoneMedoidList = KeepLeft ++ [MM | KeepRight],
           NewMedoidList = KeepMLeft ++ [NM | KeepMRight],
           {NewMedoidList,NewNoneMedoidList};
       true ->
           {MedoidList,NoneMedoidList}
    end.
```

```
%% main time check
k_medoids_timer(K,Data,NumOfProcess,Wl,Threshold) ->
    {Time,_} =
timer:tc(?MODULE,k_medoids,[K,Data,NumOfProcess,Wl,Threshold]),
    Time / 1000000.

k_medoids_file_timer(K,FileName,NumOfProcess,Wl,Threshold) ->
    {ok,File} = file:open(FileName,read),
    DataSet = read_file(File,[]),
    file:close(File),
    k_medoids_timer(K,DataSet,NumOfProcess,Wl,Threshold).

k_medoids_file(K,FileName,NumOfProcess,Wl,Threshold) ->
    {ok,File} = file:open(FileName,read),
    DataSet = read_file(File,[]),
    file:close(File),
    k_medoids(K,DataSet,NumOfProcess,Wl,Threshold).
%% main
k_medoids(K,Data,NumOfProcess,Wl,Threshold) ->
    {MedoidTable,MedoidList,NoneMedoidList} = getInitMedoid(K, Data),
    {MedoidTable,NewMedoidList,_NewNoneMedoidList} =
clusteringAll(MedoidTable,MedoidList,NoneMedoidList,NumOfProcess,Wl,Thre
shold),
    lists:foreach(
      fun(M) ->

io:format("================================================================
====~n"),
             case ets:member(MedoidTable,M) of
                true ->
                    lists:foreach(fun(MM) ->
                                          io:format("~w~n",[MM])
                                  end,

[{lower,M}|ets:lookup_element(MedoidTable,M,2)]
                                  );
                false ->
                    io:format("~w~n",[[{lower,M}]])
             end
      end,
      NewMedoidList
     ),
    ets:delete(MedoidTable),
    ok.
```

ประวัติผู้เขียน

นายวีรศักดิ์ ช่องงูเหลือม เกิดเมื่อวันที่ 21 มิถุนายน พ.ศ. 2528 เกิดที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาตรี วิศวกรรมศาสตรบัณฑิต (วิศวกรรมคอมพิวเตอร์) จากมหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา เมื่อ พ.ศ. 2551 ภายหลังจากสำเร็จการศึกษาได้เข้าทำงานกับบริษัท ดับเบิ้ลไอ สตูดิโอ จำกัด เป็นเวลา 2 ปี จึงได้เข้าศึกษาต่อในระดับปริญญาโทในสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2553 ในระหว่างการศึกษาได้รับความไว้วางใจให้เป็นผู้ช่วยวิจัย และผู้สอนปฏิบัติการของสาขาวิศวกรรมคอมพิวเตอร์ในรายวิชา Object Oriented Technology, Event Driven Programming, และ Database System

ผลงานวิจัย : ได้เสนอบทความเข้าร่วมในงานประชุมวิชาการ “ศรีนครินทรวิโรฒวิชาการ” ครั้งที่ 6 ประจำปี 2555 เรื่อง การจัดกลุ่มข้อมูลแบบ ราฟ เค-มีดอยส์ แบบขนานบนหน่วยประมวลผลกลางแบบหลายแกนหลัก