รหัสโครงการ SUT7-705-49-24-47

รายงานการวิจัย

การสุ่มแบบลำดับเพื่อการจัดกลุ่มข้อมูลที่มีขนาดและความหนาแน่นแตกต่างกัน

(Sequential random sampling for clustering data of different sizes and densities)

ได้รับทุนอุดหนุนการวิจัยจาก

มหาวิทยาลัยเทคโนโลยีสุรนารี

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

รหัสโครงการ **SUT7-705-49-24-47**

รายงานการวิจัย

การสุ่มแบบลำดับเพื่อการจัดกลุ่มข้อมูลที่มีขนาดและความหนาแน่นแตกต่างกัน

(Sequential random sampling for clustering data of different sizes and densities)

ผู้วิจัย

หัวหน้าโครงการ

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2549 และ 2550

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

พฤษภาคม 2553

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณมหาวิทยาลัยเทคโนโลยีสุรนารี และสำนักงานคณะกรรมการวิจัยแห่งชาติที่ได้จัดสรรงบประมาณในการดำเนินงานวิจัยให้ในปีงบประมาณ 2549 และ 2550 นอกจากการสนับสนุนในด้านงบประมาณแล้ว โครงการวิจัยนี้ยังได้รับความร่วมมือในการดำเนินงานจากหน่วยปฏิบัติการวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้ (Data Engineering and Knowledge Discovery -- DEKD -- Research Unit) ซึ่งเป็นหน่วยวิจัยในสังกัดสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ผู้วิจัยขอขอบคุณผู้ทรงคุณวุฒิที่ได้เสียสละเวลาทำหน้าที่ตรวจข้อเสนอโครงการ และตรวจร่างรายงานการวิจัยฉบับสมบูรณ์

บทคัดย่อภาษาไทย

การจัดกลุ่มข้อมูล เป็นงานรวมกลุ่มข้อมูลด้วยการพิจารณาความคล้ายคลึงกัน อัลกอริทึม k-means เป็นอัลกอริทึมมาตรฐานที่ทำการรวมกลุ่มข้อมูล โดยเริ่มต้นกระบวนการด้วยการคำนวณระยะห่างเพื่อกำหนดข้อมูลแต่ละตัวเข้ากลุ่มที่อยู่ใกล้ที่สุด จากนั้นคำนวณค่าจุดกึ่งกลางของแต่ละกลุ่มโดยใช้ค่าเฉลี่ยของข้อมูลทั้งหมดในกลุ่ม อัลกอริทึมจะทำสองขั้นตอนนี้ซ้ำจนกระทั่งค่าจุดกึ่งกลางไม่มีการเปลี่ยนแปลงและข้อมูลไม่มีการเปลี่ยนกลุ่ม ประสิทธิภาพของอัลกอริทึมจะขึ้นกับจำนวนข้อมูล จำนวนกลุ่ม และจำนวนรอบในการวนซ้ำ ผู้วิจัยได้พัฒนาวิธีการสุ่มตามความหนาแน่นเพื่อลดจำนวนข้อมูลที่จะใช้ในการจัดกลุ่ม ซึ่งจะช่วยให้อัลกอริทึมจัดกลุ่มข้อมูลทำงานได้เร็วขึ้น การสุ่มข้อมูลจะใช้วิธีการคัดเลือกข้อมูลตัวแทนจากบริเวณที่มีความหนาแน่นสูง อัลกอริทึมที่พัฒนาขึ้นนี้ได้รับการแปลงเป็นโปรแกรมที่เขียนด้วยภาษาเออแลง ซึ่งเป็นภาษาเชิงฟังก์ชันที่ช่วยให้การพัฒนาโปรแกรมต้นแบบทำได้อย่างรวดเร็ว และจากผลการทดสอบโปรแกรมพบว่าโปรแกรมสุ่มข้อมูลและจัดกลุ่มข้อมูลตามความหนาแน่นสามารถทำงานได้ดีกับข้อมูลที่มีการกระจายแบบซิฟ ผลการจัดกลุ่มด้วยข้อมูลสุ่มพบว่าจุดกึ่งกลางกลุ่มเบี่ยงเบนไปจากจุดกึ่งกลางที่แท้จริงเพียงเล็กน้อย เมื่อวัดประสิทธิภาพการใช้เนื้อที่หน่วยความจำของโปรแกรมที่พัฒนาขึ้น พบว่าใช้เนื้อที่หน่วยความจำน้อยกว่าโปรแกรมจัดกลุ่มข้อมูลตามปกติถึง 60% ดังนั้นเทคนิคและโปรแกรมที่พัฒนาขึ้นจึงมีศักยภาพที่จะพัฒนาให้ทำงานกับข้อมูลสตรีมได้ นอกจากนี้ภาษาเออแลงยังมีขีดความสามารถในการโปรแกรมแบบพร้อมกันกับหลายหน่วยประมวลผลได้ ซึ่งจะช่วยให้โปรแกรมทำงานได้เร็วขึ้นและรองรับข้อมูลปริมาณมากขึ้นได้

บทคัดย่อภาษาอังกฤษ

Clustering is a task of grouping data based on similarity. A standard k-means algorithm groups data by firstly assigning all data points to the closest clusters, then determining the new cluster mean of each cluster based on the average value of its members. The algorithm repeats these two steps until it converges; that is until there is no change in cluster means and cluster assignment among data points. Performance of the algorithm depends on the number of data points, number of data clusters, and number of iterations. To speed up the clustering process, we develop the density-biased reservoir sampling algorithm as an efficient data reduction technique. Instead of simply randomly selecting data for clustering, we evaluate data density and draw samples from the dense area. The proposed algorithm has been implemented with a functional programming paradigm using the Erlang language. The declarative style of Erlang facilitates a rapid prototyping. Our experimental results reveal the effectiveness of drawing samples of high density from the Zipf distributed data. The shift of cluster means is minimal, whereas the decrease in memory usage is significant. The proposed density-biased reservoir sampling technique thus shows a great potential on dealing with large and streaming data. Moreover, the Erlang language itself has an important feature of concurrent programming on a multi-core architecture that can help speed up the computation of large and continuously generated data.

สารบัญ

	หน้า
กิตติกรรมประกาศ	ก
บทคัดย่อภาษาไทย	ข
บทคัดย่อภาษาอังกฤษ	ค
สารบัญ	ง
สารบัญตาราง	ฉ
สารบัญภาพ	ช
บทที่ 1 บทนำ	
ความสำคัญและที่มาของปัญหาการวิจัย	1
วัตถุประสงค์ของการวิจัย	4
ขอบเขตของการวิจัย	4
ประโยชน์ที่ได้รับจากการวิจัย	5
บทที่ 2 วิธีดำเนินการวิจัย	
กรอบแนวคิดของงานวิจัย	6
การออกแบบอัลกอริทึมเพื่อสังเคราะห์ข้อมูล	14
การออกแบบอัลกอริทึมเพื่อคำนวณความหนาแน่นของข้อมูล	16
การออกแบบอัลกอริทึมเพื่อสุ่มข้อมูลตามความหนาแน่น	21
บทที่ 3 การทดสอบโปรแกรม	
วิธีการทดสอบโปรแกรมสุ่มข้อมูลและโปรแกรมจัดกลุ่มข้อมูล	26
ข้อมูลและเครื่องมือที่ใช้ในการทดสอบ	27
ผลการทดสอบการสุ่มข้อมูลเพื่อการจัดกลุ่มข้อมูล	30
ผลการเปรียบเทียบความคลาดเคลื่อนของการจัดกลุ่มข้อมูล	39
ผลการเปรียบเทียบเวลาที่ใช้ในการสุ่มและจัดกลุ่มข้อมูล	41
ผลการเปรียบเทียบหน่วยความจำที่ใช้เพื่อจัดเก็บข้อมูล	45
อภิปรายผล	49
บทที่ 4 บทสรุป	
สรุปผลการวิจัย	51
ข้อเสนอแนะ	54
บรรณานุกรม	55

หน้า

ภาคผนวก

ภาคผนวก ก รหัสต้นฉบับของโปรแกรมการสุ่มตามความหนาแน่นเพื่อการจัดกลุ่มข้อมูล 58

ภาคผนวก ข ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่ 67

ประวัติผู้วิจัย 89

สารบัญตาราง

หน้า

ตารางที่ 1.1 ข้อมูลลูกค้าบัตรเครดิตแสดงรายได้และการตอบสนองต่อรายการเสนอขายสินค้า 1
ตารางที่ 3.1 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวน 4 กลุ่ม ข้อมูลรวม 200 ตัว 31
ตารางที่ 3.2 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวน 8 กลุ่ม ข้อมูลรวม 200 ตัว 32
ตารางที่ 3.3 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวน 4 กลุ่ม ข้อมูลรวม 200 ตัว 33
ตารางที่ 3.4 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวน 8 กลุ่ม ข้อมูลรวม 200 ตัว 34
ตารางที่ 3.5 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวน 4 กลุ่ม ข้อมูลรวม 5,000 ตัว 35
ตารางที่ 3.6 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวน 8 กลุ่ม ข้อมูลรวม 5,000 ตัว 36
ตารางที่ 3.7 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวน 4 กลุ่ม ข้อมูลรวม 5,000 ตัว 37
ตารางที่ 3.8 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวน 8 กลุ่ม ข้อมูลรวม 5,000 ตัว 38
ตารางที่ 3.9 การเปรียบเทียบค่าความคลาดเคลื่อนของกลุ่มข้อมูลเมื่อใช้เทคนิคการสุ่มแต่ละแบบ 40

สารบัญภาพ

หน้า

รูปที่ 1.1 ผลการจัดกลุ่มข้อมูลลูกค้าบัตรเครดิตเป็น 3 กลุ่มพร้อมทั้งแสดงลักษณะเด่นในแต่ละกลุ่ม 2
รูปที่ 1.2 ตัวอย่างข้อมูลที่จำนวนข้อมูลกระจายไม่สม่ำเสมอในระหว่างกลุ่ม 3
รูปที่ 2.1 ผลการจัดกลุ่มจากข้อมูลสุ่มแบบ random เปรียบเทียบกับการจัดกลุ่มกับข้อมูลที่สุ่มแบบให้ค่าน้ำหนักตามความหนาแน่น 7
รูปที่ 2.2 Reservoir-sampling algorithm 9
รูปที่ 2.3 แผนภาพแสดงการกำหนดข้อมูลเริ่มต้นในการสุ่มด้วยเทคนิค reservoir 9
รูปที่ 2.4 กรอบแนวคิดและโมดูลต่างๆของงานวิจัย 12
รูปที่ 2.5 อัลกอริทึมสังเคราะห์ข้อมูล 14
รูปที่ 2.6 ชุดคำสั่งภาษาเออแลง (Erlang) ที่ทำหน้าที่สังเคราะห์ข้อมูล 15
รูปที่ 2.7 ข้อมูลสองมิติที่กำหนดให้สังเคราะห์ขึ้น 4 กลุ่มแต่ละกลุ่มมีข้อมูล 3, 7, 5 และ 5 ตัวตามลำดับ 15
รูปที่ 2.8 ข้อมูลสามมิติที่สร้างขึ้นด้วยข้อกำหนดเดียวกับในรูปที่ 2.7 16
รูปที่ 2.9 อัลกอริทึมการเลื่อนกรอบหน้าต่างเพื่อคำนวณความหนาแน่นของข้อมูล 17
รูปที่ 2.10 ชุดคำสั่งภาษาเออแลงที่ทำหน้าที่เลื่อนกรอบหน้าต่างและนับจำนวนข้อมูล 17
รูปที่ 2.11 ชุดข้อมูลสังเคราะห์และแผนภาพในลักษณะกราฟสองมิติ 18
รูปที่ 2.12 การกำหนดกรอบหน้าต่างและขอบเขตการนับจำนวนข้อมูลในแต่ละกรอบหน้าต่าง 19
รูปที่ 2.13 เปรียบเทียบผลการจัดกลุ่มกับชุดข้อมูลดั้งเดิมและผลการจัดกลุ่มกับชุดข้อมูลที่ถูกแปลงกลับจากอัลกอริทึม Window sliding 20
รูปที่ 2.14 อัลกอริทึมการสุ่มข้อมูลตามความหนาแน่น 22
รูปที่ 2.15 ชุดคำสั่งภาษาเออแลงที่ทำหน้าที่สุ่มข้อมูลตามความหนาแน่น 23
รูปที่ 2.16 แผนภาพเปรียบเทียบวิธีการสุ่มข้อมูลตามความหนาแน่นด้วยเทคนิคที่ต่างกันสี่แบบ 24
รูปที่ 2.17 โปรแกรม k-means clustering ในรูปแบบของภาษาเออแลง 25
รูปที่ 3.1 คำสั่งในภาษาเออแลงที่ใช้วัดเวลาการประมวลผล 27
รูปที่ 3.2 คำสั่งในภาษาเออแลงที่ใช้วัดเนื้อที่หน่วยความจำ 27
รูปที่ 3.3 ลักษณะและจำนวนจุดข้อมูลในข้อมูลสังเคราะห์ 8 ชุดที่ใช้ในการทดสอบโปรแกรมสุ่มและจัดกลุ่มข้อมูลตามความหนาแน่น 28
รูปที่ 3.4 ตัวอย่างการใช้โปรแกรม GNUPLOT เพื่อแสดงภาพข้อมูลในลักษณะกราฟ 3 มิติ ... 28

หน้า

รูปที่ 3.5 ข้อมูลสังเคราะห์ทั้ง 8 ชุดที่มีการกระจายข้อมูลแบบสม่ำเสมอ (ภาพด้านซ้าย) และกระจายแบบซิฟ (ภาพด้านขวา) 29
รูปที่ 3.6 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 4 กลุ่ม 41
รูปที่ 3.7 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 8 กลุ่ม 42
รูปที่ 3.8 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 4 กลุ่ม 43
รูปที่ 3.9 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 8 กลุ่ม 44
รูปที่ 3.10 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 4 กลุ่ม 45
รูปที่ 3.11 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 8 กลุ่ม 46
รูปที่ 3.12 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 4 กลุ่ม 47
รูปที่ 3.13 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 8 กลุ่ม 48

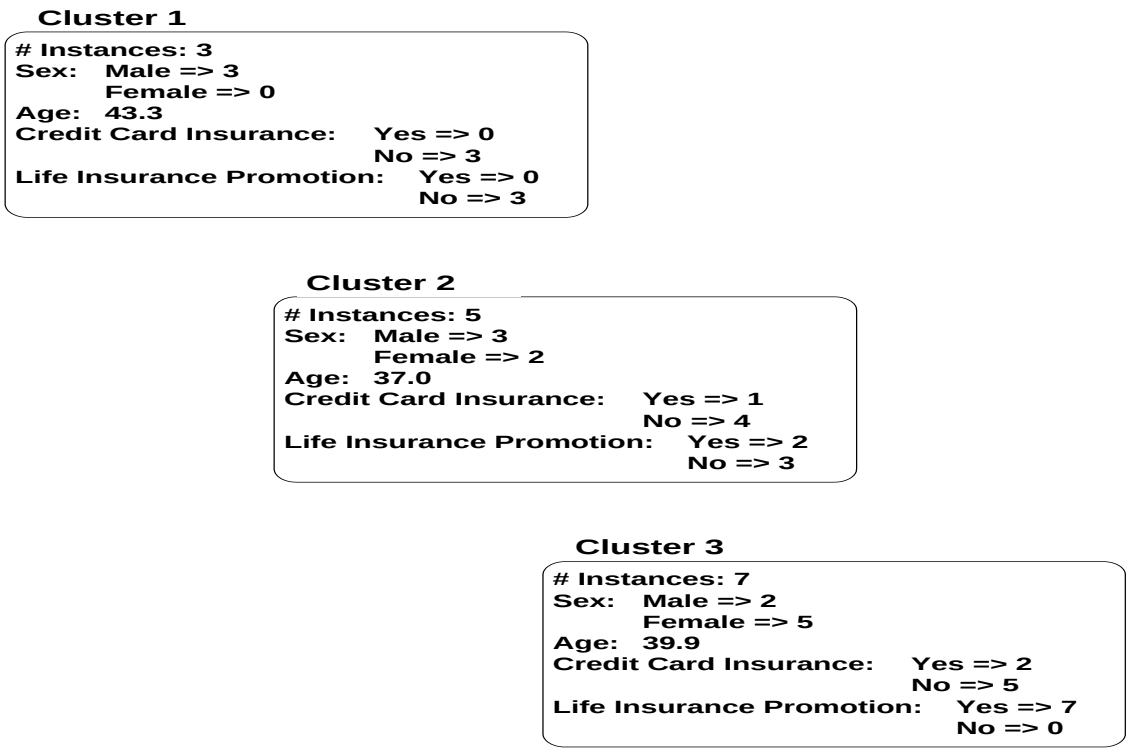
บทที่ 1
บทนำ

ความสำคัญและที่มาของปัญหาการวิจัย

การจัดกลุ่มข้อมูล (data clustering) เป็นการวิเคราะห์ข้อมูลเพื่อรวมข้อมูลที่มีลักษณะหรือคุณสมบัติที่คล้ายกันให้อยู่กลุ่มเดียวกัน เพื่อประโยชน์ในการสรุปข้อมูลและพิจารณาลักษณะของแต่ละกลุ่มข้อมูล ตัวอย่างเช่น จากข้อมูลลูกค้าบัตรเครดิตตามตารางที่ 1.1 เมื่อประมวลผลด้วยโปรแกรมจัดกลุ่มข้อมูล iDA [Roiger and Geatz, 2003] จะได้ผลสรุปดังรูปที่ 1.1 ที่แสดงผลการจัดข้อมูลเป็นสามกลุ่ม กลุ่มที่หนึ่งมีจำนวนลูกค้า 3 ราย เป็นเพศชายทั้งหมด อายุโดยเฉลี่ย 43.3 ปี ไม่สนใจทำทั้งประกันชีวิตและประกันบัตรเครดิต กลุ่มที่สองมีจำนวนลูกค้า 5 ราย เป็นเพศหญิงสามรายและเพศชายสองราย อายุโดยเฉลี่ย 37 ปี สนใจทำประกันบัตรเครดิตหนึ่งรายและทำประกันชีวิตสองราย กลุ่มที่สามมีจำนวนลูกค้า 7 ราย อายุโดยเฉลี่ย 39.9 ปี เป็นเพศหญิงห้ารายและเพศชายสองราย สนใจทำประกันบัตรเครดิตสองรายและทั้งเจ็ดรายทำประกันชีวิต

ตารางที่ 1.1 ข้อมูลลูกค้าบัตรเครดิตแสดงรายได้และการตอบสนองต่อรายการเสนอขายสินค้า [Roiger and Geatz, 2003]

Income Range	Magazine Promotion	Watch Promotion	Life Insurance Promotion	Credit Card Insurance	Sex	Age
40-50K	Yes	No	No	No	Male	45
30-40K	Yes	Yes	Yes	No	Female	40
40-50K	No	No	No	No	Male	42
30-40K	Yes	Yes	Yes	Yes	Male	43
50-60K	Yes	No	Yes	No	Female	38
20-30K	No	No	No	No	Female	55
30-40K	Yes	No	Yes	Yes	Male	35
20-30K	No	Yes	No	No	Male	27
30-40K	Yes	No	No	No	Male	43
30-40K	Yes	Yes	Yes	No	Female	41
40-50K	No	Yes	Yes	No	Female	43
20-30K	No	Yes	Yes	No	Male	29
50-60K	Yes	Yes	Yes	No	Female	39
40-50K	No	Yes	No	No	Male	55
20-30K	No	No	Yes	Yes	Female	19



รูปที่ 1.1 ผลการจัดกลุ่มข้อมูลลูกค้าบัตรเครดิตเป็น 3 กลุ่มพร้อมทั้งแสดงลักษณะเด่นในแต่ละกลุ่ม [Roiger and Geatz, 2003]

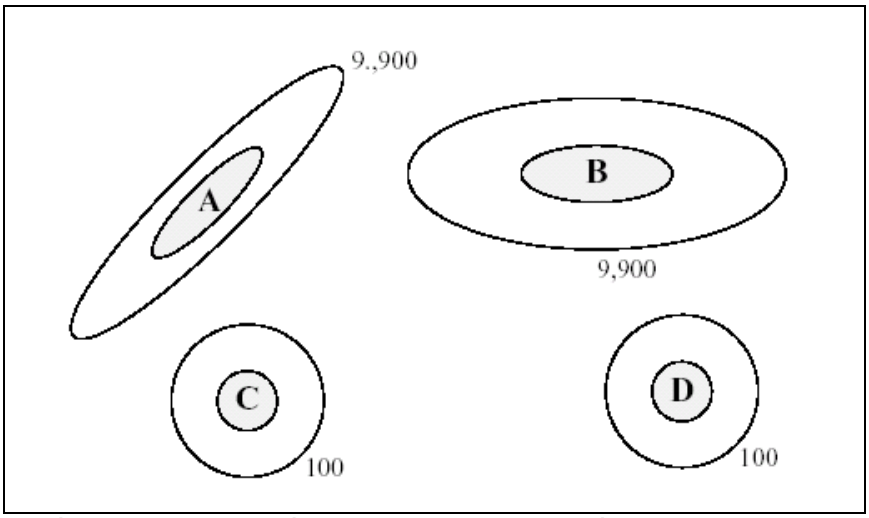
การจัดกลุ่มข้อมูลเป็นงานที่มีความสำคัญและนำไปใช้ประโยชน์ในหลายด้าน เช่น การจัดเอกสารอิเล็กทรอนิกส์ที่เนื้อหาเกี่ยวข้องกันให้อยู่ในหมวดเดียวกัน การจัดสารประกอบโปรตีนที่มีโครงสร้างโมเลกุลคล้ายกันให้อยู่ประเภทเดียวกัน การจัดกลุ่มลูกค้าที่มีพฤติกรรมการซื้อสินค้าคล้ายคลึงกัน เป็นต้น

เทคนิคการจัดกลุ่มข้อมูลได้รับการพัฒนามาอย่างต่อเนื่องโดยนักวิจัยในหลายสาขา เช่น สาขาสถิติ, สาขาการรู้จำแบบ (pattern recognition), สาขาการทำเหมืองข้อมูล หลักการเบื้องต้นของเทคนิคการจัดกลุ่มข้อมูล ประกอบด้วย การกำหนดจำนวนกลุ่ม, การหาจุดศูนย์กลางของแต่ละกลุ่มข้อมูล, การกำหนดกลุ่มให้กับข้อมูลที่อยู่โดยรอบโดยใช้เกณฑ์ต่างๆ เช่น การวัดระยะห่างแบบยูคลิเดียน (Euclidean distance) การวัดค่าความคล้ายของข้อมูล

เนื่องจากการกำหนดกลุ่มให้กับข้อมูล จะต้องวัดระยะห่างของข้อมูลกับจุดศูนย์กลางของแต่ละกลุ่มเพื่อพิจารณาว่าข้อมูลนั้นอยู่ใกล้กับกลุ่มใด จากนั้นจัดข้อมูลให้เข้ากลุ่มที่อยู่ใกล้ที่สุด ในกรณีที่ข้อมูลมีจำนวนมากการคำนวณระยะห่างของข้อมูลทุกตัวจะเป็นงานที่ใช้เวลามาก จึงมีความพยายามที่จะลดเวลาในส่วนนี้ โดยการใช้ข้อมูลที่เป็นตัวแทน (representative samples) แทนที่จะใช้ข้อมูลทั้งหมด เทคนิคการสุ่ม (sampling) จึงถูกนำมาใช้อย่างแพร่หลายในงานจัดกลุ่มข้อมูล

เทคนิคการสุ่มที่นิยมใช้กันโดยทั่วไปจะเป็นการสุ่มที่มีการกระจายการสุ่มอย่างสม่ำเสมอ (uniform random sampling) นั่นคือ ข้อมูลตั้งต้นทุกตัวมีโอกาสเท่าเทียมกันที่จะถูกเลือกเป็นตัวแทนไปใช้ในกระบวนการจัดกลุ่ม การสุ่มข้อมูลแบบนี้ใช้ได้ดีกับข้อมูลที่มีการกระจายสม่ำเสมอ แต่ในกรณีที่ข้อมูลมีการกระจายในรูปแบบอื่น เช่น แบบ Gaussian ที่มีหลายโมเดลผสมกัน (mixture models) การสุ่มแบบกระจายสม่ำเสมอจะใช้ไม่ได้ผล ดังตัวอย่างในรูปที่ 2 ที่ข้อมูลตั้งต้นมี 4 กลุ่ม กลุ่ม A มีจำนวนข้อมูล 9,900 ตัว กลุ่ม B มีจำนวน 9,900 ตัว กลุ่ม C มีจำนวนข้อมูล 100 ตัว และกลุ่ม D มีข้อมูล 100 ตัว บริเวณแกนกลางของแต่ละกลุ่มคือบริเวณที่มีข้อมูลอยู่หนาแน่น

จากจำนวนข้อมูลรวมทั้งหมด 20,000 ตัว ถ้าต้องการสุ่มเลือกข้อมูลจากทุกกลุ่มมาเพียง 1% ซึ่งคิดเป็นข้อมูลที่ถูกสุ่มเลือกรวม 200 ตัว ข้อมูลจากกลุ่ม A และ B จะถูกเลือกมากลุ่มละ 99 ตัว ในขณะที่กลุ่ม C และ D จะถูกเลือกมาเพียงกลุ่มละ 1 ตัว ปัญหาที่เกิดขึ้นคือ ข้อมูลหนึ่งตัวนี้จะถูกพิจารณาว่าเป็นข้อมูลที่อาจเป็นความผิดพลาดจากการสุ่ม หรือข้อมูลรบกวน (noise) และโปรแกรมจัดกลุ่มข้อมูลจะตัดข้อมูลนี้ออกจากการพิจารณา ทำให้ผลลัพธ์สุดท้ายที่ได้จากการจัดกลุ่มกับข้อมูลสุ่มไม่ถูกต้องตามความเป็นจริง



รูปที่ 1.2 ตัวอย่างข้อมูลที่จำนวนข้อมูลกระจายไม่สม่ำเสมอในระหว่างกลุ่ม

เพื่อแก้ปัญหาในกรณีข้อมูลแต่ละกลุ่มมีขนาดแตกต่างกัน มีความหนาแน่นของข้อมูลไม่เท่ากัน และการกระจายของข้อมูลมีหลายรูปแบบ (กระจายแบบสม่ำเสมอ, แบบ Gaussian, แบบ Zipf และอื่นๆ) การสุ่มเลือกข้อมูลจึงจำเป็นต้องให้ค่าน้ำหนักแก่ข้อมูลแต่ละตัวไม่เท่ากัน เพื่อเพิ่มโอกาสการถูกเลือกให้กับข้อมูลในบางกลุ่ม

โครงการวิจัยนี้เน้นการศึกษาและพัฒนาวิธีการสุ่มแบบลำดับ (sequential random sampling) โดยมีข้อกำหนดว่า กระบวนการสุ่มจะต้องกราดตรวจ (scan) ข้อมูลทั้งหมดเพียงรอบ

เดียว เพื่อให้ขั้นตอนการสุ่มใช้เวลาน้อยที่สุด เพราะในการทำงานกับข้อมูลจริง เช่น ข้อมูลจากการตรวจสภาพอากาศ ข้อมูลสำมะโนประชากร หรือข้อมูลลูกค้าโทรศัพท์มือถือ ปริมาณข้อมูลจะมีจำนวนมหาศาล ขั้นตอนการสุ่มเลือกข้อมูลตัวแทนเพื่อนำไปใช้ในการจัดกลุ่มจะต้องรวดเร็วและถูกต้อง

วัตถุประสงค์ของการวิจัย

1) พัฒนาเทคนิคการสุ่มข้อมูลเพื่อใช้ในงานจัดกลุ่มข้อมูลอัตโนมัติ โดยข้อมูลที่เป็นกลุ่มประชากรอาจจะมีขนาดแตกต่างกัน และมีความหนาแน่นแตกต่างกันในลักษณะที่เป็นโมเดลผสม (mixture model)
2) สร้างโปรแกรมที่สามารถรับข้อมูลเข้าขนาดใหญ่ แล้วให้ผลลัพธ์ของการสุ่มเป็นชุดข้อมูลตัวอย่าง (sample) ที่มีขนาดเล็กลง แต่สัดส่วนของข้อมูลยังคงเดิม
3) ทดสอบเทคนิคการสุ่มข้อมูลกับข้อมูลหลายลักษณะ ด้วยโปรแกรมจัดกลุ่มข้อมูลที่นิยมใช้ในปัจจุบัน เช่น k-means

ขอบเขตของการวิจัย

โครงการวิจัยนี้จะเน้นที่การพัฒนาเทคนิคการสุ่มข้อมูล โดยข้อมูลสามารถมีการกระจายได้หลายรูปแบบ เช่น แบบสม่ำเสมอ, แบบ Zipf เพื่อให้ได้ข้อมูลที่มีการกระจายหลายรูปแบบ จึงใช้ข้อมูลที่สังเคราะห์ขึ้นให้มีขนาดและความหนาแน่นที่แตกต่างกัน ข้อมูลที่สังเคราะห์ขึ้นมีตั้งแต่หนึ่งมิติไปจนถึงสิบมิติ โดยเป็นข้อมูลชนิดเลขจำนวนเต็ม (แต่ค่าจุดกึ่งกลางกลุ่มที่คำนวณได้จะเป็นเลขจำนวนจริง)

การทดสอบผลด้านความถูกต้องของการสุ่มข้อมูล จะใช้วิธีให้โปรแกรมจัดกลุ่มข้อมูลประมวลผลหากลุ่มข้อมูลและจุดกึ่งกลางกลุ่มจากข้อมูลเริ่มต้น เปรียบเทียบกับค่าจุดกึ่งกลางกลุ่มที่ได้จากการจัดกลุ่มข้อมูลที่ผ่านกระบวนการสุ่มเพื่อดูความคลาดเคลื่อนของกลุ่ม ในด้านการทดสอบประสิทธิภาพของโปรแกรมจะใช้การวัดเวลาการประมวลผลของโปรแกรม ซึ่งจะรวมทั้งเวลาที่ใช้ในการสุ่มและเวลาที่ใช้จัดกลุ่มข้อมูล นอกจากนี้ยังมีการวัดประสิทธิภาพด้านการใช้เนื้อที่หน่วยความจำที่ใช้เก็บข้อมูลในช่วงของการสุ่มและการจัดกลุ่มข้อมูล

การพัฒนาโปรแกรมใช้ภาษาเชิงฟังก์ชัน เนื่องจากมีวิธีการเขียนเชิงประกาศ (declarative programming) ซึ่งจะช่วยให้การพัฒนาโปรแกรมต้นแบบทำได้รวดเร็วและรูปแบบคำสั่งจะคล้ายกับซูโดโค้ดทำให้ง่ายต่อการอ่านโปรแกรม นอกจากนี้ภาษาเชิงประกาศยังมีแนวคิดและรูปแบบที่เหมาะสมสำหรับงานที่จะพัฒนาเป็นฟังก์ชันอันดับสูง และปรับปรุงเป็นการประมวลผลฟังก์ชัน

แบบคู่ขนานต่อไปได้โดยง่าย ในการพัฒนาโปรแกรมต้นแบบของงานวิจัยนี้ใช้ภาษาเออแลง (Erlang) เป็นภาษาในการเขียนรหัสต้นฉบับ ภาษาเออแลงเป็นภาษาเชิงฟังก์ชันพัฒนาขึ้นโดยทีมวิจัยของบริษัทอีริคสัน (Ericsson) ด้วยวัตถุประสงค์ที่จะสร้างภาษาที่มีขีดความสามารถสูงในงานโปรแกรมแบบพร้อมกัน (concurrent programming) ที่เป็นการประมวลผลด้วยสัญลักษณ์ (symbolic computation) เช่น การทำ pattern matching ภาษาเออแลงประสบความสำเร็จและนิยมใช้มากในการพัฒนาโปรแกรมกับงานด้านการสื่อสารโทรคมนาคม

การประมวลโปรแกรมภาษาเออแลงในงานวิจัยนี้ใช้คอมไพเลอร์ รีลีส R13B04 (เริ่มเผยแพร่เมื่อ February 24, 2010) ซึ่งผู้ใช้ทั่วไปสามารถดาวน์โหลดได้จาก www.erlang.org และเนื่องจากซอฟต์แวร์นี้เป็นโอเพนซอร์ส (open-source software) ทำให้ผู้ใช้สามารถนำไปใช้งานหรือนำรหัสต้นฉบับไปพัฒนาต่อได้โดยไม่มีปัญหาเรื่องลิขสิทธิ์ซอฟต์แวร์

ประโยชน์ที่ได้รับจากการวิจัย

โครงการวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาองค์ความรู้ใหม่ในด้านการพัฒนาเทคนิคการลดขนาดข้อมูล (data reduction) ที่จะเอื้อประโยชน์ต่องานวิเคราะห์ข้อมูลอัตโนมัติด้านการจัดกลุ่มข้อมูล ดังนั้นผลสำเร็จของโครงการวิจัยนี้จะเป็นเทคนิคใหม่ในการสุ่มเลือกข้อมูล (data sampling) ไปใช้ในขั้นตอนการวิเคราะห์อัตโนมัติ โดยเฉพาะการวิเคราะห์เพื่อจัดกลุ่มข้อมูล ความรู้ที่ได้จึงใช้ประโยชน์ได้กับทุกหน่วยงานที่ต้องมีการวิเคราะห์ข้อมูลขนาดใหญ่เพื่อจัดกลุ่มข้อมูล และเรียนรู้ลักษณะเด่นในกลุ่มข้อมูลหรือลักษณะประชากรในแต่ละกลุ่ม

ผลการวิจัยในส่วนการพัฒนาเทคนิคการลดขนาดข้อมูลที่เป็นองค์ความรู้ใหม่ ได้มีการเผยแพร่ด้วยการเขียนบทความวิจัย นำเสนอและตีพิมพ์ในเอกสารการประชุมวิชาการระดับประเทศและระดับนานาชาติ

ผลงานในส่วนโปรแกรมการสุ่มข้อมูลแบบลำดับ โปรแกรมการสังเคราะห์ข้อมูลและโปรแกรมการจัดกลุ่มข้อมูล จะเผยแพร่ซอฟต์แวร์และคู่มือการใช้งานผ่านอินเทอร์เน็ตโดยให้นักวิจัยและผู้สนใจทั่วไป ดาวน์โหลดได้จากเว็บไซต์ของหน่วยปฏิบัติการวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้ มหาวิทยาลัยเทคโนโลยีสุรนารี

บทที่ 2
วิธีดำเนินการวิจัย

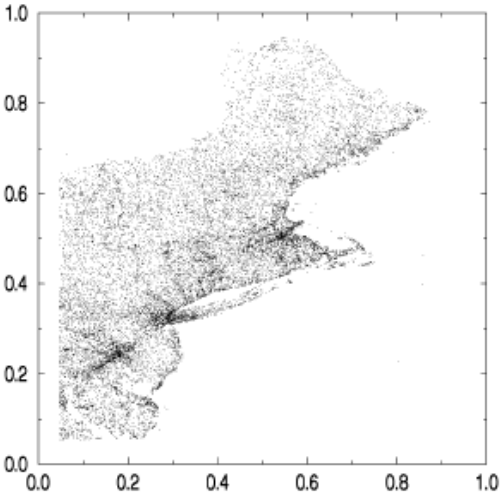
กรอบแนวคิดของงานวิจัย

การจัดกลุ่มข้อมูล (data clustering) เป็นงานพื้นฐานของการวิเคราะห์ข้อมูล [Anderberg, 1973] การวิเคราะห์ให้ได้ผลลัพธ์ที่ถูกต้องจะต้องพิจารณาการกระจาย (distribution) ของข้อมูล รูปแบบการกระจายของข้อมูลที่รู้จักกันมากที่สุดคือ การกระจายแบบสม่ำเสมอ (uniform distribution) แต่ข้อมูลที่เกิดขึ้นจริงในงานบางประเภทได้รับการพิสูจน์แล้วว่าการกระจายเป็นแบบซิฟ (Zipf distribution) ตัวอย่างเช่น การกระจายของประชากรในเขตตัวเมืองและเขตรอบนอกของเมืองที่ไม่สม่ำเสมอกัน [Zipf, 1949] การกระจายรายได้ของประชากร [Schroeder, 1991] การกระจายของยอดขายสินค้า [Faloutsos, Matias, and Silbershatz, 1996]

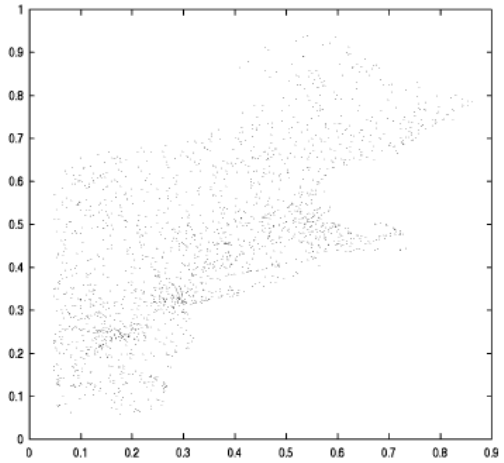
การลดขนาดของข้อมูลด้วยการสุ่มเลือกข้อมูลตัวแทน (sampling) เพื่อเพิ่มความเร็วในการจัดกลุ่มข้อมูล จึงจำเป็นต้องคำนึงถึงรูปแบบการกระจายของข้อมูลในกรณีที่ข้อมูลกระจายไม่สม่ำเสมอ การสุ่มจากบริเวณที่มีข้อมูลอยู่น้อยจำเป็นต้องเพิ่มความน่าจะเป็นที่ข้อมูลจะถูกเลือกเพื่อยังคงรักษาข้อมูลในกลุ่มเล็กไว้ แต่ในขณะเดียวกันข้อมูลในกลุ่มหรือบริเวณที่มีความหนาแน่นสูงควรเพิ่มอัตราการสุ่มเพื่อยังคงรักษาขอบเขตที่ถูกต้องของกลุ่ม ดังแสดงในผลการศึกษาของ Kollios และคณะ [Kollios *et al*., 2003] ที่แสดงการสุ่มประชากรในเขตเมืองใหญ่ 3 เมือง คือ นิวยอร์ค ฟิลลาเดลเฟีย และบอสตัน (รูปที่ 2.1)

ในรูปที่ 2.1(a) แสดงข้อมูลประชากรทั้งหมดในลักษณะของจุด รูปที่ 2.1(b) แสดงการสุ่มเลือกตัวแทนมา 1,000 จุดด้วยวิธี random sampling รูปที่ 2.1(c) เป็นการสุ่มเลือกตัวแทนจำนวนเท่ากันแต่สุ่มโดยให้ค่าน้ำหนักตามความหนาแน่น รูปที่ 2.1(d) แสดงผลลัพธ์ของการจัดกลุ่มข้อมูลโดยแสดงข้อมูล 10 ตัวในแต่ละกลุ่ม และเป็นการจัดกลุ่มกับข้อมูลที่ได้จากการทำ random sampling รูปที่ 2.1(e) เป็นการจัดกลุ่มกับข้อมูลในรูปที่ 2.1(c) จะเห็นว่ากลุ่มข้อมูลที่ได้จะใกล้เคียงกับตำแหน่งกลุ่มข้อมูลดั้งเดิมในเขตเมืองนิวยอร์ค ฟิลลาเดลเฟีย และบอสตัน

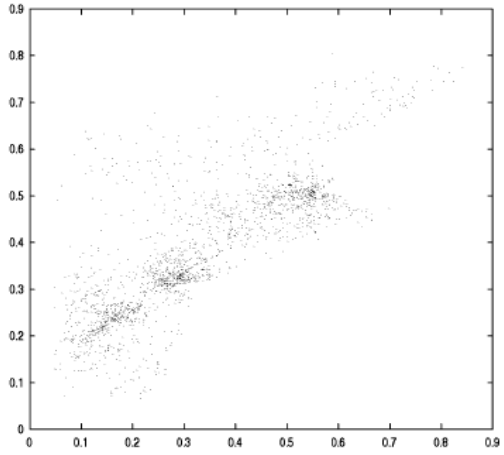
ผลการศึกษานี้ยืนยันข้อสมมติฐานที่ว่า การสุ่มข้อมูลแบบ random ที่เป็นการสุ่มแบบมีการกระจายความน่าจะเป็นที่จะถูกเลือกอย่างสม่ำเสมอให้กับข้อมูลทุกตัว ให้ผลลัพธ์ในการจัดกลุ่มข้อมูลได้ไม่ดีเท่าการสุ่มที่ปรับค่าน้ำหนักตามความหนาแน่น (density-biased sampling) ในกรณีที่ข้อมูลมีการกระจายของประชากรไม่สม่ำเสมอ



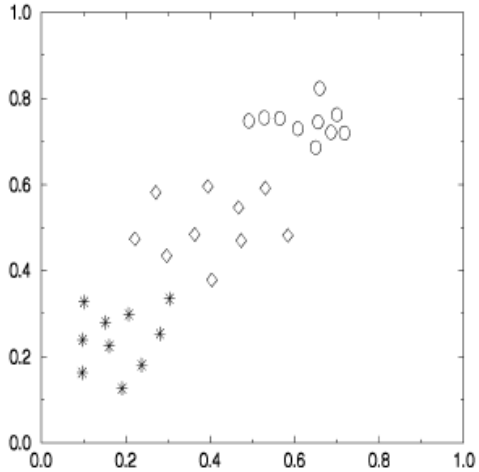
(a) ประชากรทั้งหมด



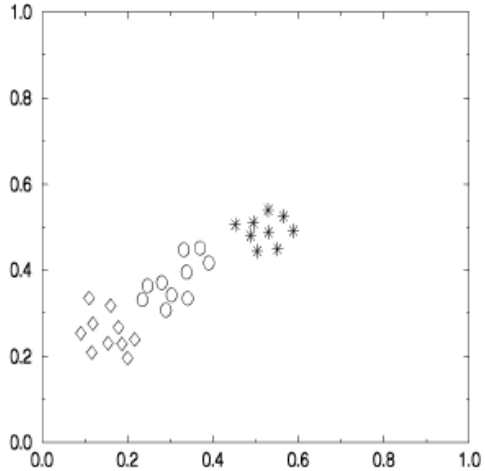
(b) ข้อมูลสุ่มแบบ random



(c) ข้อมูลสุ่มตามความหนาแน่น



(d) ผลการจัดกลุ่มกับข้อมูลสุ่มแบบ random



(e) ผลการจัดกลุ่มกับข้อมูลสุ่มตามความหนาแน่น

รูปที่ 2.1 ผลการจัดกลุ่มข้อมูลจากข้อมูลสุ่มแบบ random เปรียบเทียบกับการจัดกลุ่มกับข้อมูลที่สุ่มแบบให้ค่าน้ำหนักตามความหนาแน่น

เทคนิคการจัดกลุ่มข้อมูล เป็นการรวมกลุ่มข้อมูลโดยพิจารณาจากค่าที่ระบุลักษณะของข้อมูล (attribute values) กระบวนการทำงานของการจัดกลุ่มข้อมูลจึงเป็นการค้นหากลุ่มต่างๆ ของข้อมูลโดยข้อมูลภายในกลุ่มเดียวกันจะต้องมีลักษณะที่เหมือนกันมากที่สุด ในขณะที่ข้อมูลที่อยู่ต่างกลุ่มกันจะต้องมีลักษณะที่ต่างกันมากที่สุด

การวัดความเหมือนหรือความต่างนี้จะใช้เกณฑ์ที่แตกต่างกันไปในแต่ละอัลกอริทึม เช่น ใช้วัด distance ในอัลกอริทึม k-means [Jain and Dubes, 1988] หรืออาจจะใช้การวัดค่า cosine และค่าสัมประสิทธิ์ Jaccard [Stehl, Ghosh, and Mooney, 2000] การเลือกเกณฑ์วัดค่าความเหมือน (similarity) จะมีผลต่อรูปร่างของกลุ่มข้อมูลที่ค้นพบ เช่น ถ้าใช้เกณฑ์ระยะห่าง Euclidean จะได้กลุ่มข้อมูลที่มีรูปร่างเป็นทรงกลม

ในระยะเริ่มแรกของการพัฒนาเทคนิคการจัดกลุ่มข้อมูล กลุ่มข้อมูลที่ค้นพบมักจะมีลักษณะทรงกลม แต่ในระยะหลังได้มีการพัฒนาอัลกอริทึมที่สามารถค้นหากลุ่มข้อมูลที่มีรูปทรงแตกต่างไปจากทรงกลม เช่น อัลกอริทึม DBSCAN [Sander, Ester, Kriegel, and Xu, 1998], CURE [Guha, Rastogi, and Shim, 1998], Chameleon [Karypis, Ham, and Kumar, 1999]

ถึงแม้เทคนิคการค้นหากลุ่มข้อมูลจะพัฒนาขึ้นตามลำดับ แต่ปัญหาหลักของการจัดกลุ่มข้อมูลยังคงเป็นเรื่องของประสิทธิภาพการประมวลผล ทั้งนี้เนื่องจากการจัดกลุ่มข้อมูลเป็นงานเรียนรู้แบบไม่มีการชี้แนะ (unsupervised learning) ขั้นตอนการค้นหาคำตอบจึงใช้เวลามาก เทคนิคการลดเวลาที่ได้ผลดีที่สุด คือการลดขนาดข้อมูล โดยการเลือกใช้เฉพาะข้อมูลตัวแทน และวิธีการสุ่มเลือกข้อมูลตัวแทนที่ให้ผลดี คือ วิธี biased sampling

การสุ่มแบบ biased เป็นการสุ่มโดยกำหนดค่าความน่าจะเป็นที่จะถูกเลือกให้กับข้อมูลแต่ละตัวไม่เท่ากันขึ้นอยู่กับความหนาแน่นในกลุ่มข้อมูล รูปร่างของกลุ่มข้อมูล และรูปแบบการกระจายของข้อมูล วิธีการสุ่มแบบ uniform random sampling อาจพิจารณาได้ว่าเป็น biased sampling ที่กำหนดความน่าจะเป็นที่จะถูกเลือกให้กับข้อมูลทุกตัวเท่ากัน

Palmer และ Faloutsos [2000] ได้พัฒนาวิธีการสุ่มแบบ biased เพื่อใช้ในงานจัดกลุ่มข้อมูล โดยใช้สมมติฐานว่าข้อมูลมีการกระจายแบบ Zipf และเทคนิคการสุ่มใช้ตารางแฮชช่วยในการทำงาน ดังนั้นปัญหาหลักของเทคนิคนี้คือ การชนกันของข้อมูล (collision) ซึ่งจะทำให้ประสิทธิภาพของการสุ่มลดลง

โครงการวิจัยนี้จึงมีวัตถุประสงค์หลักที่จะพัฒนาเทคนิค biased sampling เพื่องานจัดกลุ่มข้อมูลเช่นเดียวกับแนวคิดของ Palmer และ Faloutsos แต่เพื่อหลีกเลี่ยงปัญหาการชนกันของข้อมูล แทนที่จะใช้เทคนิคแฮชชิง โครงการวิจัยนี้จะใช้วิธีการสุ่มแบบลำดับ โดยมีโครงสร้างข้อมูลที่เรียกว่า reservoir [Vitter, 1985] เป็นโครงสร้างพื้นฐานแทนตารางแฮช reservoir เป็นเนื้อที่หน่วยความจำชั่วคราวที่ใช้เก็บข้อมูลตัวอย่างระหว่างกระบวนการสุ่ม วิธีการ random sampling

ด้วย reservoir แสดงอัลกอริทึมได้ดังรูปที่ 2.2 และแสดงแนวคิดในการประยุกต์ใช้กับงานจัดกลุ่มข้อมูลได้ดังรูปที่ 2.3 แนวคิดหลักของโครงการวิจัยนี้จึงเป็นการปรับปรุง random reservoir-sampling ให้เป็น biased reservoir-sampling

---

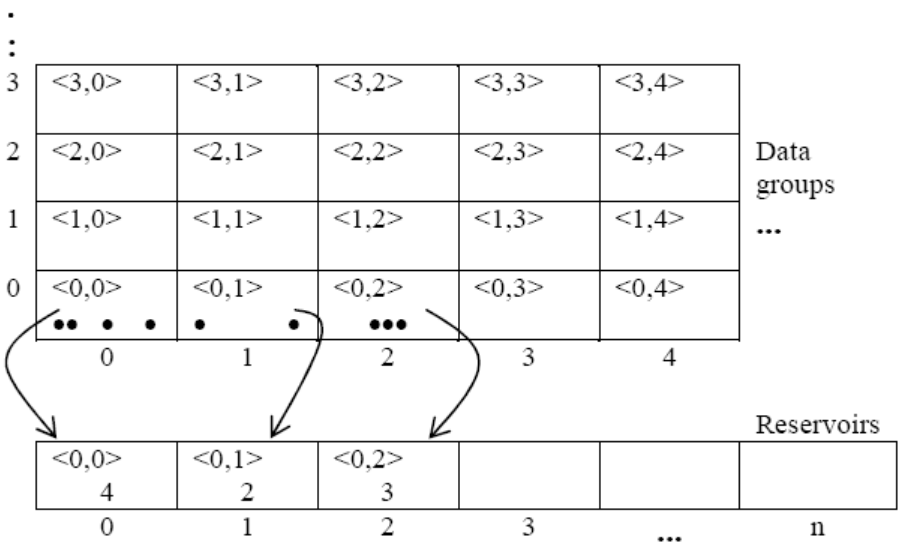
Algorithm 1 Reservoir sampling

Input: a data file of N population,
Output: a random sample of size n, n ≤ N

---

(1) Initialize the reservoir $X_1, ..., X_n$ to be the first n records of the file
(2) $W \leftarrow \exp(\log(\text{random}()) / n)$
// initialize W to be the largest value in a sample of
// size n from the uniform distribution on the interval (0, 1).
(3) While not eof do
(4) $S \leftarrow \lfloor \log(\text{random}()) / \log(1-W) \rfloor$
// generate the random variate S to denote
// the number of records to be skipped over before
// a new record can enter the reservoir
(5) If (not eof)
Then $Y \leftarrow (S+1)^{\text{th}}$ record
// search for the next potential record to be in the reservoir
Else return $X_1, ..., X_n$
(6) $X_{1+\lfloor n * \text{random}() \rfloor} \leftarrow Y$ // update X
(7) $W \leftarrow W * \exp(\log(\text{random}()) / n)$ // update W
(8) End While

รูปที่ 2.2 Reservoir-sampling algorithm [Vitter, 1985]



รูปที่ 2.3 แผนภาพแสดงการกำหนดข้อมูลเริ่มต้นในการสุ่มด้วยเทคนิค reservoir

วัตถุประสงค์หลักของโครงการวิจัยนี้ คือ การสุ่มข้อมูลแบบเอนเอียง (biased sampling) ตามขนาดและความหนาแน่นของข้อมูล เพื่อให้สามารถนำข้อมูลไปใช้ได้อย่างมีประสิทธิภาพในกระบวนการจัดกลุ่มข้อมูล (clustering) โดยมีขั้นตอนการวิจัยดังนี้

ขั้นตอนที่ 1 การรวบรวมเอกสารที่เกี่ยวข้อง

รวบรวมงานวิจัยที่เกี่ยวข้องกับ biased sampling เพื่อการวิเคราะห์กลุ่มข้อมูลที่มีขนาดต่างกัน และมีความหนาแน่นแตกต่างกัน รวมถึงการวิเคราะห์เทคนิคที่ใช้ในการประมาณการความหนาแน่นของข้อมูล (density approximation)

ขั้นตอนที่ 2 การพัฒนาวิธีการสุ่มแบบลำดับที่เอนเอียงตามความหนาแน่นของข้อมูล

การสุ่มแบบลำดับ (sequential random sampling) จะต้องได้รับการปรับปรุงรูปแบบและวิธีการทำงาน ให้สามารถเอนเอียงตามความหนาแน่นของกลุ่มข้อมูล (sequential biased sampling) และวิธีการสุ่มจะต้องอ่านข้อมูลเพียงรอบเดียว นั่นคือ การประมาณการความหนาแน่นของข้อมูล จะต้องกระทำไปพร้อมกับการสุ่มเลือกข้อมูล ทั้งนี้เพื่อเพิ่มความเร็วในการสุ่มเลือกข้อมูล ในงานวิจัยนี้จะใช้โครงสร้างของ reservoir ร่วมกับเทคนิคการสุ่มโดยจะพิจารณาทั้งเทคนิค simple random sampling และ rejection sampling เพื่อช่วยในการประมาณการความหนาแน่นของข้อมูล ดังนั้นเทคนิคการสุ่มที่พัฒนาขึ้น จึงเป็นวิธีการสุ่มแบบลำดับที่เอนเอียงตามขนาดและความหนาแน่นของข้อมูลโดยใช้โครงสร้างข้อมูล reservoir เป็นพื้นฐาน และเรียกเทคนิคนี้ว่า biased reservoir sampling

ขั้นตอนที่ 3 สร้างโปรแกรมการสุ่มแบบลำดับที่สามารถเอนเอียงตามขนาดและความหนาแน่นของข้อมูล

วิธีการสุ่มแบบลำดับที่ออกแบบไว้แล้วในขั้นตอนที่ 2 (biased reservoir sampling) จะถูกแปลงเป็นรหัสคำสั่งในรูปแบบการโปรแกรมเชิงฟังก์ชันด้วยภาษาเออแลง (Erlang) เพื่อการทดสอบประสิทธิภาพในการใช้งานจริง

ขั้นตอนที่ 4 สร้างโปรแกรมสังเคราะห์ข้อมูล

ข้อมูลที่จะใช้ในการทดสอบโปรแกรมการสุ่มแบบลำดับที่สามารถเอนเอียงตามขนาดและความหนาแน่นของข้อมูล จะต้องเป็นข้อมูลที่มีขนาด (size) ของแต่ละกลุ่มเล็ก-ใหญ่แตกต่างกัน และต้องเป็นข้อมูลที่มีความหนาแน่น (density) ในแต่ละกลุ่มแตกต่างกัน นอกจากนี้จำนวนกลุ่มภายในชุดข้อมูลจะต้องมีได้หลากหลาย เช่น ตั้งแต่ 2 กลุ่มไปจนถึง 10 กลุ่ม เป็นต้น มิติของข้อมูล (dimension) ก็จะต้องสามารถ

ควบคุมได้ เช่น ข้อมูลที่ไม่ซับซ้อนจะมีเพียง 1 หรือ 2 มิติ แต่ถ้าต้องการวิเคราะห์ข้อมูลที่ซับซ้อนขึ้น สามารถเพิ่มมิติสูงขึ้นได้

ดังนั้นเพื่อความสะดวกในการทดสอบความสามารถของโปรแกรมสุ่มข้อมูลแบบลำดับที่เอนเอียงตามความหนาแน่น รวมถึงการทดสอบการจัดกลุ่มข้อมูล จึงจำเป็นต้องสร้างโปรแกรมที่สามารถควบคุมการผลิตชุดข้อมูลที่มีจำนวน ลักษณะ และรูปแบบตรงตามต้องการได้ เรียกข้อมูลที่สร้างขึ้นนี้ว่าข้อมูลสังเคราะห์ โปรแกรมที่พัฒนาขึ้นสามารถสังเคราะห์ข้อมูลที่ความหนาแน่นระดับต่างๆ และที่มิติตั้งแต่ 1-10 มิติ (ถ้าต้องการเพิ่มมิติมากกว่านี้ สามารถปรับปรุงจากโปรแกรมต้นแบบนี้ได้)

ขั้นตอนที่ 5 สร้างโปรแกรมจัดกลุ่มข้อมูล (clustering)

โครงการวิจัยนี้เน้นการพัฒนาเทคนิคการสุ่มข้อมูลที่สามารถเอนเอียงตามขนาดและความหนาแน่น เพื่อประโยชน์ในการเพิ่มประสิทธิภาพการจัดกลุ่มข้อมูล ดังนั้นเพื่อความสมบูรณ์ของโครงการวิจัย จึงจำเป็นต้องนำข้อมูลที่สุ่มได้มาทดสอบจริงกับโปรแกรมจัดกลุ่มข้อมูล ซึ่งในโครงการนี้ใช้วิธีการของ k-means clustering [MacQueen, 1967] เป็นเทคนิคหลักในการพัฒนาโปรแกรม

ขั้นตอนที่ 6 ทดสอบโปรแกรมการสุ่มข้อมูลแบบเอนเอียงตามขนาดและความหนาแน่น

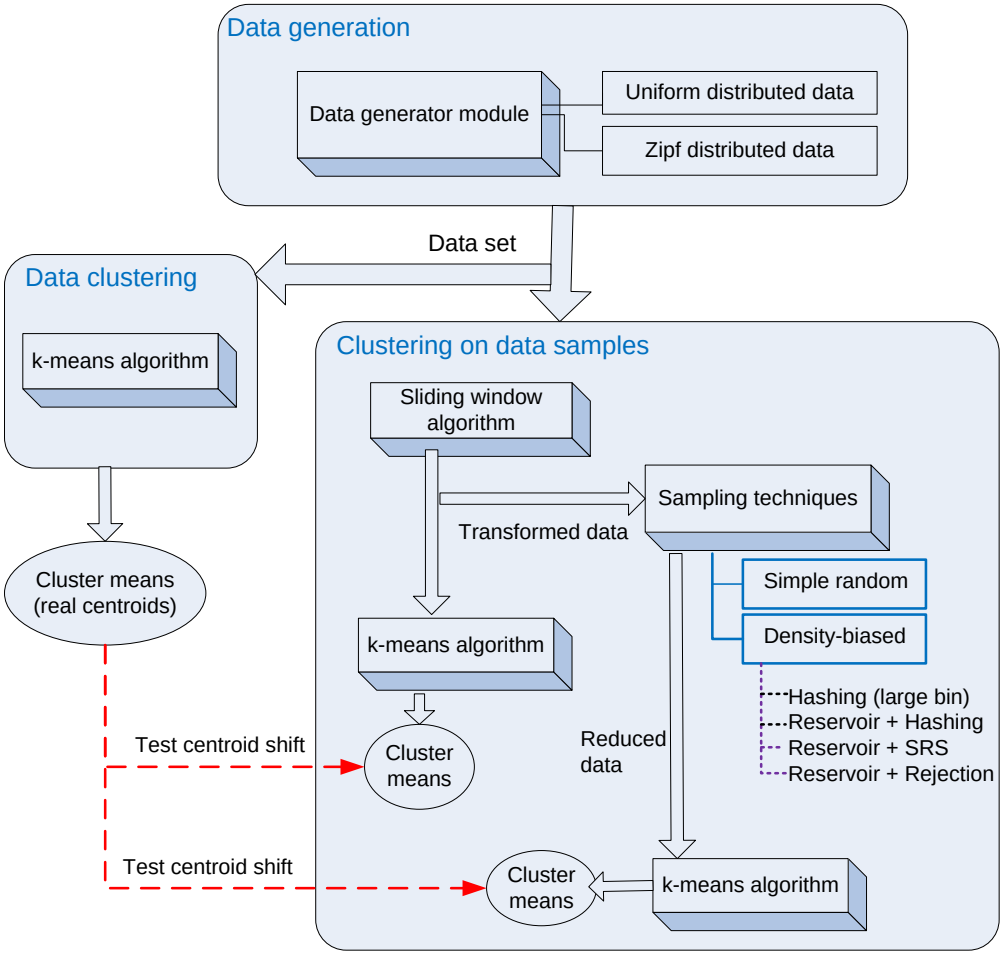
การทดสอบโปรแกรมการสุ่มข้อมูลแบบเอนเอียง (biased sampling) จะเปรียบเทียบกับวิธีการสุ่มของ Palmer และ Faloutsos [2000] ที่ใช้ตารางแฮชเป็นเครื่องมือในการประมาณการความหนาแน่นของข้อมูล ในการทดสอบเปรียบเทียบประสิทธิภาพการสุ่มข้อมูลแบบเอนเอียงตามความหนาแน่น จึงเป็นการเปรียบเทียบระหว่างการทำแฮชชิงด้วยตารางแฮชซึ่งสามารถเข้าถึงข้อมูลได้โดยตรง และการใช้โครงสร้าง reservoir ที่จัดเป็นการสุ่มแบบลำดับ การเปรียบเทียบใช้การพิจารณาประสิทธิภาพจากผลลัพธ์ที่ได้จากการประมวลผลด้วยโปรแกรมจัดกลุ่มข้อมูล

ขั้นตอนที่ 7 วิเคราะห์ผลการทดสอบและจัดทำรายงาน

วิเคราะห์เปรียบเทียบผลลัพธ์จากขั้นตอนที่ 6 พร้อมทั้งเสนอแนะแนวทางการวิจัยต่อยอด และสรุปผลงานทั้งหมดจัดทำเป็นรายงานการวิจัยฉบับสมบูรณ์

กรอบแนวคิดและโมดูลต่างๆในแต่ละขั้นตอนของการดำเนินงานโครงการวิจัยนี้ สรุปเป็นแผนภาพได้ดังรูปที่ 2.4 ขั้นตอนเริ่มต้นตามแผนภาพเป็นโมดูล Data generation ทำหน้าที่สร้างข้อมูลสังเคราะห์ โดยสามารถกำหนดจำนวนกลุ่มข้อมูลและขนาดของข้อมูลในแต่ละกลุ่ม ถ้าข้อมูล

มีลักษณะการกระจายอย่างสม่ำเสมอ (uniform distribution) ขนาดของข้อมูลในแต่ละกลุ่มจะใกล้เคียงกัน แต่ถ้าข้อมูลมีลักษณะการกระจายแบบซิฟ (Zipf distribution) ขนาดของข้อมูลในแต่ละกลุ่มจะเล็กใหญ่ต่างกันมาก



รูปที่ 2.4 กรอบแนวคิดและโมดูลต่างๆของงานวิจัย

ข้อมูลสังเคราะห์ที่สร้างจากโมดูล Data generation จะถูกนำไปใช้ในโมดูล Data clustering เพื่อให้โปรแกรม k-means ทำหน้าที่จัดกลุ่มข้อมูลและรายงานผลลัพธ์ของการจัดกลุ่มเป็นค่าจุดกึ่งกลาง (cluster means, or centroids) ของแต่ละกลุ่ม ค่าจุดกึ่งกลางนี้จะใช้เป็นค่าจุดกึ่งกลางที่แท้จริง (real centroids) ของกลุ่มข้อมูล เพื่อเป็นเกณฑ์ในการเปรียบเทียบผลการจัดกลุ่มกับข้อมูลที่ได้จากการสุ่มแบบต่างๆ ทั้งนี้เนื่องจากข้อมูลที่ใช้ในโมดูล Data clustering เป็นข้อมูลทั้งหมดที่สังเคราะห์จากโมดูล Data generation และผลการทดสอบเบื้องต้นกับโปรแกรม k-means พบว่าเมื่อกำหนดจำนวนกลุ่ม (ค่า k) ตรงกับจำนวนกลุ่มของข้อมูลสังเคราะห์ โปรแกรม k-means สามารถจัดกลุ่มและคำนวณจุดกึ่งกลางกลุ่มได้ถูกต้อง

ข้อมูลสังเคราะห์นอกจากใช้ในการคำนวณหา real centroids แล้ว ยังถูกนำไปใช้ในกระบวนการสุ่มข้อมูลตามความหนาแน่น (แผนภาพส่วน Clustering on data samples ของรูปที่ 2.4) โดยข้อมูลสังเคราะห์จะถูกส่งไปยังโมดูล Window sliding algorithm เพื่อแปลงข้อมูลและวัดความหนาแน่นของข้อมูลในแต่ละ window ข้อมูลที่ถูกแปลงแล้วแต่ยังไม่มีการสุ่มเพื่อลดขนาดข้อมูล จะถูกส่งไปยังโปรแกรม k-means เพื่อทดสอบผลการจัดกลุ่ม ในขณะเดียวกันข้อมูลที่ถูกแปลงและวัดความหนาแน่นแล้วจะถูกส่งต่อไปยังโมดูล Sampling techniques เพื่อทดสอบวิธีการสุ่มในแบบต่างๆ ซึ่งประกอบด้วย

- Simple random sampling เป็นวิธีการสุ่มเลือกข้อมูลอย่างง่าย โดยไม่มีการพิจารณาความหนาแน่นของข้อมูล
- Density-biased sampling เป็นวิธีการสุ่มเลือกข้อมูล โดยคัดเลือกจากข้อมูลที่มีความหนาแน่นถึงเกณฑ์ที่กำหนด โดยมีการใช้โครงสร้างข้อมูลประกอบกับเทคนิคการคัดเลือกข้อมูลดังนี้
  - Hashing ใช้เนื้อที่เก็บข้อมูลขนาดใหญ่ (ด้วยสมมติฐานว่าข้อมูลที่ความหนาแน่นถึงเกณฑ์ทั้งหมด สามารถเก็บลงตารางแฮชได้) และมีวิธีการเข้าถึงข้อมูลโดยตรง
  - Reservoir + Hashing ใช้เนื้อที่เก็บข้อมูลที่มีขนาดจำกัด (ด้วยโครงสร้างข้อมูลแบบแถวลำดับที่เรียกว่า reservoir) และมีวิธีการเข้าถึงข้อมูลได้โดยตรง (วิธีนี้เป็นเทคนิคแบบเดียวกับที่ Palmer และ Faloutsos [2000] ใช้ในงานวิจัย)
  - Reservoir + Simple random sampling (SRS) ใช้เนื้อที่เก็บข้อมูลที่มีขนาดจำกัด และใช้วิธีการคัดเลือกข้อมูลเพื่อเก็บลง reservoir แบบการสุ่มอย่างง่าย
  - Reservoir + Rejection sampling ใช้เนื้อที่เก็บข้อมูลที่มีขนาดจำกัด และใช้การสุ่มสองครั้ง (rejection sampling) เพื่อพิจารณาว่าจะทิ้งข้อมูล หรือจะเก็บข้อมูลลงเนื้อที่ reservoir

ข้อมูลที่ได้รับการสุ่มเลือกด้วยเทคนิคต่างๆกัน จะถูกนำไปจัดกลุ่มด้วยโปรแกรม k-means เพื่อพิจารณาค่า cluster means ที่เป็นผลลัพธ์ของการจัดกลุ่ม เปรียบเทียบกับค่า real centroids เทคนิคการสุ่มเลือกที่ดีควรจะต้องให้ผลลัพธ์การจัดกลุ่มที่เบี่ยงเบนไปจาก real centroids น้อยที่สุด

การออกแบบอัลกอริทึมเพื่อสังเคราะห์ข้อมูล

ข้อมูลสังเคราะห์ที่ใช้ในงานวิจัยนี้เป็นลักษณะจุดข้อมูล (data point) ข้อมูลแต่ละตัวระบุด้วยพิกัดบนแกนมิติ เช่นถ้าเป็นข้อมูลสองมิติ จุดข้อมูล [2,7] จะหมายถึงข้อมูลหนึ่งตัว ปรากฏอยู่ที่ตำแหน่งพิกัดแกน x=2 และแกน y=7 หรือถ้าข้อมูลเป็นสามมิติ จุดข้อมูล [11,2,6] จะหมายถึงข้อมูลหนึ่งตัว ปรากฏที่ตำแหน่งแกน x=11 แกน y=2 และแกน z=6 เป็นต้น ในงานวิจัยนี้จะกำหนดตำแหน่งข้อมูลบนแต่ละแกนเป็นเลขจำนวนเต็ม และแกนมิติมีได้ตั้งแต่หนึ่งมิติไปจนถึงสิบมิติ (จำนวนมิติสามารถสร้างให้สูงกว่าสิบมิติได้ แต่เนื่องจากข้อจำกัดทางด้านหน่วยความจำหลักของเครื่องคอมพิวเตอร์ส่วนบุคคลที่ใช้ในการทดลอง พบว่าถ้ากำหนดข้อมูลให้มีจำนวนมิติสูงมาก ขนาดของหน่วยความจำฮีพจะไม่เพียงพอต่อการประมวลผล)

วิธีการสังเคราะห์ข้อมูลจะให้ผู้ใช้กำหนดจำนวนมิติของข้อมูล จากนั้นกำหนดจำนวนกลุ่มที่ต้องการ และภายในแต่ละกลุ่มสามารถกำหนดจำนวนข้อมูลภายในกลุ่ม พร้อมทั้งค่าขอบเขตล่างและขอบเขตบนเพื่อกำหนดช่วงของการสุ่มค่าเพื่อสร้างข้อมูลสังเคราะห์ ขั้นตอนการทำงานของอัลกอริทึมสังเคราะห์ข้อมูลแสดงได้ดังรูปที่ 2.5

Algorithm Data-point generation

Input: number of dimensions (D),
number of data clusters (K),
number of data points in each cluster (N1, …, Nk),
range of lower (L) and upper (U) bounds on each dimension
Output: a set of all synthesized data points

---

(1) Start the GUI and read the values of <D, K, Ni, L, H>, i ∈ {1, .., K}, arguments
(2) For each K cluster
(3) For each dimension Di, i ∈ {1, .., D-1} % no range restriction on the last dimension
% to avoid linearly generated data points
(4) Generate random number within the range (H-L)
(5) Freely generate random number of the last dimension
(6) Accumulate the random number of each dimension in a list of data points
(7) If number of data points less than Nk, then repeat steps 3-6
(8) Concatenate lists of data points in each cluster into a list L
(9) Return L as a set of all synthesized data points

รูปที่ 2.5 อัลกอริทึมสังเคราะห์ข้อมูล

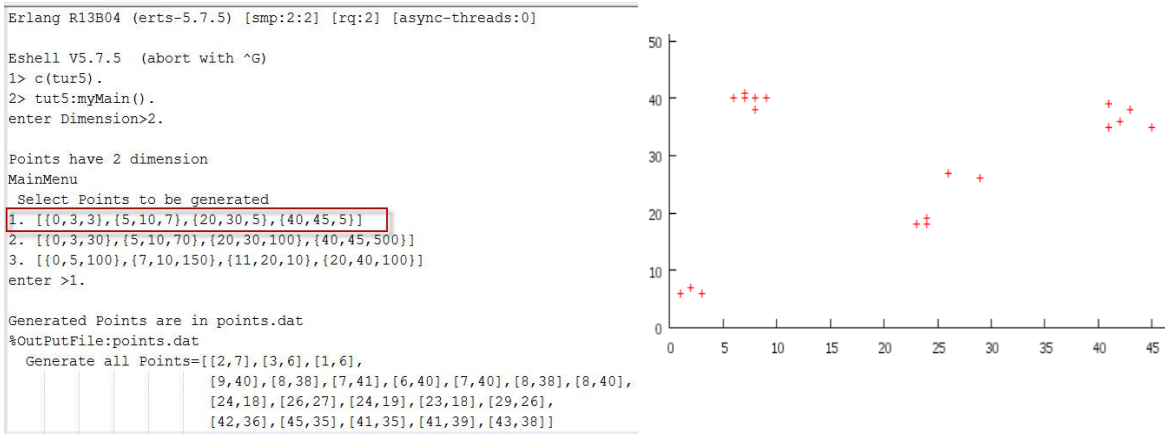
เมื่อผู้ใช้กำหนดมิติของข้อมูล (D), จำนวนกลุ่มของข้อมูล (K), จำนวนข้อมูลในแต่ละกลุ่ม (N1, …, Nk) และขอบเขตล่าง (L) และบน (H) ของพิกัดข้อมูลบนแต่ละแกนมิติ อัลกอริทึมจะ

เริ่มสร้างข้อมูลโดยสุ่มเลขจำนวนเต็มที่อยู่ภายในช่วง H-L บนแต่ละแกนมิติจนครบทุกมิติและจนกระทั่งได้ข้อมูลในแต่ละกลุ่มครบตามจำนวนที่ระบุ การกำหนดขอบเขตพิกัดข้อมูลให้อยู่ภายในช่วง H-L จะใช้กับการสุ่มข้อมูลบนแต่ละแกน ยกเว้นแกนสุดท้ายที่จะใช้การสุ่มอย่างอิสระ ทั้งนี้เพื่อให้กลุ่มของข้อมูลที่ได้ลอยตัวอยู่ได้อย่างอิสระและอาจซ้อนทับกันได้ โดยไม่เกิดการเรียงตัวของกลุ่มข้อมูลในแนวเชิงเส้น อัลกอริทึมในรูปที่ 2.5 สามารถแปลงเป็นรหัสคำสั่งในภาษาเออแลงได้ดังรูปที่ 2.6

```
myGenGroup(K,InitGroup) -> lists:concat(lists:map(
                              fun(A)-> Seed=random:uniform(40),
                                       myDim(Seed,K,A) end,
                            InitGroup)).
myDim(_,_,{_,_,0})->[];
myDim(Rand,K,{L,H,N})->[gen(Rand,{K,L,H})|myDim(Rand,K,{L,H,N-1})].
gen(Rand,{1,L,H})->[random:uniform(H-L)+Rand ];
gen(Rand,{K,L,H})->[random:uniform(H-L)+L|gen(Rand,{K-1,L,H})].
```

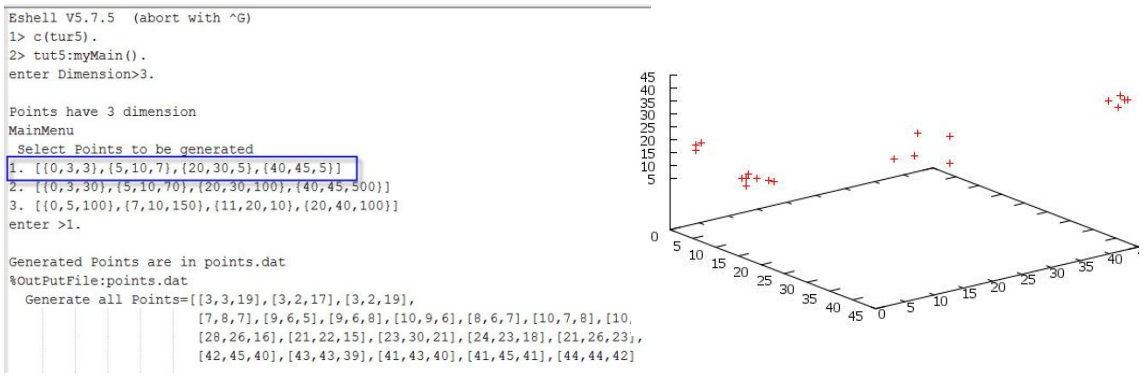
รูปที่ 2.6 ชุดคำสั่งภาษาเออแลง (Erlang) ที่ทำหน้าที่สังเคราะห์ข้อมูล

ชุดคำสั่งดังรูปที่ 2.6 เริ่มการทำงานเมื่อมีการเรียกใช้คำสั่งสร้างข้อมูลสังเคราะห์ เช่น AllPoints = myGenGroup(Dimension, InitGroup) ซึ่งหมายถึงการสั่งให้โมดูล myGenGroup สร้างข้อมูลเพื่อเก็บไว้ในตัวแปร AllPoints โดยมีการกำหนดมิติข้อมูลด้วยอาร์กิวเม้นต์ Dimension และกำหนดลักษณะกลุ่มข้อมูลที่ต้องการด้วยอาร์กิวเม้นต์ InitGroup เช่นถ้ากำหนด InitGroup เป็น [{0,3,3}, {5,10,7}, {20,30,5}, {40,45,5}] และกำหนด Dimension เป็น 2 จะได้กลุ่มข้อมูลสองมิติจำนวนสี่กลุ่มดังรูปที่ 2.7 (หมายเหตุ โปรแกรมสังเคราะห์ข้อมูลดังรูป 2.7(a) สร้างข้อมูลในกลุ่มที่สองที่มีตำแหน่งซ้ำกันคือ [8,38] เมื่อแสดงเป็นภาพกราฟดังรูปที่ 2.7(b) จึงปรากฏจุดเพียง 19 จุด) แต่ถ้ากำหนด Dimension เป็น 3 จะได้กลุ่มข้อมูลสามมิติจำนวนสี่กลุ่มดังรูปที่ 2.8



(a) ข้อมูลในลักษณะ text file (b) ข้อมูลในลักษณะ graphic

รูปที่ 2.7 ข้อมูลสองมิติที่กำหนดให้สังเคราะห์ขึ้น 4 กลุ่มแต่ละกลุ่มมีข้อมูล 3,7,5 และ 5 ตัวตามลำดับ



(a) ข้อมูลในลักษณะ text file (b) ข้อมูลในลักษณะ graphic

รูปที่ 2.8 ข้อมูลสามมิติที่สร้างขึ้นด้วยข้อกำหนดเดียวกับในรูปที่ 2.7

ข้อมูลที่สังเคราะห์ขึ้นดังรูปที่ 2.7 และ 2.8 สร้างจากการกำหนดอาร์กิวเม้นต์ InitGroup เป็น [{0,3,3}, {5,10,7}, {20,30,5}, {40,45,5}] หมายถึงการกำหนดให้สร้างข้อมูลสี่กลุ่ม โดยกลุ่มแรกมีข้อกำหนดเป็น {0,3,3} ตัวเลขจำนวนแรก (คือ 0) หมายถึงขอบเขตล่างของพิกัดข้อมูลในแต่ละมิติ และตัวเลขลำดับที่สอง (คือ 3) หมายถึงขอบเขตบนของพิกัดข้อมูลในแต่ละมิติ นั่นคือข้อมูลที่สร้างขึ้นจะมีพิกัดอยู่ระหว่าง [0..3] ข้อกำหนดนี้จะใช้กับการสังเคราะห์ข้อมูลในทุกมิติยกเว้นมิติสุดท้าย และตัวเลขลำดับที่สาม (คือ 3) หมายถึงจำนวนข้อมูลทั้งหมดในกลุ่มนี้ ดังนั้นเมื่อพิจารณาอาร์กิวเม้นต์ InitGroup ทั้งหมดจะพบว่าผู้ใช้สั่งให้สังเคราะห์ข้อมูลสี่กลุ่ม โดยมีข้อมูลรวมในทุกกลุ่มเป็น 10 ตัว ซึ่งข้อมูลในภาพกราฟแบบ 2 มิติ (รูปที่ 2.7(b)) และแบบ 3 มิติ (รูปที่ 2.8(b)) แสดงการกระจายตัวของกลุ่มข้อมูลที่สังเคราะห์ขึ้น

การออกแบบอัลกอริทึมเพื่อคำนวณความหนาแน่นของข้อมูล

ก่อนที่จะมีการสุ่มตามความหนาแน่นกับข้อมูลที่สังเคราะห์ขึ้น จะต้องมีกระบวนการคำนวณความหนาแน่นของข้อมูล ในงานวิจัยนี้ใช้เทคนิคการเลื่อนกรอบหน้าต่าง (Window sliding algorithm) ไปบนแกนข้อมูล เพื่อนับจำนวนจุดข้อมูลที่ปรากฏในแต่ละกรอบหน้าต่าง ข้อมูลที่ผ่านกระบวนการนี้จะถูกแปลงรูปแบบและลดจำนวนด้วยการแสดงเฉพาะข้อมูลตัวแทน (ใช้จุดกึ่งกลางของกรอบหน้างต่างเป็นตัวแทนข้อมูลทั้งหมดในกรอบหน้าต่างนั้น) จากนั้นเมื่อถึงขั้นตอนการสุ่มตามความหนาแน่น ผู้ใช้จะระบุค่าความหนาแน่นขั้นต่ำของข้อมูลในแต่ละกรอบหน้าต่าง เฉพาะกรอบหน้าต่างที่มีค่าความหนาแน่นถึงเกณฑ์จะถูกนำไปใช้ในขั้นตอนการสุ่ม วิธีการกำหนดกรอบหน้าต่างและการนับข้อมูลในแต่ละกรอบหน้าต่าง อธิบายได้ดังอัลกอริทึมในรูปที่ 2.9

Algorithm Window sliding

Input: a set of data points
Output: a new set of transformed data points annotated with density value

% Initialize windows
(1) Interact with user to obtain dimension value
(2) Generate window grid of size W along dimension axes
% Count density
(3) Sequential move on each window and count number of data points, N, in the window
(4) Record a list of window's central point and its N value in a file F
(5) Return F as a set of transformed data

รูปที่ 2.9 อัลกอริทึมการเลื่อนกรอบหน้าต่างเพื่อคำนวณความหนาแน่นของข้อมูล

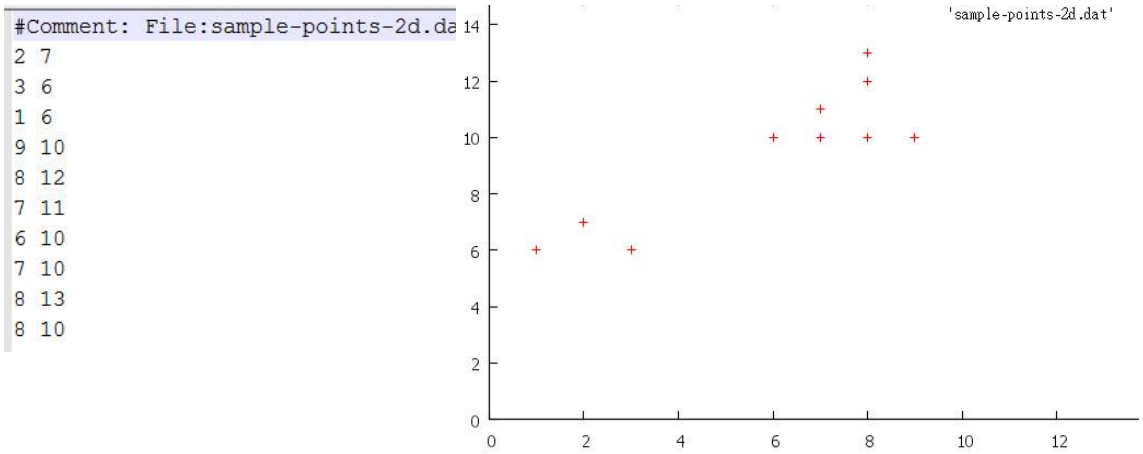
เนื่องจากข้อจำกัดในด้านหน่วยความจำของเครื่องคอมพิวเตอร์ส่วนบุคคลที่ใช้ทดสอบโปรแกรม งานวิจัยนี้จึงได้กำหนดจำนวนมิติของข้อมูลไว้สูงสุดที่ 10 มิติ บนแต่ละแกนมิติกำหนดสเกลข้อมูลที่ 0 ถึง 50 และกำหนดกรอบหน้าต่าง (ค่า W) ไว้ที่ 2 อัลกอริทึมการเลื่อนกรอบหน้าต่างเพื่อคำนวณความหนาแน่นของข้อมูล ได้รับการแปลงเป็นรหัสคำสั่งในภาษาเออแลง แสดงได้ดังรูปที่ 2.10 (คำสั่งในรูปที่ 2.10 แสดงเฉพาะกรณีข้อมูลเป็นสองมิติและสามมิติ ชุดคำสั่งที่สมบูรณ์ซึ่งสามารถทำงานกับข้อมูลได้ถึงสิบมิติดูได้ในภาคผนวก ก)

```
callWindowSliding(Dimension, AllPoints)->
              Stream=myZip(Dimension),
              eachWindow(Dimension, AllPoints, Stream).

myZip(2)-> L=myGenL(0), [[X1,X2]||X1<-L,X2<-L];
myZip(3)-> L=myGenL(0), [[X1,X2,X3]||X1<-L,X2<-L,X3<-L];

myGenL(50)->[];  % axis scaling is in the range 0 to 50
myGenL(Now)->[Now|myGenL(Now+WindowStepSize)]. % WindowStepSize=2

eachWindow(_,_,[])->[];
eachWindow(Del, AllPoints,[Now|StreamT])->
              Lc = count(Now, Del, AllPoints),
              Sum= lists:sum(Lc),
              [{Now,Sum}| eachWindow(Del,AllPoints,StreamT)].
```

รูปที่ 2.10 ชุดคำสั่งภาษาเออแลงที่ทำหน้าที่เลื่อนกรอบหน้าต่างและนับจำนวนข้อมูล

ชุดคำสั่งในรูปที่ 2.10 เริ่มต้นทำงานเมื่อมีการเรียกใช้คำสั่ง callWindowSliding(Dimension, AllPoints) โดย Dimension หมายถึงจำนวนมิติของข้อมูล และ AllPoints หมายถึง

จำนวนข้อมูลสังเคราะห์ทั้งหมด ถ้ากำหนดมิติเป็น 2 และสังเคราะห์ข้อมูลขึ้นสองกลุ่ม (ตามรูปที่ 2.11(a)) กลุ่มที่หนึ่งมีข้อมูล 3 ตัว ปรากฏอยู่ที่พิกัด [x,y] ดังนี้ [2,7], [3,6] และ [1,6] กลุ่มที่สองมีข้อมูล 7 ตัวอยู่ที่พิกัด [9,10], [8,12], [7,11], [6,10], [7,10], [8,13] และ [8,10] ข้อมูลรวมทั้งสองกลุ่มแสดงในลักษณะกราฟสองมิติได้ดังรูปที่ 2.11(b)

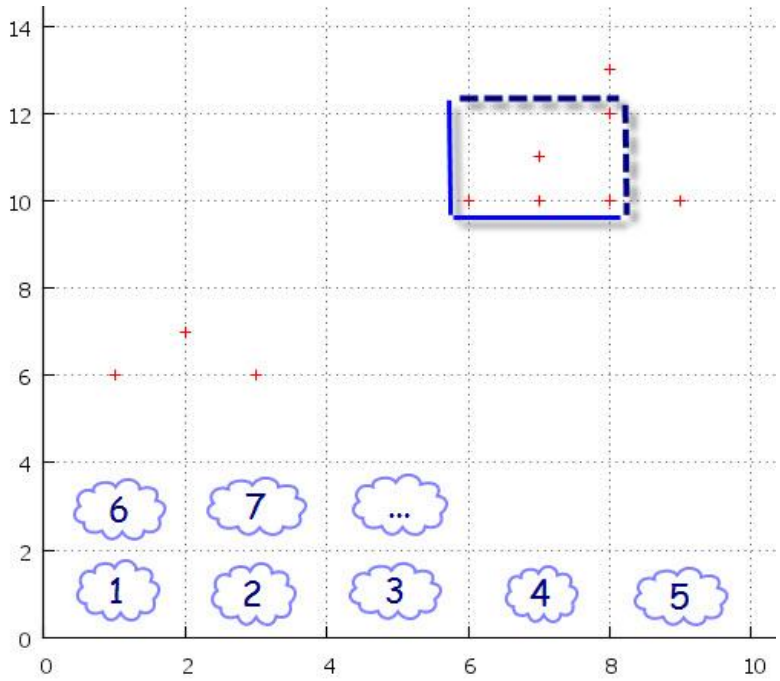


(a) ข้อมูลสองมิติจำนวน 10 ตัว (b) ข้อมูลในลักษณะของกราฟสองมิติ

รูปที่ 2.11 ชุดข้อมูลสังเคราะห์และแผนภาพในลักษณะกราฟสองมิติ

โปรแกรมเลื่อนกรอบหน้าต่างและนับจำนวนข้อมูล จะทำหน้าที่สร้างกรอบหน้าต่าง (ด้วยการเรียกใช้คำสั่ง myZip(Dimension)) ที่มีขนาด 2x2 (ถ้าเป็นข้อมูลสามมิติ กรอบหน้าต่างจะเป็นขนาด 2x2x2) กรอบหน้าต่างทั้งหมดจะอยู่ในลิสต์ชื่อ Stream จากนั้นนับจำนวนจุดข้อมูลในแต่ละกรอบหน้าต่าง ด้วยการเรียกใช้คำสั่ง eachWindow(Dimension, AllPoints, Stream) รูปที่ 2.12 แสดงแผนภาพของการกำหนดกรอบหน้าต่าง (แสดงด้วยเส้นประบาง) และแสดง Stream ของกรอบหน้าต่างด้วยหมายเลข 1,2,3, …, 35 (หมายเลข 1 ถึง 35 นี้ใช้เฉพาะกับกรณีข้อมูลตัวอย่างนี้เท่านั้น ทั้งนี้เพื่อความสะดวกในการแสดงแผนภาพ ในการทำงานจริงของโปรแกรม กรอบหน้าต่างของข้อมูลจะมีตั้งแต่หมายเลข 1 ถึง 625) จากนั้นโปรแกรมจะสแกนเพื่อนับข้อมูลไปเป็นลำดับตั้งแต่กรอบหน้าต่างหมายเลข 1 ไปจนถึง 35

กรอบหน้าต่างหมายเลข 1 จะครอบคลุมขอบเขตของพิกัด [x,y] ในช่วง [0..1.99, 0..1.99] โดยไม่รวมขอบเขตบนที่พิกัด x=2 และ y=2 และกรอบหน้าต่างหมายเลข 2 จะครอบคลุมขอบเขตของพิกัด [x, y] ในช่วง [2..3.99, 0..1.99] เรียงเป็นลำดับเช่นนี้ไปจนถึงกรอบหน้าต่างหมายเลข 35



รูปที่ 2.12 การกำหนดกรอบหน้าต่างและขอบเขตการนับจำนวนข้อมูลในแต่ละกรอบหน้าต่าง

ในรูปที่ 2.12 แสดงขอบเขตของกรอบหน้าต่างหมายเลข 29 ที่อยู่ในช่วงพิกัด [6..7.99, 10..11.99] โดยใช้เส้นประหนาแสดงให้เห็นว่าขอบเขตบนจะไม่รวมพิกัด y=12 และขอบเขตด้านขวาจะไม่รวมพิกัด x=12 ดังนั้นข้อมูลที่อยู่ภายในขอบเขตของกรอบหน้าต่างที่ 29 จะประกอบด้วยข้อมูล 3 ตัวคือ [6,10], [7,10] และ [7,11] กรอบหน้าต่างนี้มีตำแหน่งกึ่งกลางอยู่ที่พิกัด [7,11] ดังนั้นข้อมูลทั้ง 3 ตัวจะถูกเปลี่ยนรูปแบบการเก็บเป็น {[7,11], 3} ซึ่งจะเป็นรูปแบบที่ย่อลงกว่าการเก็บจุดข้อมูลที่แท้จริง และจะช่วยให้การเก็บข้อมูลใช้เนื้อที่น้อยลงด้วย จากรูปที่ 2.12 กรอบหน้าต่างทั้งหมดที่มีข้อมูลปรากฏอยู่สรุปได้ดังนี้

Window 16:

ขอบเขต: [0..1.99, 6..7.99]

ข้อมูล: [1,6]

ข้อมูลที่แปลงแล้ว: {[1,7],1}

Window 17:

ขอบเขต: [2..3.99, 6..7.99]

ข้อมูล: [2,7], [3,6]

ข้อมูลที่แปลงแล้ว: { [3,7], 2}

Window 29:

ขอบเขต: [6..7.99, 10..11.99]

ข้อมูล: [6,10], [7,10], [7,11]

ข้อมูลที่แปลงแล้ว: {[7,11],3}

Window 30:

ขอบเขต: [8..9.99, 10..11.99]

ข้อมูล: [8,10], [9,10]

ข้อมูลที่แปลงแล้ว: {[9,11],2}

Window 35:

ขอบเขต: [8..9.99, 12..13.99]

ข้อมูล: [8,12], [8,13]

ข้อมูลที่แปลงแล้ว: {[9,13],2}

งานวิจัยนี้ได้มีการทดสอบจัดกลุ่มข้อมูลกับข้อมูลที่ถูกแปลงรูปแบบ ด้วยวิธีการเลื่อนกรอบหน้าต่างเพื่อพิจารณาว่าการใช้พิกัดตัวแทนของข้อมูลในกรอบหน้าต่าง เมื่อนำไปจัดกลุ่มแล้วจะให้ผลลัพธ์แตกต่างจากการจัดกลุ่มกับข้อมูลดั้งเดิมมากน้อยเพียงใด โดยการจัดกลุ่มกับข้อมูลที่ผ่านการแปลงรูปแบบ จะต้องมีขั้นตอนการสร้างข้อมูลกลับเพื่อให้มีจำนวนจุดข้อมูลเหมือนเดิม (จำนวนจุดเท่ากับข้อมูลเดิม แต่พิกัดข้อมูลจะเปลี่ยนไป) ข้อมูลที่ถูกสร้างกลับจะเป็นดังนี้

Generate back to data points

Window 16: data = { [1,7], 1 } ⟹ [1,7]

Window 17: data = { [3,7], 2 } ⟹ [3,7], [3,7]

Window 29: data = { [7,11], 3 } ⟹ [7,11], [7,11], [7,11]

Window 30: data = { [9,11], 2 } ⟹ [9,11], [9,11], [9,11]

Window 35: data = { [9,13], 2 } ⟹ [9,13], [9,13]

ผลการจัดกลุ่มเมื่อเปรียบเทียบระหว่างชุดข้อมูลเดิมที่สังเคราะห์ขึ้น กับชุดข้อมูลที่ถูกแปลงกลับจากวิธีการเลื่อนกรอบหน้าต่าง แสดงได้ดังรูปที่ 2.13

Original data set		Data set generated back from Window sliding	
Cluster 1	Cluster 2	Cluster 1	Cluster 2
[1,6]	[6,10], [7,10], [7,11],	[1,7]	[7,11], [7,11], [7,11],
[2,7]	[8,10], [8,12], [8,13]	[3,7]	[9,11], [9,11], [9,13],
[3,6]	[9,10]	[3,7]	[9,13]
Mean of cluster 1 = [(1+2+3)/3, (6+7+6)/3] = [2.0, 6.33]		Mean of cluster 1 = [(1+3+3)/3, (7+7+7)/3] = [2.33, 7.0]	
Mean of cluster 2 = [(6+7+7+8+8+8+9)/7, (10+10+11+10+12+13+10)/7] = [7.57, 10.86]		Mean of cluster 2 = [(7+7+7+9+9+9+9)/7, (11+11+11+11+11+13+13)/7] = [8.14, 11.57]	

Distance of mean shift = $\sqrt{(2.0-2.33)^2+(6.33-7.0)^2}$ + $\sqrt{(7.57-8.14)^2+(10.86-11.57)^2}$

= 0.75 + 0.9

= 1.65

รูปที่ 2.13 เปรียบเทียบผลการจัดกลุ่มกับชุดข้อมูลดั้งเดิมและผลการจัดกลุ่มกับข้อมูลที่ถูกแปลงกลับจากอัลกอริทึม Window sliding

จากรูปที่ 2.13 เมื่อวัดระยะห่างของค่าจุดกึ่งกลางกลุ่มของข้อมูลสังเคราะห์ทั้งสองกลุ่ม (ได้แก่ [2, 6.33] และ [7.57, 10.86]) เทียบกับจุดกึ่งกลางกลุ่มของข้อมูลที่ถูกแปลงกลับ ([2.33, 7] และ [8.14, 11.57]) ด้วยวิธีการวัดระยะแบบยูคลิเดียน (Euclidean distance) พบว่าจุดกึ่งกลางกลุ่มเคลื่อนไปเป็นระยะทางเพียง 1.65 ดังนั้นวิธีการแปลงข้อมูลด้วยเทคนิคการเลื่อนกรอบหน้าต่าง (Window sliding algorithm) จึงมีความเหมาะสมที่จะนำไปใช้ในการประมวลผลข้อมูลก่อนที่จะส่งต่อไปยังขั้นตอนการสุ่มข้อมูลตามความหนาแน่นได้

การออกแบบอัลกอริทึมเพื่อสุ่มข้อมูลตามความหนาแน่น

อัลกอริทึมการสุ่มข้อมูลตามความหนาแน่น จะใช้ข้อมูลที่ผ่านการแปลงรูปแบบและวัดความหนาแน่นด้วยอัลกอริทึม Window sliding โดยผู้ใช้จะเป็นผู้กำหนดเกณฑ์ขั้นต่ำของค่าความหนาแน่นในแต่ละกรอบหน้าต่าง เฉพาะกรอบหน้าต่างที่มีความหนาแน่นถึงเกณฑ์เท่านั้นที่จะถูกคัดเลือกไปใช้เพื่อการจัดกลุ่มข้อมูล โดยก่อนที่จะเริ่มกระบวนการจัดกลุ่มข้อมูล จะต้องมีการแปลงรูปแบบข้อมูลกลับไปเป็นจุดข้อมูล วิธีการแปลงจะใช้จุดกึ่งกลางกรอบหน้าต่างเป็นพิกัดของข้อมูล จากนั้นสร้างพิกัดข้อมูลนั้นซ้ำให้มีจำนวนเท่ากับความหนาแน่นของกรอบหน้าต่างนั้นๆ ข้อมูลที่ถูกแปลงกลับเหล่านี้จะถูกนำไปผ่านอัลกอริทึมสุ่มข้อมูล ที่ใช้เทคนิคการสุ่มและโครงสร้างการเก็บข้อมูลที่แตกต่างกันดังนี้

- Density-biased, Hashing ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บไว้ในตารางแฮชที่กำหนดให้มีขนาดใหญ่สามารถเก็บข้อมูลได้ปริมาณมาก งานวิจัยนี้ใช้สมมติฐานว่าเทคนิคนี้เป็นกรณีที่มีเนื้อที่เก็บข้อมูลได้ไม่จำกัด ทำให้ไม่ต้องคำนึงถึงการชนกันของข้อมูล
- Density-biased, Reservoir + Hashing ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด และการเข้าถึงเนื้อที่เก็บข้อมูลใช้เทคนิคแฮชชิงด้วยฟังก์ชัน (n mod r)+1 เมื่อ n คือลำดับที่ของข้อมูล และ r คือจำนวนเนื้อที่ใน Reservoir แต่ละเนื้อที่จะเก็บข้อมูลได้หนึ่งตัว ดังนั้นถ้าเกิดการชนกันของข้อมูลที่ผ่านการคำนวณด้วยฟังก์ชันแฮช ข้อมูลเดิมใน Reservoir จะถูกข้อมูลใหม่บันทึกทับ
- Density-biased, Reservoir + Simple random sampling ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด โดยการเลือกเก็บข้อมูลลง Reservoir จะใช้วิธีการสุ่มอย่างง่าย

- Density-biased, Reservoir + Rejection sampling ข้อมูลที่มีความหนาแน่นถึงเกณฑ์ จะถูกเก็บลงโครงสร้างข้อมูล Reservoir ที่มีขนาดจำกัด โดยการเลือกเก็บข้อมูลลง Reservoir จะใช้วิธีการสุ่มสองครั้งด้วยเทคนิค Rejection sampling การสุ่มครั้งแรกเป็นการสุ่มเลือกข้อมูล การสุ่มครั้งที่สองเป็นการสุ่มตัวเลข uniform ที่มีค่าอยู่ระหว่าง [0..1] จากนั้นเทียบค่า uniform ที่สุ่มได้ว่าอยู่ภายในช่วงที่กำหนดหรือไม่ (เช่น กำหนดช่วง [0.3..0.7]) ถ้าค่าไม่อยู่ภายในช่วงจะทิ้งข้อมูลนั้นไป (เรียกว่า reject) แต่ถ้าค่าอยู่ภายในช่วงจะเก็บข้อมูลนั้นลง Reservoir กระบวนการจะวนซ้ำจนกว่า Reservoir จะเต็ม

วิธีการสุ่มข้อมูลตามความหนาแน่นทั้งสี่วิธีการข้างต้น มีขั้นตอนการทำงานแสดงได้ดังอัลกอริทึมในรูปที่ 2.14 และแสดงรหัสคำสั่งในรูปแบบของภาษาเออแลงได้ดังรูปที่ 2.15

Algorithm Density-biased sampling

Input: a set of high density data generated back from the Window sliding algorithm
Output: a new set of data samples

---

(1) Display GUI to obtain a desired sampling technique choice from user
(2) If choice = 'Density-biased, Hashing'
(3) Then Hash each data point to store in a hash table T
(4) Return T as an output
(5) If choice = 'Density-biased, Reservoir+Hashing'
(6) Then Interactive with user to obtain reservoir size
(7) Hash each data point to store in a reservoir R
(8) If collision occurs, the stored data is replaced with a new one
(9) Repeat steps 7-8 until there is no more data point, and return R as output
(10) If choice = 'Density-biased, Reservoir+Simple Random Sampling'
(11) Then Interact with user to obtain bin size
(12) Randomly select data point to store in a reservoir R
% random sampling without replacement
(13) Repeat step 12 until R is full, and return R as an output
(14) If choice = 'Density-biased, Reservoir+Rejection Sampling'
(15) Then Interact with user to obtain bin size and interval I, I $\in$ [0.0..0.5]
(16) Randomly select data point D % sampling without replacement
(17) Generate a uniform random number U from the range [0.0 .. 1.0]
(18) If U is within the range [0.5-I .. 0.5+I], then store D in R
(19) Otherwise, reject and discard D
(20) Repeat steps 16-19 until R is full, and return R as an output

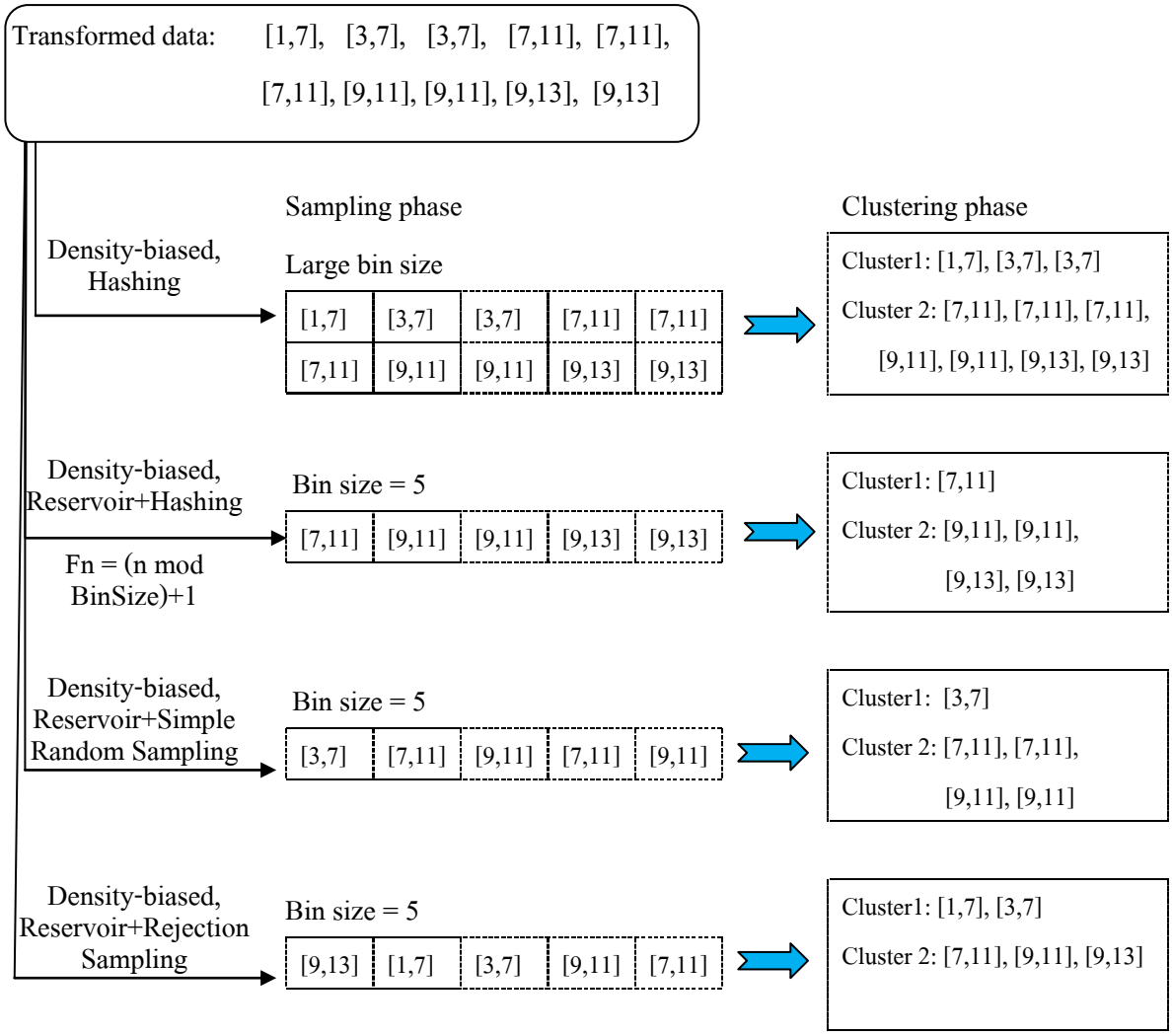
---

รูปที่ 2.14 อัลกอริทึมการสุ่มข้อมูลตามความหนาแน่น

```
% --- Density-biased sampling
%
%     generate back from slidingWindows -> POINTs
%     input arguments: WindowsList,Den
%
genWin2P([])->[];
genWin2P([{P,Den}|T])->
        if Den>=1 -> dup(center(P),Den)++genWin2P(T);
        true -> genWin2P(T)
     end.

%      take Bin elements
specificDensBin(Bin,L,Den) -> take(Bin,specificDens(L,Den)).

specificDens([],_)->[];
specificDens([{P,D}|T],Den)->
       if D>=Den -> [{P,D}|specificDens(T,Den)];
       true -> specificDens(T,Den)
       end.

%--- Simple Random with density bias
simpleRandom(_,_,0)->[];
simpleRandom(WindowL,Dens,Bin)->
    Nth=random:uniform(length(WindowL)),
    {P,D}=lists:nth(Nth,WindowL),
    if D>=Dens -> [{P,D}|simpleRandom(WindowL,Dens,Bin-1)]  ;
      true-> simpleRandom(WindowL,Dens,Bin)
    end.

%--- Rejection sampling with density bias
rejectionRandom(_,_,_,0)->[];
rejectionRandom(Para,WindowL,Dens,Bin)->
    Nth=random:uniform(length(WindowL)),
    Par=random:uniform(),
    {P,D}=lists:nth(Nth,WindowL),
    if (0.5-Para)=<Par,Par=<(0.5+Para),D>=Dens ->
      [{P,D}|rejectionRandom(Para,WindowL,Dens,Bin-1)]  ;
      true-> rejectionRandom(Para,WindowL,Dens,Bin)
    end.
 % ---------------------
```

รูปที่ 2.15 ชุดคำสั่งภาษาเออแลงที่ทำหน้าที่สุ่มข้อมูลตามความหนาแน่น

จากข้อมูลตัวอย่างสองกลุ่มที่แปลงกลับจากอัลกอริทึม Window sliding และกำหนดค่าความหนาแน่นขั้นต่ำเป็น 1 จะประกอบด้วยข้อมูล 10 ตัว ดังนี้ [1,7], [3,7], [3,7], [7,11], [7,11], [7,11], [9,11], [9,11], [9,13], [9,13] เมื่อสุ่มตามความหนาแน่นด้วยเทคนิคต่างๆกัน จะได้ผลลัพธ์การสุ่มและการจัดกลุ่มข้อมูล สรุปเป็นแผนภาพได้ดังรูปที่ 2.16



รูปที่ 2.16 แผนภาพเปรียบเทียบวิธีการสุ่มข้อมูลตามความหนาแน่นด้วยเทคนิคที่ต่างกันสี่แบบ

ข้อมูลที่ผ่านขั้นตอนการสุ่มตามความหนาแน่นดังรูปที่ 2.16 จะถูกส่งต่อไปยังขั้นตอนการจัดกลุ่มข้อมูลด้วยโปรแกรม k-means clustering (แสดงชุดคำสั่งในโปรแกรมดังรูปที่ 2.17) เพื่อจัดข้อมูลแต่ละตัวเข้ากลุ่ม และสุดท้ายจะคำนวณจุดกึ่งกลางกลุ่มข้อมูล (cluster means, or centroids) ในงานวิจัยนี้จะใช้ค่าจุดกึ่งกลางกลุ่มที่ได้จากข้อมูลที่ผ่านการสุ่มแต่ละแบบ เปรียบเทียบกับค่าจุดกึ่งกลางกลุ่มของข้อมูลเริ่มต้นก่อนที่จะทำ Window sliding เพื่อพิจารณาว่าเทคนิคการสุ่มตามความหนาแน่นแบบใด ให้ผลการจัดกลุ่มที่ใกล้เคียงกับกลุ่มข้อมูลดั้งเดิมมากที่สุด

```
kMeans(PL)->
   {_,N}=read('enter number of clustering>'),
   CL=take(N,sets:to_list(sets:from_list(PL))), %initial centroid
   clustering(1,CL,PL).

% take firts  distinct-n element of list
take(0,_)->[];
take(N,[H|T])->[H|take(N-1,T)].

clustering(N,CL,PL)->
   L1 = lists:map( fun(A) -> nearCentroid(A,CL) end ,PL),
   L2 = transform(CL,L1),
   NewCentroid = lists:map(fun({_,GL})-> findMeans(GL)end, L2),
   N1= N+1,
   if NewCentroid==CL -> NewCentroid; % return new centroid
               N>=90 -> NewCentroid;  % max iterations=90
               true ->  clustering(N1,NewCentroid,PL)
   end.

nearCentroid(Point,CentroidL)->
        LenList=lists:zip(
            lists:map(fun(A)->distance(Point,A)end,CentroidL),
                      CentroidL),
        [{_,Centroid}|_]=lists:keysort(1,LenList),
        {Point,Centroid}.

% transform Point-CentroidList to Centroid-PointList
transform([],_)->[];
transform([C|TC],PC)->[{C,t1(C,PC)} |transform(TC,PC)].

t1(_,[])->[] ;
t1(C1,[H|T])->{P,C}=H,
              if C1==C -> [P|t1(C1,T)];
                 C1=/=C -> t1(C1,T)
              end.
% compute Euclidean distance
distance([],[])-> 0;
distance([X1|T1],[X2|T2])->
             math:sqrt((X2-X1)*(X2-X1)+distance(T1,T2) ).

% findMeans([[1,2],[3,4]]) --> [2.0,3.0]
findMeans(PointL) ->[H|_]=PointL ,Len=length(H),
      AllDim=lists:reverse(allDim(Len,PointL)),
      lists:map(fun(A)-> mymean(A) end, AllDim ).

mymean(L)->lists:sum(L)/length(L).

allDim(0,_)->[];
allDim(D,L)->[eachDimList(D,L)|allDim(D-1,L)].

eachDimList(_,[]) ->[];
eachDimList(N,[H|T]) ->[lists:nth(N,H)|eachDimList(N,T)].
```

รูปที่ 2.17 โปรแกรม k-means clustering ในรูปแบบของภาษาเออแลง

บทที่ 3
การทดสอบโปรแกรม

วิธีการทดสอบโปรแกรมสุ่มข้อมูลและโปรแกรมจัดกลุ่มข้อมูล

การทดสอบโปรแกรมของโครงการวิจัยนี้ มีวัตถุประสงค์หลักเพื่อการทดสอบประสิทธิภาพของโปรแกรมสุ่มข้อมูลตามความหนาแน่นทั้ง 4 เทคนิคได้แก่ (1) Density-biased sampling: hashing (unlimited number of bins), (2) Density-biased sampling: reservoir and hashing, (3) Density-biased sampling: reservoir and simple random sampling, และ (4) Density-biased sampling: reservoir and rejection sampling และเพื่อความสมบูรณ์ในการเปรียบเทียบ ผู้วิจัยได้เพิ่มเติมการจัดกลุ่มข้อมูลกับข้อมูลที่ถูกสุ่มอย่างง่าย โดยไม่มีการพิจารณาค่าความหนาแน่นและไม่ต้องมีการแปลงข้อมูลด้วยเทคนิค window sliding โดยกำหนดขนาดและจำนวนของ bin หรือเนื้อที่เก็บข้อมูลในโครงสร้าง reservoir ให้เท่ากับที่ใช้ในเทคนิคที่ (2), (3) และ (4) ส่วนในเทคนิคที่ (1) จะใช้จำนวน bin ให้มากที่สุดเท่าที่เนื้อที่หน่วยความจำขณะนั้นจะจัดสรรให้ได้ ทั้งนี้เพื่อใช้เป็นเกณฑ์เปรียบเทียบประสิทธิภาพการใช้เนื้อที่หน่วยความจำกับเทคนิคที่ (2)-(4)

เนื่องจากเทคนิคการสุ่มข้อมูลตามความหนาแน่น ต้องการคัดเลือกข้อมูลเพื่อนำไปใช้ในงานจัดกลุ่มข้อมูล การทดสอบประสิทธิภาพของโปรแกรมสุ่มข้อมูลแบบต่างๆ จึงใช้ผลลัพธ์ของการจัดกลุ่มข้อมูลเป็นเกณฑ์ในการเปรียบเทียบ ผลลัพธ์ของโปรแกรมจัดกลุ่มข้อมูลจะเป็นผลของการจัดข้อมูลเข้ากลุ่มโดยแสดงค่าจุดกึ่งกลาง (mean, or centroid) ของแต่ละกลุ่ม ค่าจุดกึ่งกลางกลุ่มของข้อมูลที่ได้จากการจัดกลุ่มข้อมูลที่ถูกสุ่มเลือกด้วยเทคนิคต่างๆ จะถูกนำมาเปรียบเทียบกับค่าจุดกึ่งกลางกลุ่มที่แท้จริงของข้อมูลเริ่มต้น มาตรวัดเพื่อการเปรียบเทียบจะใช้ระยะห่างยูคลิเดียนของจุดกึ่งกลางกลุ่มที่จัดกลุ่มด้วยข้อมูลสุ่มเทียบกับจุดกึ่งกลางที่แท้จริง

นอกจากวัดประสิทธิภาพการจัดกลุ่มกับข้อมูลที่ได้จากการสุ่มแล้ว ผู้วิจัยยังได้วัดประสิทธิภาพการจัดกลุ่มกับข้อมูลที่ผ่านการแปลงรูปแบบด้วยเทคนิค window sliding โดยข้อมูลจะถูกสร้างกลับให้มีจำนวนเท่ากับข้อมูลตั้งต้น แต่พิกัดของข้อมูลจะเปลี่ยนไป จากนั้นจะวัดความคลาดเคลื่อนของจุดกึ่งกลางกลุ่มข้อมูล การทดสอบนี้มีวัตถุประสงค์ที่จะตรวจสอบประสิทธิภาพของโปรแกรม window sliding ว่าสามารถแปลงข้อมูลได้ใกล้เคียงกับข้อมูลจริงมากน้อยเพียงใด

การทดสอบโปรแกรมสุ่มข้อมูลตามความหนาแน่น นอกจากทดสอบประสิทธิภาพการจัดกลุ่มข้อมูลในด้านความคลาดเคลื่อนของจุดกึ่งกลางข้อมูลแล้ว ยังมีการวัดเวลาที่ใช้ประมวลผลและวัดเนื้อที่หน่วยความจำที่ใช้ในการจัดเก็บข้อมูล ในระหว่างกระบวนการสุ่มและกระบวนการจัดกลุ่ม ตัวอย่างคำสั่งในภาษาเออแลงที่ใช้วัดเวลาและวัดเนื้อที่หน่วยความจำแสดงได้ดังรูปที่ 3.1 และ 3.2 ตามลำดับ

```
callWindowSliding(Dim,AllP) -> % initialization
     Stream=myZip(Dim),
                    % window sliding -- running time computation
     {TT,L}=timer:tc(dbs,eachWindow,[Dim,AllP,Stream]),
     T11=TT/1000000,      % time is returned in second
     format("~nTime for window sliding of dimension ~w is ~w
              second ~n",[Dim,T11]),
               L .       % and return Windows List
```

รูปที่ 3.1 คำสั่งในภาษาเออแลงที่ใช้วัดเวลาการประมวลผล

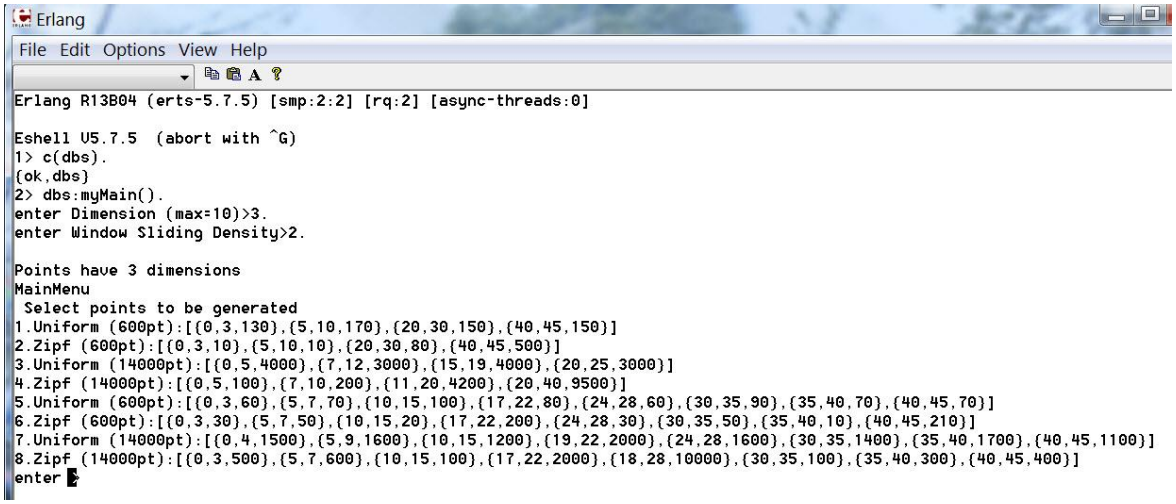
```
mymain(Dim,AllP,WindowList,RealCentroid)->
       format("~nMenu (all choices use the same set of points)~n"),
       format("0. Generate a new set of points~n"),
       format("1. K-means~n"),
       format("2. K-means of Window sliding~n"), % … more choices
       format("11. Exit~n"),
       {_,Menu}=read('enter >'),
       case Menu of
       0-> callMenu() ;
       1-> format("~nRealCentroid with K-means=~w~n",[RealCentroid]),
           Mem=erts_debug:size(AllP),
           MemByte = Mem*4,
           format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
           mymain(Dim,AllP,WindowList,RealCentroid);
       2->  …
```

รูปที่ 3.2 คำสั่งในภาษาเออแลงที่ใช้วัดเนื้อที่หน่วยความจำ

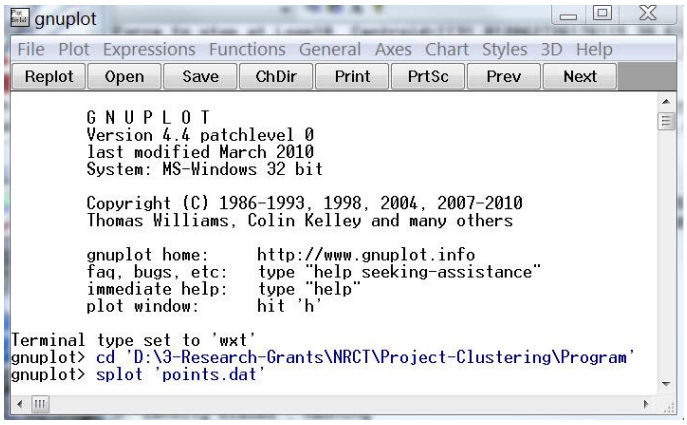
ข้อมูลและเครื่องมือที่ใช้ในการทดสอบ

ข้อมูลที่ใช้ในการทดสอบเป็นข้อมูลในลักษณะของจุด แสดงด้วยพิกัดตำแหน่งข้อมูลที่เป็นเลขจำนวนเต็ม จำนวนแกน (หรือมิติ) ของพิกัดข้อมูลมีได้ตั้งแต่ 1 ถึง 10 มิติ (สำหรับผู้ที่สนใจในการทดสอบกับข้อมูลสูงกว่า 10 มิติ สามารถปรับปรุงรหัสคำสั่งของโปรแกรมที่ปรากฏในภาคผนวก ก ในส่วนของโมดูล myZip(Dimension) ให้รองรับ Dimension ที่มีค่าสูงกว่า 10 ได้) ในการสร้างข้อมูลสังเคราะห์ผู้ใช้สามารถกำหนดจำนวนกลุ่มข้อมูล จำนวนจุดข้อมูลภายในแต่ละกลุ่ม และลักษณะการกระจายตัวของข้อมูลในแต่ละกลุ่ม รูปที่ 3.3 แสดงตัวอย่างเมนูในการเลือกลักษณะของข้อมูลสังเคราะห์

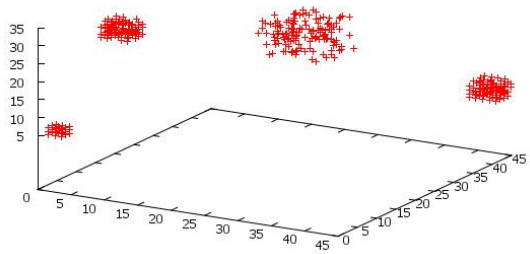
ข้อมูลที่ใช้ในการทดสอบเป็นข้อมูลสังเคราะห์ขนาดสามมิติตามข้อกำหนดในรูปที่ 3.3 ข้อมูลมีจำนวน 8 ชุดแบ่งเป็นข้อมูลขนาดเล็กที่มีข้อมูล 6,000 ตัวมีทั้งที่กระจายตัวแบบสม่ำเสมอและแบบซิฟประกอบด้วยสี่กลุ่มและแปดกลุ่มข้อมูล และข้อมูลขนาดใหญ่ที่มีข้อมูล 14,000 ตัวที่มีการกระจายแบบสม่ำเสมอและแบบซิฟประกอบด้วยสี่กลุ่มและแปดกลุ่มข้อมูล ข้อมูลที่ถูกสร้างขึ้นจะบันทึกไว้ในไฟล์ points.dat ที่มีลักษณะแฟ้มข้อมูลแบบข้อความหรือเท็กซ์ไฟล์ ถ้าผู้ใช้ต้องการดูข้อมูลในลักษณะของกราฟ 2 มิติ หรือ 3 มิติ (ข้อมูลสูงกว่า 3 มิติ ไม่สามารถแสดงภาพกราฟได้) สามารถใช้โปรแกรม GNUPLOT ช่วยในการสร้างภาพกราฟได้ ตัวอย่างจอภาพ GNUPLOT และการสั่งแสดงกราฟ 3 มิติ แสดงดังรูปที่ 3.4 (ถ้าต้องการแสดงกราฟ 2 มิติ ให้เปลี่ยนจากการใช้คำสั่ง splot เป็น plot) และภาพข้อมูลขนาดสามมิติทั้ง 8 ชุดที่ใช้ในการทดสอบโปรแกรมสุ่มและจัดกลุ่มข้อมูลตามความหนาแน่นของโครงการวิจัยนี้แสดงได้ดังรูปที่ 3.5 เครื่องมือที่ใช้ในการทดสอบโปรแกรมเป็นเครื่องคอมพิวเตอร์โน้ตบุค ใช้หน่วยประมวลผล Mobile Intel 4 รุ่น U4100 ความเร็ว 1.30 GHz หน่วยความจำหลัก 2.0 GB ฮาร์ดดิสก์ 250 GB



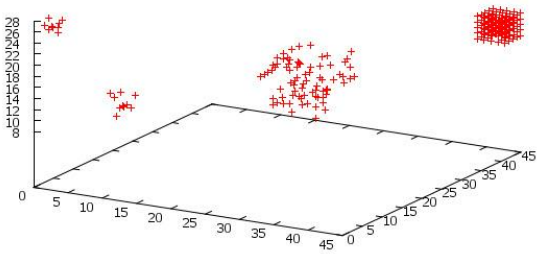
รูปที่ 3.3 ลักษณะและจำนวนจุดข้อมูลในข้อมูลสังเคราะห์ 8 ชุดที่ใช้ในการทดสอบโปรแกรมสุ่มและจัดกลุ่มข้อมูลตามความหนาแน่น



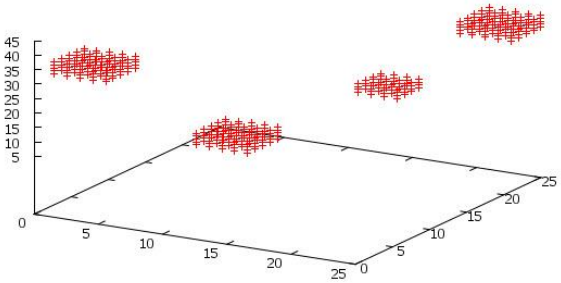
รูปที่ 3.4 ตัวอย่างการใช้โปรแกรม GNUPLOT เพื่อแสดงภาพข้อมูลในลักษณะกราฟ 3 มิติ



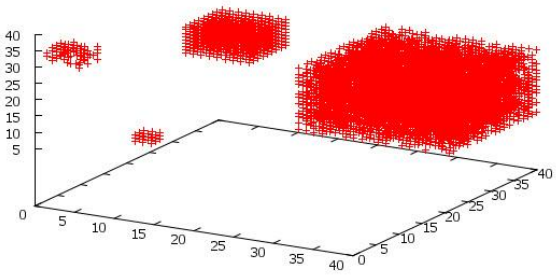
(a) ข้อมูลชุดที่ 1 มีข้อมูล 4 กลุ่ม



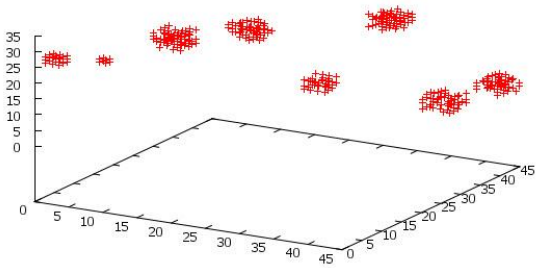
(b) ข้อมูลชุดที่ 2 มีข้อมูล 4 กลุ่ม



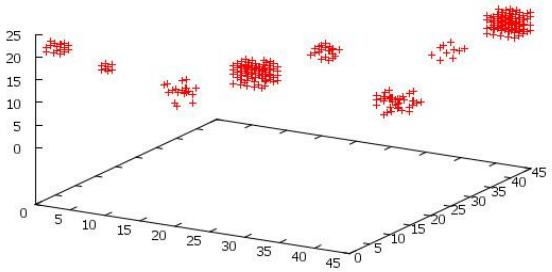
(c) ข้อมูลชุดที่ 3 มีข้อมูล 4 กลุ่ม



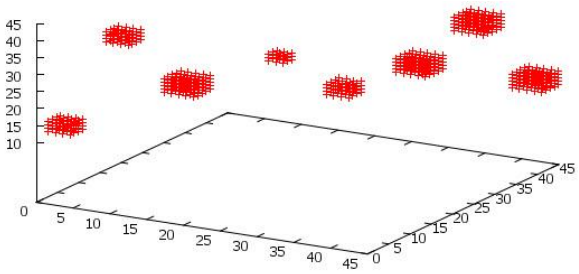
(d) ข้อมูลชุดที่ 4 มีข้อมูล 4 กลุ่ม



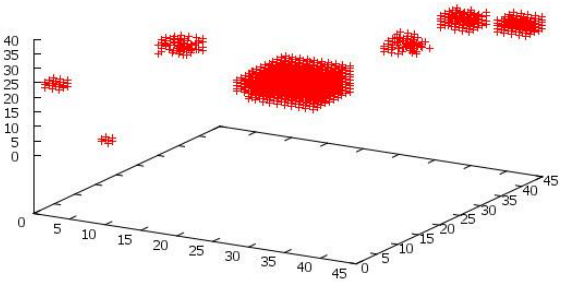
(e) ข้อมูลชุดที่ 5 มีข้อมูล 8 กลุ่ม



(f) ข้อมูลชุดที่ 6 มีข้อมูล 8 กลุ่ม



(g) ข้อมูลชุดที่ 7 มีข้อมูล 8 กลุ่ม



(h) ข้อมูลชุดที่ 8 มีข้อมูล 8 กลุ่ม

รูปที่ 3.5 ข้อมูลสังเคราะห์ทั้ง 8 ชุดที่มีการกระจายข้อมูลแบบสม่ำเสมอ (ภาพด้านซ้าย) และกระจายแบบซิฟ (ภาพด้านขวา)

ผลการทดสอบการสุ่มข้อมูลเพื่อการจัดกลุ่มข้อมูล

ข้อมูลที่ใช้ในการทดสอบประกอบด้วย 8 ชุดข้อมูล (เรียงลำดับหมายเลขชุดข้อมูลตามที่ปรากฏในรูปที่ 3.3) โดยจะทดสอบสังเคราะห์ข้อมูลทั้งที่เป็นข้อมูล 2 มิติ และ 3 มิติ ผลการทดสอบแสดงได้ดังตารางที่ 3.1-3.8 ข้อมูลที่แสดงในตารางเป็นเวลาทั้งหมดที่ใช้ในการสุ่มข้อมูลและจัดกลุ่มข้อมูล หน่วยความจำที่ใช้ในการเก็บข้อมูล และความคลาดเคลื่อนของจุดกึ่งกลางข้อมูลรวมทุกกลุ่ม ข้อมูลที่แสดงในแนวนอนของตารางเป็นวิธีการจัดกลุ่มข้อมูลร่วมกับเทคนิคการสุ่มข้อมูลในแบบต่างๆ เทคนิคเหล่านี้ประกอบด้วย

(1) K-means เป็นการรัน k-means กับข้อมูลสังเคราะห์ที่ยังไม่ได้ผ่านการแปลงและการสุ่ม ผลลัพธ์ที่ได้จัดเป็นจุดกึ่งกลางกลุ่มที่แท้จริง ใช้เป็นค่ามาตรฐานเพื่อเปรียบเทียบแต่ละเทคนิค

(2) K-means, window sliding เป็นการจัดกลุ่มด้วยโปรแกรม k-means กับข้อมูลทุกจุดที่ถูกแปลงพิกัดข้อมูลด้วยเทคนิค window sliding

(3) K-means, DBS&Hashing การสุ่มข้อมูลใช้เทคนิคการสุ่มตามความหนาแน่น (density-biased sampling, DBS) กับข้อมูลที่ถูกแปลงด้วยเทคนิค window sliding ร่วมกับวิธีการเข้าถึงข้อมูลแบบแฮชและมีเนื้อที่เก็บข้อมูลไม่จำกัด การทดสอบการสุ่มจะกำหนดความหนาแน่น (density) ของการคัดเลือกข้อมูลจาก window sliding ที่แตกต่างกัน

(4) K-means, DBS&Reservoir การสุ่มข้อมูลใช้เทคนิคการสุ่มตามความหนาแน่นที่กำหนด density แตกต่างกัน และใช้เนื้อที่เก็บข้อมูล (reservoir) ที่มีขนาดหรือจำนวน bin ที่แตกต่างกัน เทคนิคการเข้าถึงข้อมูลใช้วิธีแฮชชิง ถ้าเกิดการชนกันจะบันทึกข้อมูลตัวใหม่

(5) K-means, DBS&SRS การสุ่มข้อมูลใช้เทคนิคการสุ่มตามความหนาแน่นที่กำหนด density แตกต่างกัน และใช้เนื้อที่เก็บข้อมูล (reservoir) ที่มีขนาดจำกัด โดยทดลองกำหนดจำนวน bin ที่แตกต่างกัน วิธีการคัดเลือกข้อมูลเก็บลงในแต่ละ bin ใช้วิธีการสุ่มอย่างง่าย (simple random sampling) จนกระทั่งได้ข้อมูลครบทุก bin ในการกำหนดแต่ละค่า bin และค่า density จะทดลองซ้ำ 10 ครั้งแล้วบันทึกค่าเฉลี่ยของการทดลองสิบครั้ง

(6) K-means, DBS&Rejection การสุ่มข้อมูลใช้เทคนิคการสุ่มตามความหนาแน่นที่กำหนด density แตกต่างกัน และใช้เนื้อที่เก็บข้อมูล (reservoir) ที่มีขนาดจำกัด โดยทดลองกำหนดจำนวน bin ที่แตกต่างกัน วิธีการคัดเลือกข้อมูลเก็บลงในแต่ละ bin ใช้วิธีการ rejection sampling ที่ทดลองกำหนดช่วงของค่า uniform สองขนาดคือ range = 0.2 และ 0.4 ทำการสุ่มจนกระทั่งได้ข้อมูลครบทุก bin ในการกำหนดต่ละค่า bin และค่า density จะทดลองซ้ำ 10 ครั้งแล้วบันทึกค่าเฉลี่ยของการทดลองสิบครั้ง

(7) K-means, random เป็นการรัน k-means กับข้อมูลที่ถูกสุ่มโดยตรงจากข้อมูลเริ่มต้น โดยทดลองกำหนดค่า bin แตกต่างกัน ในแต่ละค่า bin จะทดลองซ้ำ 10 ครั้งแล้วบันทึกค่าเฉลี่ยของการทดลองสิบครั้ง

ตารางที่ 3.1 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวนกลุ่ม 4 กลุ่ม ข้อมูลรวม 200 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 1)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 2)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.016	4,800	--	0.016	4,800	--
K-means, sliding window	0.094	2,048	5.798	0.079	3,024	16.466
K-means, DBS&Hashing						
Density=2	0.016	2,000	5.829	0.016	2,232	6.139
Density=3	0.016	1,904	5.860	0.000	1,496	15.402
K-means, DBS&Reservoir						
Density=2 , Bin=15	0.016	1,280	5.461	0.000	592	21.023
Density=2 , Bin=20	0.015	1,552	6.926	0.016	848	19.654
Density=3 , Bin=15	0.000	1,376	3.700	0.000	704	19.960
Density=3 , Bin=20	0.016	1,696	6.072	0.015	928	20.874
K-means, DBS&SRS						
Density=2 , Bin=15	0.030	1,123	6.384	0.014	603	15.552
Density=2 , Bin=20	0.032	1,552	5.408	0.016	798	13.162
Density=3 , Bin=15	0.030	1,249	6.364	0.030	677	13.796
Density=3 , Bin=20	0.046	1,645	6.113	0.016	924	17.914
K-means, DBS&Rejection						
Range=0.2,Dens=2,Bin=15	0.046	1,157	5.923	0.030	596	15.088
Range=0.2,Dens=2,Bin=20	0.061	1,574	7.051	0.031	784	12.653
Range=0.2,Dens=3,Bin=15	0.045	1,343	6.581	0.031	672	13.979
Range=0.2,Dens=3,Bin=20	0.061	1,635	5.226	0.062	900	14.400
Range=0.4,Dens=2,Bin=15	0.015	1,190	6.251	0.031	601	14.550
Range=0.4,Dens=2,Bin=20	0.046	1,560	4.978	0.031	805	13.025
Range=0.4,Dens=3,Bin=15	0.031	1,248	6.346	0.030	702	14.387
Range=0.4,Dens=3,Bin=20	0.046	1,650	5.879	0.016	910	14.916
K-means, random						
Bin=15	0.000	350	4.531	0.000	345	14.556
Bin=20	0.015	456	4.573	0.015	462	12.784

ตารางที่ 3.2 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวนกลุ่ม 8 กลุ่ม ข้อมูลรวม 200 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 5)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 6)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.032	4,800	--	0.031	4,800	--
K-means, sliding window	0.110	2,480	14.028	0.109	2,512	20.081
K-means, DBS&Hashing						
Density=2	0.031	2,120	14.971	0.062	2,272	21.910
Density=3	0.016	1,832	15.953	0.032	1,856	6.800
K-means, DBS&Reservoir						
Density=2 , Bin=15	0.016	912	11.175	0.015	752	33.708
Density=2 , Bin=20	0.032	1,128	10.178	0.015	976	23.544
Density=3 , Bin=15	0.016	952	12.061	0.031	792	30.880
Density=3 , Bin=20	0.016	1,200	9.545	0.015	1,096	23.425
K-means, DBS&SRS						
Density=2 , Bin=15	0.030	805	15.882	0.014	724	25.125
Density=2 , Bin=20	0.016	1,079	13.254	0.031	940	23.998
Density=3 , Bin=15	0.030	877	13.130	0.030	818	23.227
Density=3 , Bin=20	0.031	1,145	16.379	0.030	1,083	22.406
K-means, DBS&Rejection						
Range=0.2,Dens=2,Bin=15	0.046	767	14.339	0.030	757	23.015
Range=0.2,Dens=2,Bin=20	0.062	992	14.921	0.046	997	25.493
Range=0.2,Dens=3,Bin=15	0.062	884	13.230	0.030	809	23.398
Range=0.2,Dens=3,Bin=20	0.061	1,159	13.656	0.046	1,089	22.631
Range=0.4,Dens=2,Bin=15	0.030	765	14.264	0.030	717	23.648
Range=0.4,Dens=2,Bin=20	0.030	1,064	13.753	0.016	966	24.574
Range=0.4,Dens=3,Bin=15	0.030	910	16.398	0.016	817	21.336
Range=0.4,Dens=3,Bin=20	0.030	1,186	16.948	0.031	1,098	22.708
K-means, random						
Bin=15	0.000	352	13.734	0.016	352	26.949
Bin=20	0.016	470	14.126	0.014	465	22.459

ตารางที่ 3.3 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวนกลุ่ม 4 กลุ่ม ข้อมูลรวม 200 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 1)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 2)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.015	6,400	--	0.078	6,400	--
K-means, sliding window	2.667	6,768	0.841	3.166	14,832	9.096
K-means, DBS&Hashing						
Density=2	0.046	6,672	0.870	0.046	7,248	10.931
Density=3	0.078	6,232	0.962	0.016	2,848	11.184
K-means, DBS&Reservoir						
Density=2 , Bin=15	0.016	1,296	7.996	0.000	824	46.442
Density=2 , Bin=20	0.015	1,744	8.257	0.016	1,032	45.152
Density=3 , Bin=15	0.015	1,320	8.097	0.016	928	31.441
Density=3 , Bin=20	0.032	1,936	7.779	0.031	1,184	14.016
K-means, DBS&SRS						
Density=2 , Bin=15	2.246	1,174	8.566	1.559	661	17.255
Density=2 , Bin=20	2.963	1,520	8.414	1.621	909	18.736
Density=3 , Bin=15	2.136	1,248	9.898	2.339	844	14.660
Density=3 , Bin=20	4.135	1,728	11.040	5.554	1,074	14.630
K-means, DBS&Rejection						
Range=0.2,Dens=2,Bin=15	6.130	1,180	8.285	3.540	664	18.608
Range=0.2,Dens=2,Bin=20	7.190	1,639	13.020	6.067	911	18.498
Range=0.2,Dens=3,Bin=15	4.835	1,225	13.929	8.439	822	20.056
Range=0.2,Dens=3,Bin=20	10.935	1,706	8.690	9.686	1,092	15.188
Range=0.4,Dens=2,Bin=15	2.932	1,258	11.857	1.310	668	17.329
Range=0.4,Dens=2,Bin=20	4.320	1,525	10.071	2.588	916	17.466
Range=0.4,Dens=3,Bin=15	3.665	1,233	10.622	3.416	808	16.926
Range=0.4,Dens=3,Bin=20	4.710	1,660	9.311	6.535	1,101	16.134
K-means, random						
Bin=15	0.014	470	11.071	0.000	470	17.980
Bin=20	0.015	620	6.347	0.000	618	17.198

ตารางที่ 3.4 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวนกลุ่ม 8 กลุ่ม ข้อมูลรวม 200 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 5)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 6)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.031	6,400	--	0.078	6,400	--
K-means, sliding window	2.574	10,408	26.312	2.653	10,960	19.526
K-means, DBS&Hashing						
Density=2	0.140	8,392	41.403	0.202	8,240	29.658
Density=3	0.125	6,752	42.984	0.062	5,360	50.073
K-means, DBS&Reservoir						
Density=2 , Bin=15	0.031	1,568	60.941	0.031	936	107.052
Density=2 , Bin=20	0.016	1,912	50.879	0.031	1,152	94.188
Density=3 , Bin=15	0.032	1,568	60.941	0.015	984	89.251
Density=3 , Bin=20	0.032	1,920	50.795	0.032	1,272	70.949
K-means, DBS&SRS						
Density=2 , Bin=15	2.245	960	25.958	1.258	740	41.569
Density=2 , Bin=20	2.605	1,230	25.691	1.216	992	27.188
Density=3 , Bin=15	1.621	1,004	25.550	2.917	884	36.116
Density=3 , Bin=20	3.322	1,358	27.006	2.823	1,164	29.266
K-means, DBS&Rejection						
Range=0.2,Dens=2,Bin=15	2.916	919	26.643	3.667	748	35.960
Range=0.2,Dens=2,Bin=20	5.568	1,232	25.011	4.195	1,006	34.826
Range=0.2,Dens=3,Bin=15	3.728	1,024	24.290	3.603	867	31.341
Range=0.2,Dens=3,Bin=20	7.581	1,401	24.721	7.425	1,147	35.210
Range=0.4,Dens=2,Bin=15	1.933	918	23.081	1.450	730	33.940
Range=0.4,Dens=2,Bin=20	3.478	1,174	23.069	2.386	1,005	33.874
Range=0.4,Dens=3,Bin=15	1.169	1,053	26.291	2.230	879	34.781
Range=0.4,Dens=3,Bin=20	2.526	1,403	27.463	2.605	1,112	36.709
K-means, random						
Bin=15	0.015	465	30.099	0.000	470	38.123
Bin=20	0.015	613	33.859	0.016	616	30.044

ตารางที่ 3.5 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวนกลุ่ม 4 กลุ่ม ข้อมูลรวม 5,000 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 3)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 4)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.265	120,000	--	0.749	120,000	--
K-means, sliding window	1.716	40,528	2.709	1.763	42,512	36.181
K-means, DBS&Hashing						
Density=5	0.296	40,528	2.709	0.265	42,488	36.186
Density=20	0.186	40,528	2.709	0.203	36,760	12.184
K-means, DBS&Reservoir						
Density=5 , Bin=20	0.062	23,408	4.212	0.015	8,880	59.450
Density=5 , Bin=30	0.266	35,752	2.902	0.031	16,016	60.691
Density=20 , Bin=20	0.063	23,408	4.212	0.031	13,440	17.263
Density=20 , Bin=30	0.265	35,752	2.902	0.031	18,416	14.150
K-means, DBS&SRS						
Density=5 , Bin=20	0.062	24,632	8.384	0.045	5,592	26.471
Density=5 , Bin=30	0.093	38,235	4.416	0.031	8,947	17.383
Density=20 , Bin=20	0.092	24,907	7.198	0.031	6,576	17.531
Density=20 , Bin=30	0.125	36,638	6.403	0.047	10,534	11.858
K-means, DBS&Rejection						
Range=0.2,Dens=5,Bin=20	0.124	24,276	6.776	0.014	5,065	17.966
Range=0.2,Dens=5,Bin=30	0.139	38,454	7.963	0.077	8,614	19.657
Range=0.2,Dens=20,Bin=20	0.124	23,686	8.723	0.030	6,827	12.324
Range=0.2,Dens=20,Bin=30	0.110	35,003	5.546	0.093	10,169	10.965
Range=0.4,Dens=5,Bin=20	0.155	23,744	7.560	0.030	6,222	13.543
Range=0.4,Dens=5,Bin=30	0.141	38,102	4.928	0.046	8,929	16.068
Range=0.4,Dens=20,Bin=20	0.094	34,496	7.777	0.015	6,281	16.283
Range=0.4,Dens=20,Bin=30	0.141	37,497	5.128	0.092	9,878	13.381
K-means, random						
Bin=20	0.015	476	13.261	0.015	480	18.357
Bin=30	0.016	713	13.780	0.014	716	20.700

ตารางที่ 3.6 ผลการจัดกลุ่มข้อมูล 2 มิติ จำนวนกลุ่ม 8 กลุ่ม ข้อมูลรวม 5,000 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 7)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 8)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.452	120,000	--	0.825	96,000	--
K-means, sliding window	1.669	40,896	12.452	1.623	34,240	8.839
K-means, DBS&Hashing						
Density=5	0.249	40,896	12.452	0.468	34,192	8.804
Density=20	0.250	40,736	12.438	0.265	25,576	17.299
K-means, DBS&Reservoir						
Density=5 , Bin=20	0.109	17,376	10.270	0.047	2,584	57.139
Density=5 , Bin=30	0.156	25,232	20.170	0.047	4,152	48.285
Density=20 , Bin=20	0.109	17,376	10.270	0.031	5,744	31.854
Density=20 , Bin=30	0.172	25,232	20.170	0.124	9,144	43.889
K-means, DBS&SRS						
Density=5 , Bin=20	0.156	14,948	11.311	0.047	4,707	35.687
Density=5 , Bin=30	0.141	21,934	13.047	0.061	8,640	28.100
Density=20 , Bin=20	0.077	15,488	16.544	0.031	6,867	35.829
Density=20 , Bin=30	0.296	22,555	13.982	0.139	11,404	31.506
K-means, DBS&Rejection						
Range=0.2,Dens=5,Bin=20	0.077	14,652	15.392	0.030	4,921	33.935
Range=0.2,Dens=5,Bin=30	0.233	22,577	13.676	0.078	8,188	27.322
Range=0.2,Dens=20,Bin=20	0.124	15,113	16.562	0.046	6,832	32.852
Range=0.2,Dens=20,Bin=30	0.171	22,289	16.076	0.092	11,528	31.525
Range=0.4,Dens=5,Bin=20	0.077	14,152	16.422	0.046	4,876	32.040
Range=0.4,Dens=5,Bin=30	0.093	21,657	15.257	0.046	7,249	30.607
Range=0.4,Dens=20,Bin=20	0.124	14,950	16.110	0.031	7,136	29.422
Range=0.4,Dens=20,Bin=30	0.265	25,360	13.588	0.093	11,563	27.084
K-means, random						
Bin=20	0.015	480	16.401	0.015	476	39.209
Bin=30	0.016	720	14.492	0.015	720	31.832

ตารางที่ 3.7 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวนกลุ่ม 4 กลุ่ม ข้อมูลรวม 5,000 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 3)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 4)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.234	160,000	--	0.671	160,000	--
K-means, sliding window	49.92	142,160	7.172	49.687	180,824	1.719
K-means, DBS&Hashing						
Density=5	1.232	142,160	7.172	1.622	162,408	1.561
Density=50	1.077	126,000	6.786	0.187	44,024	8.063
K-means, DBS&Reservoir						
Density=5 , Bin=60	0.358	41,904	4.473	0.093	12,024	48.865
Density=5 , Bin=100	0.515	76,824	8.218	0.218	23,496	37.614
Density=50 , Bin=30	0.109	27,048	6.818	0.218	18,928	18.627
Density=50 , Bin=50	0.250	59,552	4.671	0.219	37,096	16.821
K-means, DBS&SRS						
Density=5 , Bin=60	4.556	52,260	18.747	0.513	7,952	14.218
Density=5 , Bin=100	6.303	92,902	12.809	0.873	13,397	12.131
Density=50 , Bin=30	2.714	36,782	13.915	2.777	18,661	11.433
Density=50 , Bin=50	4.710	56,700	25.807	5.693	36,972	9.056
K-means, DBS&Rejection						
Range=0.2,Dens=5,Bin=60	8.768	57,715	12.924	1.184	8,160	15.261
Range=0.2,Dens=5,Bin=100	16.614	93,699	11.839	1.918	13,195	13.828
Range=0.2,Dens=50,Bin=30	7.346	39,560	10.871	11.248	18,883	12.389
Range=0.2,Dens=50,Bin=50	11.685	57,016	15.308	19.999	37,255	10.585
Range=0.4,Dens=5,Bin=60	4.071	57,316	10.453	0.576	8,412	14.134
Range=0.4,Dens=5,Bin=100	7.628	87,054	21.281	1.013	13,843	14.640
Range=0.4,Dens=50,Bin=30	4.275	34,241	27.860	4.960	18,743	10.914
Range=0.4,Dens=50,Bin=60	6.022	60,742	18.905	8.891	36,743	12.282
K-means, random						
Bin=30	0.015	960	33.375	0.030	957	16.895
Bin=50	0.030	1,595	36.196	0.031	1,592	15.884
Bin=60	0.030	1,905	46.326	0.030	1,912	13.821
Bin=100	0.046	3,180	29.615	0.045	3,171	11.413

ตารางที่ 3.8 ผลการจัดกลุ่มข้อมูล 3 มิติ จำนวนกลุ่ม 8 กลุ่ม ข้อมูลรวม 5,000 ตัว

เทคนิคการจัดกลุ่ม และค่าพารามิเตอร์	ข้อมูลกระจายแบบสม่ำเสมอ (ข้อมูลชุดที่ 7)			ข้อมูลกระจายแบบซิฟ (ข้อมูลชุดที่ 8)		
	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน	เวลา (วินาที)	หน่วยความจำ (ไบต์)	ความคลาดเคลื่อน
K-means	0.656	160,000	--	3.635	128,000	--
K-means, sliding window	49.343	135,688	17.505	37.331	139,136	13.952
K-means, DBS&Hashing						
Density=5	4.009	135,688	17.505	3.947	118,904	15.091
Density=50	2.090	107,736	29.897	0.312	39,448	21.388
K-means, DBS&Reservoir						
Density=5 , Bin=60	0.515	38,704	58.061	0.110	6,216	53.317
Density=5 , Bin=100	0.639	63,800	40.372	0.108	5,984	42.314
Density=50 , Bin=30	0.234	27,952	52.241	0.188	21,544	40.378
Density=50 , Bin=50	0.218	41,362	45.231	0.281	35,848	22.198
K-means, DBS&SRS						
Density=5 , Bin=60	2.856	33,294	31.115	0.810	8,386	35.767
Density=5 , Bin=100	4.618	58,164	27.877	1.216	13,184	32.996
Density=50 , Bin=30	2.308	25,734	22.132	4.945	20,540	20.552
Density=50 , Bin=50	4.352	39,438	36.335	8.471	34,148	21.516
K-means, DBS&Rejection						
Range=0.2,Dens=5,Bin=60	6.895	32,595	34.854	1.778	7,935	32.762
Range=0.2,Dens=5,Bin=100	9.407	57,548	27.560	2.401	14,007	37.905
Range=0.2,Dens=50,Bin=30	5.897	25,905	33.680	14.743	21,642	20.130
Range=0.2,Dens=50,Bin=50	10.406	43,680	25.803	18.267	35,542	19.416
Range=0.4,Dens=5,Bin=60	3.478	32,078	30.156	0.686	8,849	34.977
Range=0.4,Dens=5,Bin=100	5.539	57,688	29.205	1.591	12,598	36.491
Range=0.4,Dens=50,Bin=30	2.136	25,670	36.854	6.942	20,830	22.909
Range=0.4,Dens=50,Bin=60	5.383	40,673	34.443	12.168	34,266	22.890
K-means, random						
Bin=30	0.014	960	20.928	0.030	960	40.565
Bin=50	0.030	1,595	24.263	0.030	1,592	31.856
Bin=60	0.030	1,910	16.973	0.030	1,912	31.747
Bin=100	0.032	3,176	19.109	0.046	3,173	25.751

ผลการเปรียบเทียบความคลาดเคลื่อนของการจัดกลุ่มข้อมูล

การจัดกลุ่มตามความหนาแน่น มีวัตถุประสงค์ที่จะพัฒนาเทคนิคการจัดกลุ่มกับข้อมูลขนาดใหญ่ที่สามารถพัฒนาต่อไปให้จัดการกับข้อมูลสตรีมได้ ดังนั้นเทคนิคการสุ่มและการจัดกลุ่มที่ดี จะต้องใช้เนื้อที่หน่วยความจำต่ำ แต่ให้ผลการจัดกลุ่มที่คลาดเคลื่อนไปจากกลุ่มที่แท้จริงเป็นระยะทางน้อยที่สุด ดังนั้นการสรุปผลเทคนิคต่างๆที่ใช้ในการจัดกลุ่ม กำหนดเกณฑ์ในการเปรียบเทียบว่าแต่ละเทคนิคจะต้องใช้หน่วยความจำไม่มากกว่า 30% ของหน่วยความจำที่โปรแกรม k-means ใช้ในการจัดกลุ่มข้อมูลสังเคราะห์ทั้งหมด (ข้อมูลในแถวแรกของตารางที่ 3.1-3.8) เช่น ถ้าการรัน k-means กับข้อมูลทั้งหมดใช้หน่วยความจำ 4,800 ไบต์ เทคนิคการสุ่มและการจัดกลุ่มจะต้องใช้หน่วยความจำไม่เกิน 1,440 ไบต์

เทคนิคการสุ่มและการจัดกลุ่มที่ผ่านเกณฑ์ขั้นต่ำนี้ แสดงเปรียบเทียบได้ดังตารางที่ 3.9 โดยค่าความคลาดเคลื่อนที่แสดงในตารางเป็นค่าเฉลี่ยจากการทดสอบในทุกค่า density และทุกค่า bin เครื่องหมาย ‘--’ ที่ปรากฏในตารางหมายถึงเทคนิคนั้นๆใช้หน่วยความจำมากกว่าเกณฑ์ขั้นต่ำที่กำหนด จึงไม่สามารถนำผลการจัดกลุ่มมาเปรียบเทียบร่วมกับเทคนิคอื่นๆได้

ตารางที่ 3.9 การเปรียบเทียบค่าความคลาดเคลื่อนของกลุ่มข้อมูลเมื่อใช้เทคนิคการสุ่มแต่ละแบบ

ลักษณะของข้อมูล		K-means, DBS &Hashing	K-means, DBS &Reservoir	K-means, DBS&SRS	K-means, DBS &Rejection	K-means, random
ข้อมูลกระจายสม่ำเสมอ						
ขนาดเล็ก	2 มิติ, 4 กลุ่ม	--	--	6.067	6.029	4.552
	2 มิติ, 8 กลุ่ม	--	10.739	14.661	16.471	13.930
	3 มิติ, 4 กลุ่ม	--	8.032	9.479	10.723	8.709
	3 มิติ, 8 กลุ่ม	--	55.889	26.051	25.071	31.979
ขนาดใหญ่	2 มิติ, 4 กลุ่ม	--	3.557	6.600	6.800	13.520
	2 มิติ, 8 กลุ่ม	--	15.22	13.721	15.385	15.446
	3 มิติ, 4 กลุ่ม	--	--	--	--	44.721
	3 มิติ, 8 กลุ่ม	--	48.965	29.364	31.560	20.318
ข้อมูลกระจายแบบซิฟ						
ขนาดเล็ก	2 มิติ, 4 กลุ่ม	--	20.377	15.098	14.124	14.634
	2 มิติ, 8 กลุ่ม	--	27.889	23.689	23.350	24.704
	3 มิติ, 4 กลุ่ม	--	45.873	16.320	17.525	17.589
	3 มิติ, 8 กลุ่ม	--	90.360	33.534	34.580	34.083
ขนาดใหญ่	2 มิติ, 4 กลุ่ม	--	37.888	18.310	15.228	19.528
	2 มิติ, 8 กลุ่ม	--	32.780	31.408	30.598	35.520
	3 มิติ, 4 กลุ่ม	--	30.481	11.709	13.003	14.503
	3 มิติ, 8 กลุ่ม	--	39.551	27.707	28.434	32.479

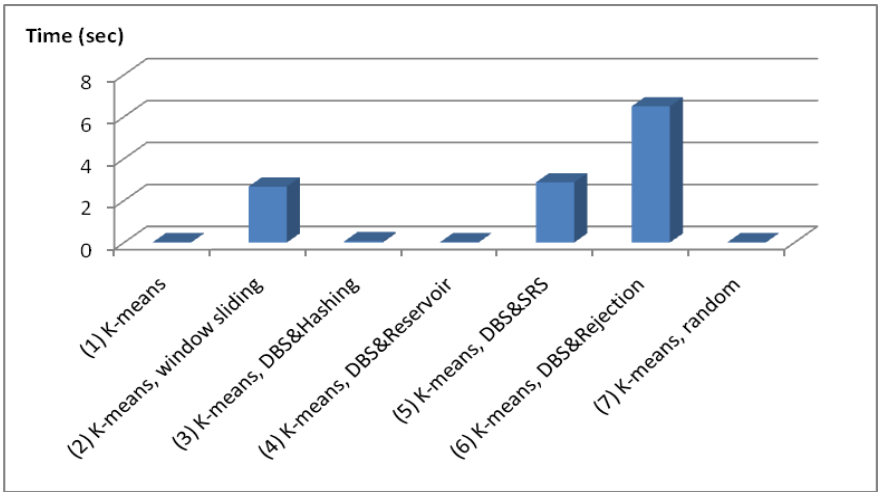
เมื่อพิจารณาจากผลการทดลองตามตารางที่ 3.9 จะเห็นได้ว่าเทคนิค K-means, DBS& Hashing เมื่อทดสอบกับข้อมูลทุกชุด ปรากฏผลเป็นเครื่องหมาย '--' ทั้งนี้เนื่องจากเทคนิคนี้ไม่มีข้อกำหนดเกี่ยวกับจำนวน bin ทำให้ใช้หน่วยความจำมากกว่าเกณฑ์ที่ตั้งไว้ จากผลการเปรียบเทียบค่าความคลาดเคลื่อนของกลุ่มข้อมูล จะเห็นได้ว่าเมื่อกลุ่มข้อมูลมีการกระจายแบบสม่ำเสมอและเป็นข้อมูลขนาดเล็ก เทคนิค K-means, DBS&Reservoir ที่ใช้การเข้าถึงข้อมูลแบบแฮชจะให้ผลลัพธ์ที่ดี แต่ถ้าข้อมูลมีขนาดใหญ่เทคนิค K-means, random จะให้ผลลัพธ์การจัดกลุ่มที่ดี

ในกรณีที่ข้อมูลกระจายแบบซิฟและเป็นข้อมูลขนาดสองมิติ เทคนิค K-means, DBS& Rejection จะให้ผลลัพธ์การจัดกลุ่มที่ดี แต่เมื่อข้อมูลเพิ่มขนาดเป็นสามมิติ เทคนิค K-means, DBS& SRS จะให้ผลลัพธ์การจัดกลุ่มที่ดีกว่า

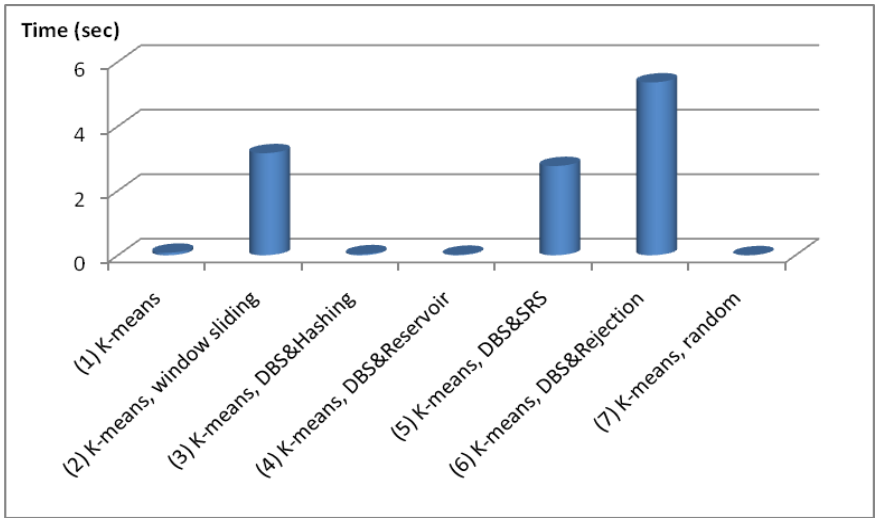
ผลการเปรียบเทียบเวลาที่ใช้ในการสุ่มและจัดกลุ่มข้อมูล

การเปรียบเทียบเวลาในการประมวลผล จะรวมทั้งเวลาในขั้นตอนการสุ่มและเวลาในขั้นตอนการจัดกลุ่มของทั้ง 7 เทคนิค ประกอบด้วย (1) K-means, (2) K-means, window sliding, (3) K-means, DBS&Hashing, (4) K-means, DBS&Reservoir, (5) K-means, DBS&SRS, (6) K-means, DBS&Rejection และ (7) K-means, random

การแสดงผลทดสอบเปรียบเทียบเวลาจะแสดงเฉพาะกรณีของข้อมูล 3 มิติ (ข้อมูลสองมิติให้ผลสอดคล้องเช่นเดียวกับในกรณีสามมิติ) จำแนกเป็นข้อมูลขนาดเล็กจำนวน 4 กลุ่ม (รูปที่ 3.6), ข้อมูลขนาดเล็กจำนวน 8 กลุ่ม (รูปที่ 3.7), ข้อมูลขนาดใหญ่จำนวน 4 กลุ่ม (รูปที่ 3.8) และข้อมูลขนาดใหญ่จำนวน 8 กลุ่ม (รูปที่ 3.9) โดยในแต่ละรูปจะเปรียบเทียบทั้งกรณีข้อมูลกระจายสม่ำเสมอ และข้อมูลกระจายแบบซิฟ

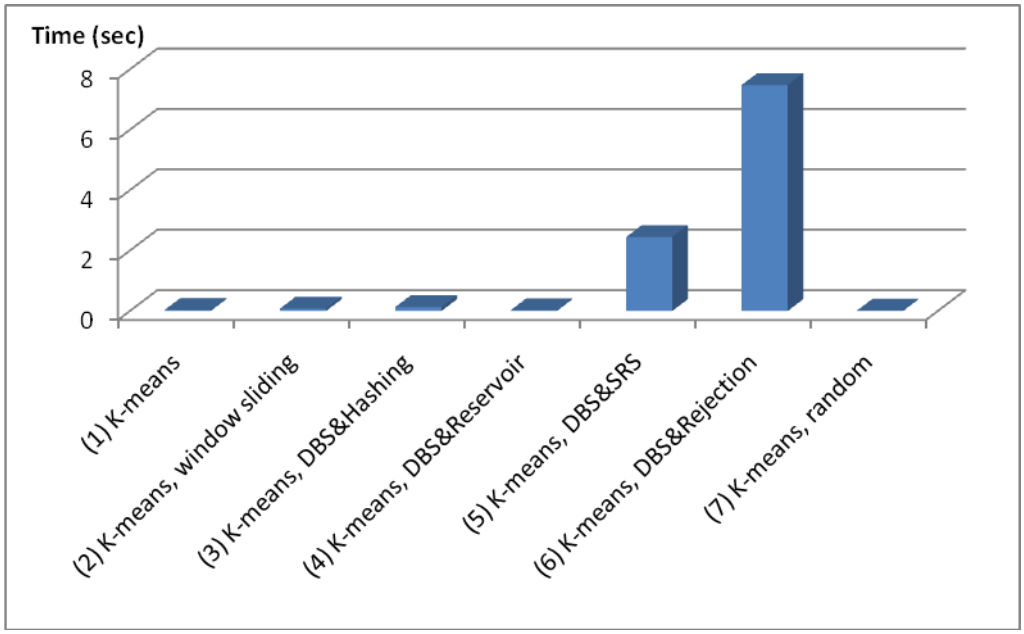


(a) ข้อมูลกระจายแบบ uniform

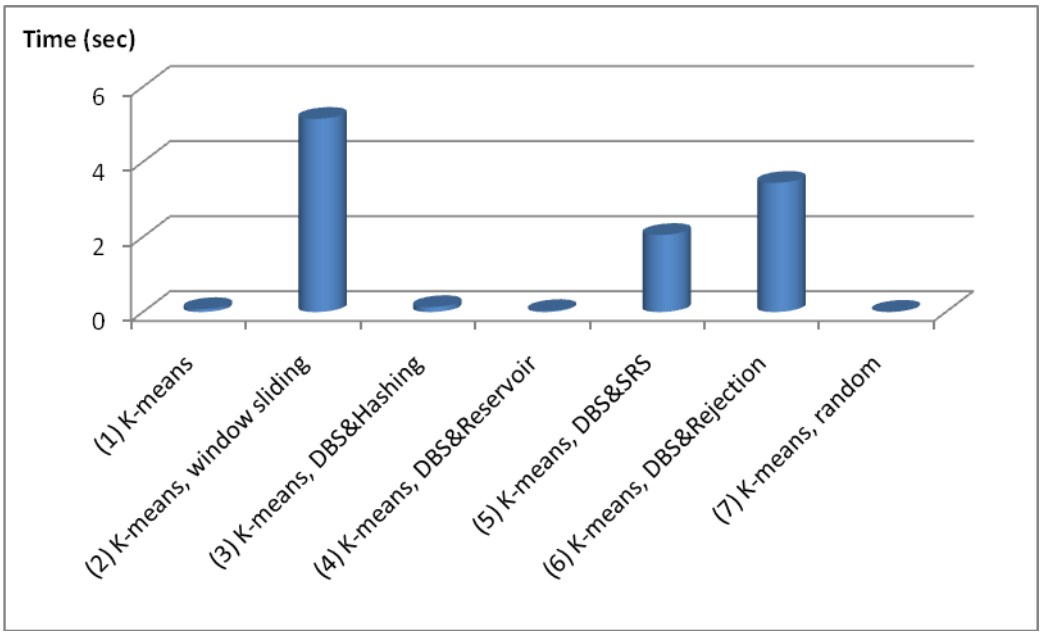


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.6 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 4 กลุ่ม

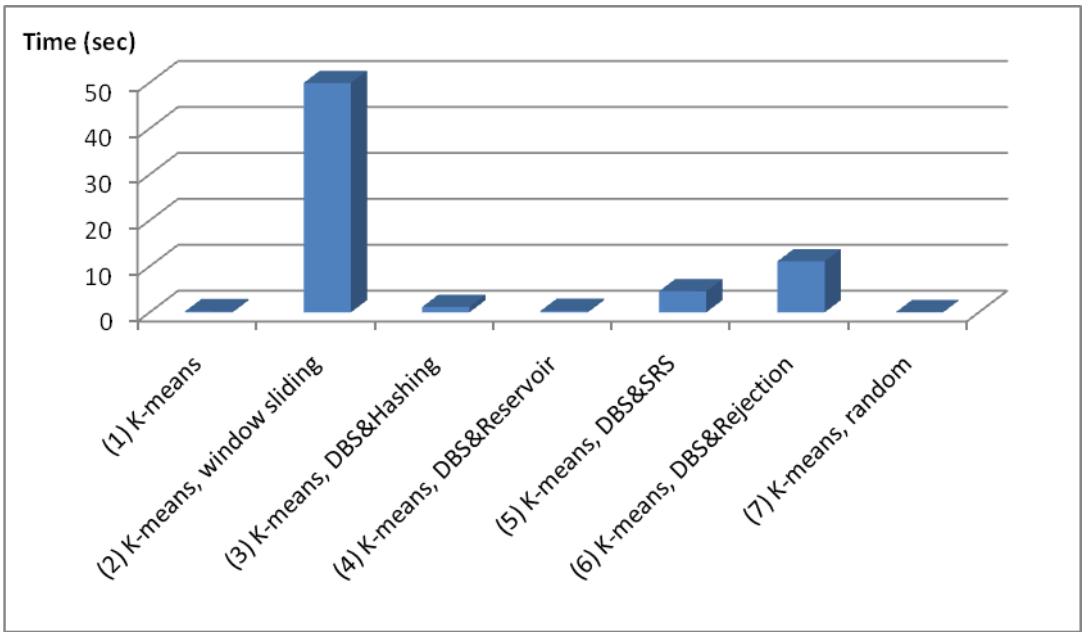


(a) ข้อมูลกระจายแบบ uniform

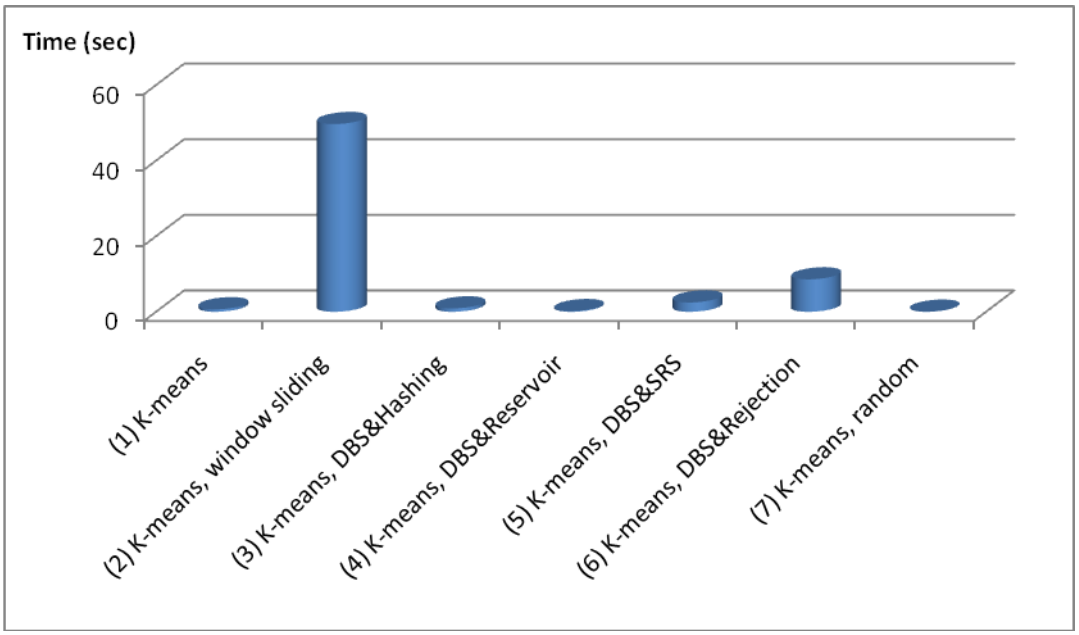


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.7 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 8 กลุ่ม

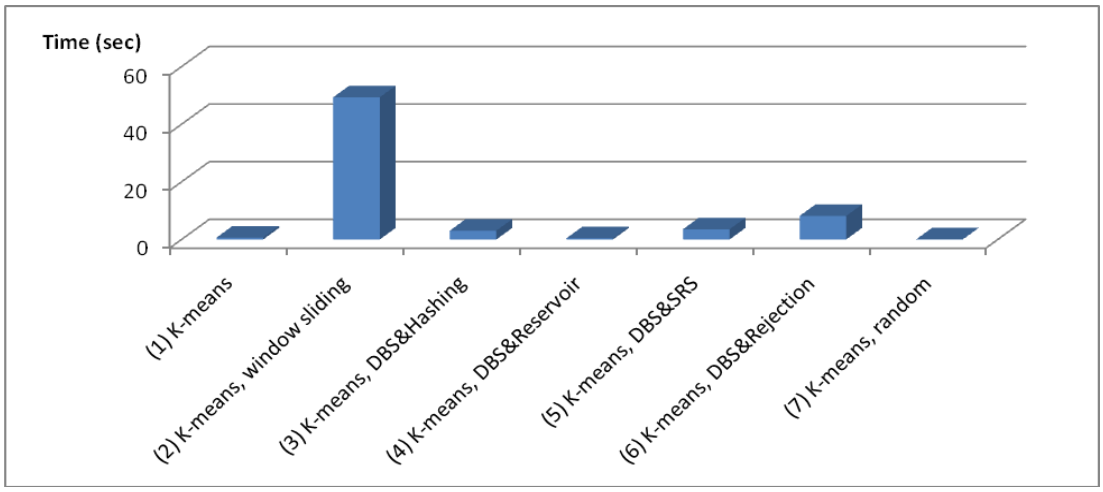


(a) ข้อมูลกระจายแบบ uniform

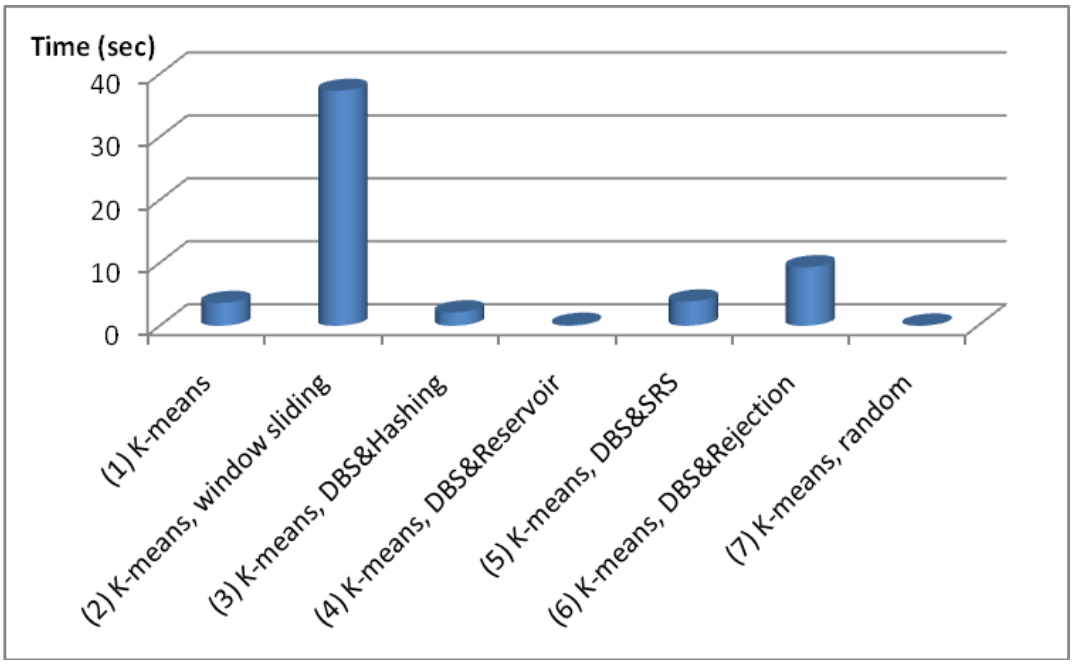


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.8 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 4 กลุ่ม



(a) ข้อมูลกระจายแบบ uniform

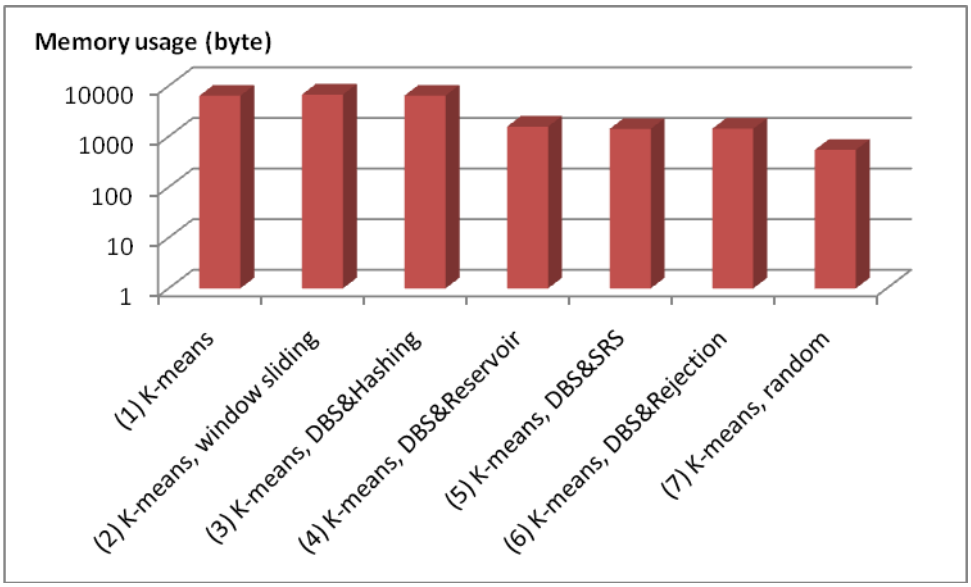


(b) ข้อมูลกระจายแบบ Zipf

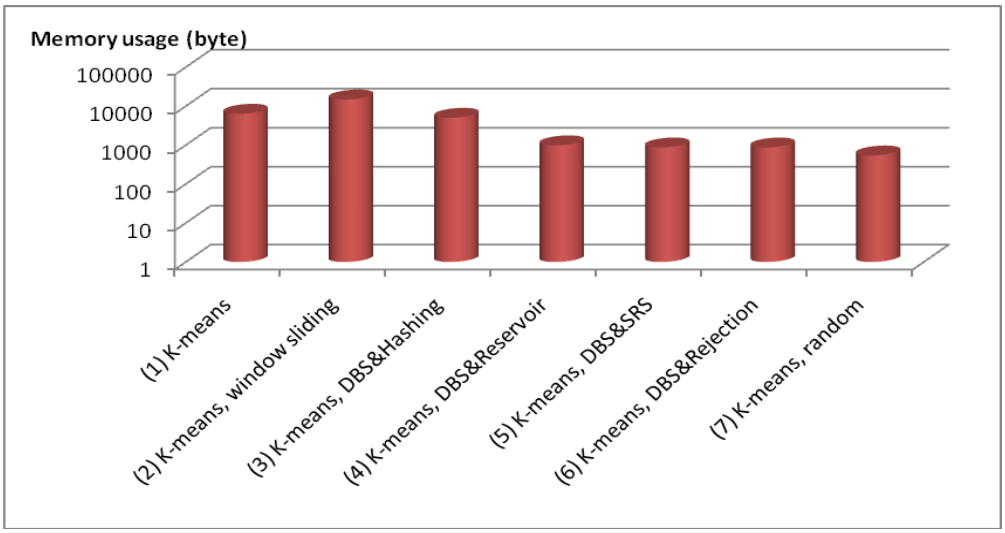
รูปที่ 3.9 เปรียบเทียบเวลาที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 8 กลุ่ม

ผลการเปรียบเทียบหน่วยความจำที่ใช้เพื่อจัดเก็บข้อมูล

การทดสอบหน่วยความจำที่ใช้ในการสุ่มและจัดกลุ่มข้อมูลของแต่ละเทคนิค จะเป็นการวัดขนาดของเนื้อที่หน่วยความจำที่ใช้ในการจัดเก็บข้อมูลมีหน่วยของการวัดเป็นไบต์ กรณีของข้อมูลสองมิติและสามมิติให้ผลสอดคล้องกัน จึงจะแสดงผลการเปรียบเทียบเฉพาะของกรณีข้อมูล 3 มิติ จำแนกตามลักษณะข้อมูลเช่นเดียวกับการเปรียบเทียบเวลา แต่เนื่องจากแต่ละเทคนิคของการสุ่มและจัดกลุ่มข้อมูลใช้เนื้อที่หน่วยความจำที่แตกต่างกันมาก การแสดงภาพกราฟเพื่อการเปรียบเทียบจึงต้องใช้ log scale และแสดงผลการเปรียบเทียบได้ดังรูปที่ 3.10-3.13

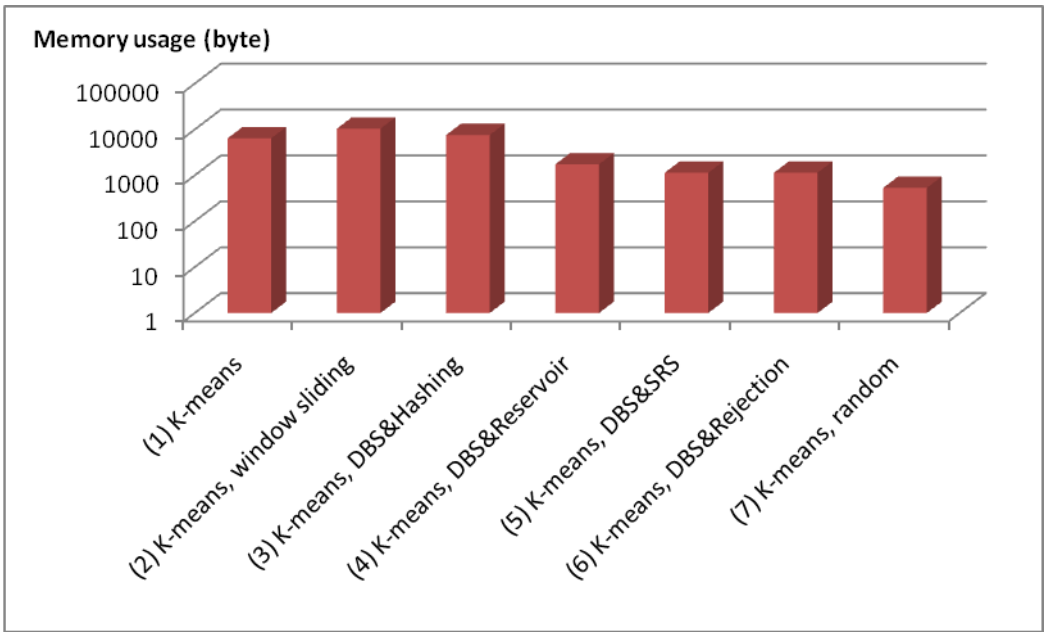


(a) ข้อมูลกระจายแบบ uniform

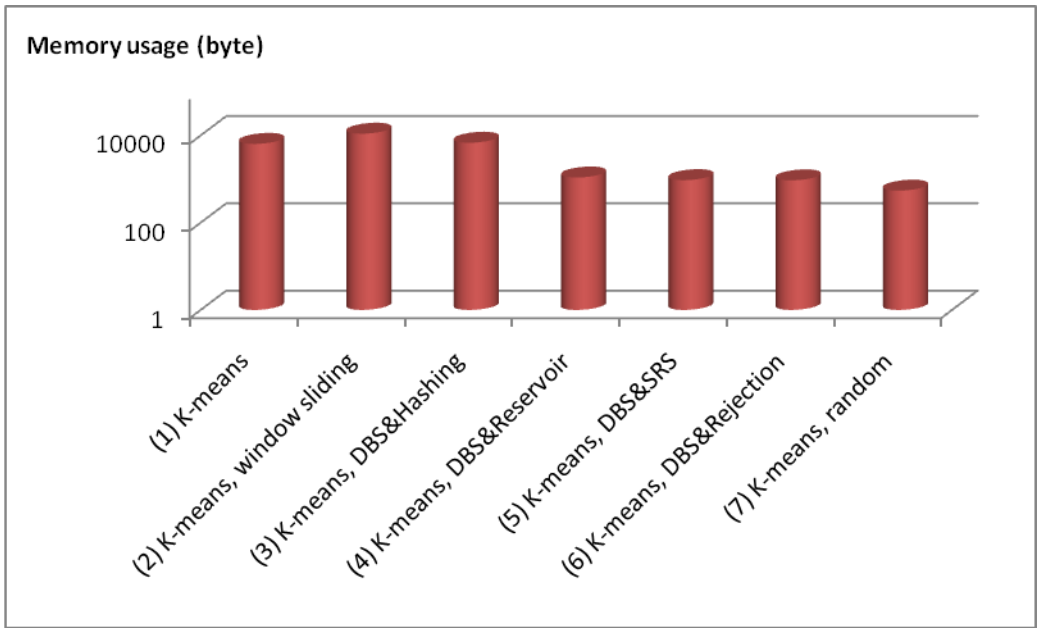


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.10 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 4 กลุ่ม

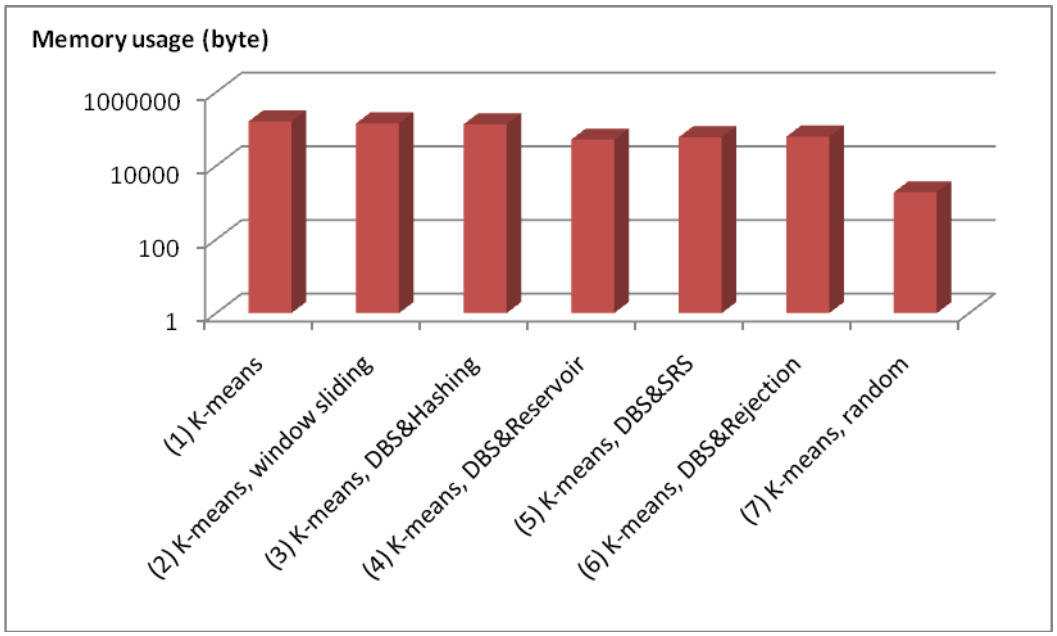


(a) ข้อมูลกระจายแบบ uniform

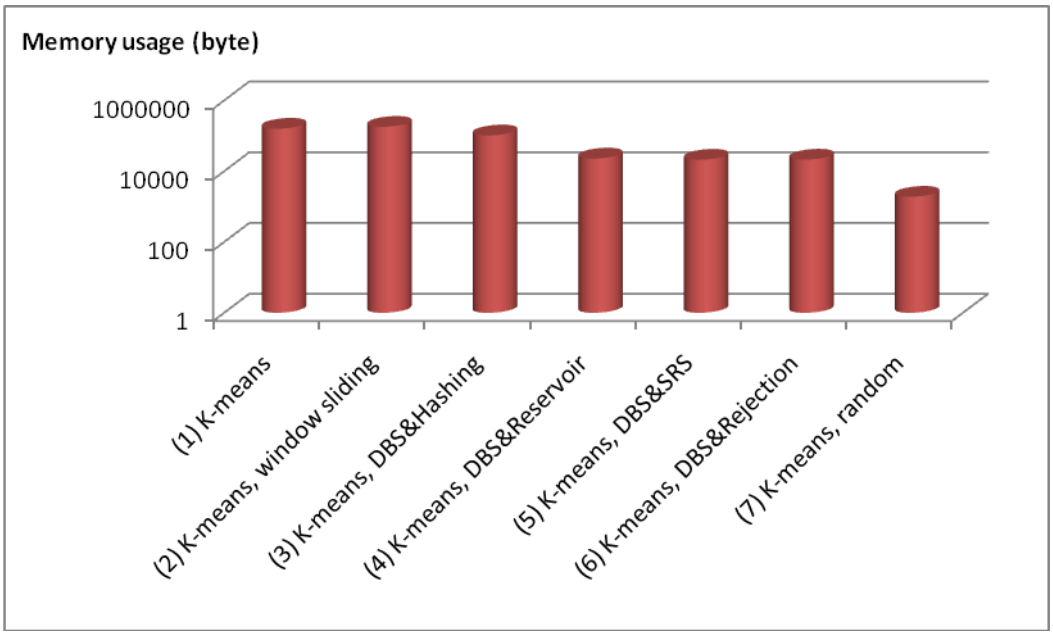


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.11 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดเล็กและมีจำนวน 8 กลุ่ม

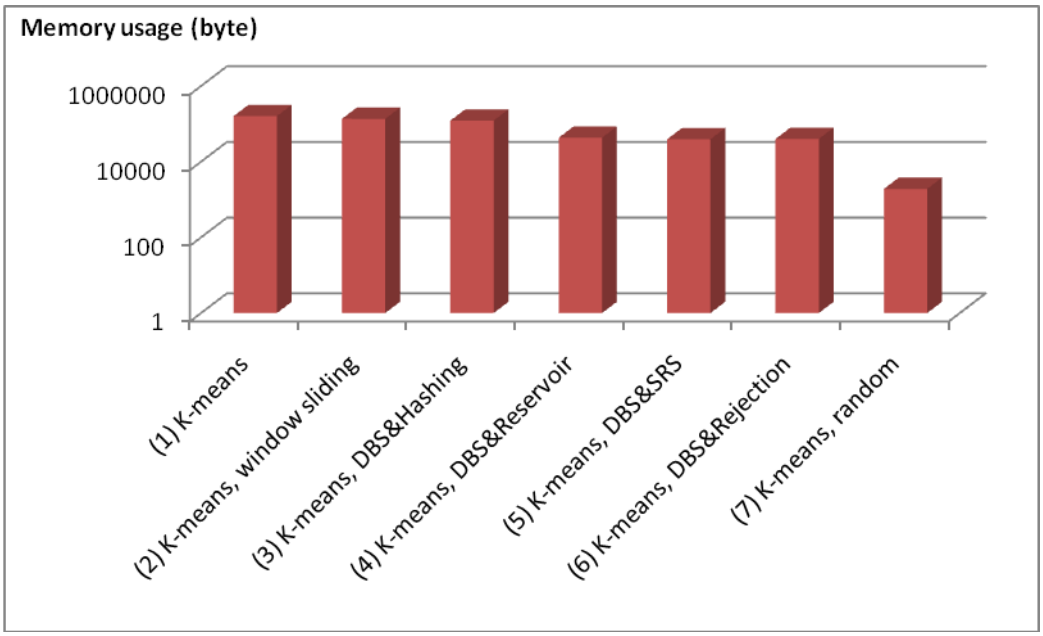


(a) ข้อมูลกระจายแบบ uniform

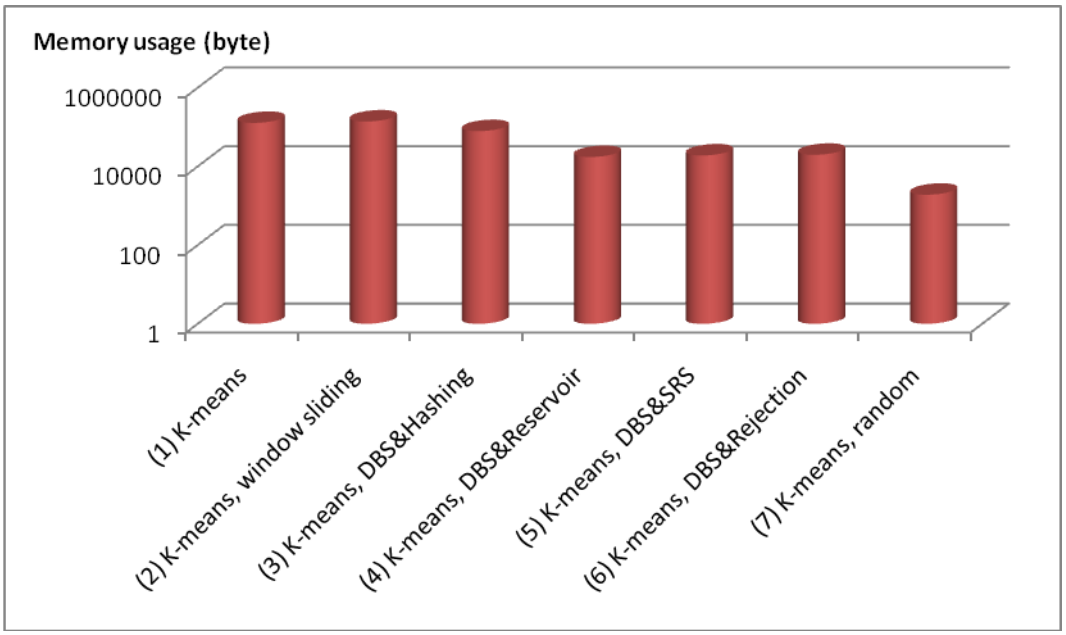


(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.12 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 4 กลุ่ม



(a) ข้อมูลกระจายแบบ uniform



(b) ข้อมูลกระจายแบบ Zipf

รูปที่ 3.13 เปรียบเทียบหน่วยความจำที่ใช้ในการจัดกลุ่มข้อมูล 3 มิติ ขนาดใหญ่และมีจำนวน 8 กลุ่ม

อภิปรายผล

จากผลการทดสอบเทคนิคการสุ่มข้อมูลและการจัดกลุ่มข้อมูล สามารถวิเคราะห์และสรุปในประเด็นที่สำคัญได้ดังนี้

1) จากการทดสอบความสามารถของเทคนิค window sliding เพื่อแปลงรูปแบบข้อมูลให้สะดวกสำหรับขั้นตอนการวัดความหนาแน่นและการสุ่ม พบว่าเมื่อแปลงข้อมูลกลับให้มีจำนวนจุดข้อมูลเท่าเดิม แล้วจัดกลุ่มด้วยโปรแกรม k-means จะได้กลุ่มที่มีความคลาดเคลื่อนของจุดกึ่งกลางประมาณ 13.292 (เฉลี่ยจากข้อมูล 8 ชุดทั้งกรณีสองมิติและสามมิติ) ซึ่งเป็นระยะทางที่ค่อนข้างน้อยเมื่อคำนึงถึงว่าข้อมูลโดยรวมมี 5,000 จุด จึงถือได้ว่าเทคนิค window sliding ทำงานได้ดีตามวัตถุประสงค์

2) ในกรณีที่ข้อมูลเป็น 2 มิติ การทำ window sliding ก่อนที่จะจัดกลุ่มข้อมูล จะลดเนื้อที่หน่วยความจำในการจัดเก็บข้อมูลลงได้มากถึงประมาณ 60% (เช่น จาก 120,000 ไบต์ ลดลงเหลือ 40,528 ไบต์) แต่เมื่อข้อมูลเพิ่มขนาดเป็น 3 มิติ เทคนิคนี้กลับใช้หน่วยความจำเพิ่มขึ้น เช่น การประมวลผล k-means กับข้อมูลสังเคราะห์ 5,000 ตัว ใช้หน่วยความจำ 160,000 ไบต์ แต่การประมวลผล k-means ร่วมกับเทคนิค window sliding ใช้เนื้อที่หน่วยความจำ 180,284 ไบต์ ดังนั้นเมื่อมิติของข้อมูลสูงขึ้นควรต้องปรับปรุงรูปแบบการจัดเก็บข้อมูลของเทคนิค window sliding

3) การสุ่มข้อมูลก่อนนำมาจัดกลุ่มข้อมูล พบว่าเมื่อกำหนดเกณฑ์ความหนาแน่น (พารามิเตอร์ density) และจำนวนเนื้อที่สำหรับจัดเก็บข้อมูลสุ่ม (พารามิเตอร์ bin) ที่แตกต่างกัน จะให้ผลการจัดกลุ่มที่มีความคลาดเคลื่อนของจุดกึ่งกลางกลุ่มมาก-น้อยต่างกัน แต่ไม่มีรูปแบบที่ชัดเจน จึงไม่สามารถกำหนดค่าพารามิเตอร์ที่เหมาะสมได้

4) ในกรณีที่ข้อมูลภายในกลุ่มมีการกระจายสม่ำเสมอและขนาดของแต่ละกลุ่ม (วัดขนาดด้วยจำนวนข้อมูลภายในกลุ่ม) แตกต่างกันไม่มากนัก พบว่าการสุ่มเลือกข้อมูลโดยตรงโดยไม่ต้องผ่านขั้นตอน window sliding แล้วนำข้อมูลนั้นมาจัดกลุ่มด้วยโปรแกรม k-means จะให้ผลลัพธ์ของการจัดกลุ่มที่ค่อนข้างดี และใช้หน่วยความจำน้อยกว่าเทคนิคการสุ่มตามความหนาแน่น (DBS) แบบต่างๆ

5) ในกรณีที่ข้อมูลมีการกระจายแบบซิฟ โดยข้อมูลแต่ละกลุ่มมีขนาดเล็ก-ใหญ่ต่างกันมาก พบว่าเทคนิคการสุ่มตามความหนาแน่น (โดยข้อมูลต้องผ่านขั้นตอน window sliding เพื่อวัดความหนาแน่น) จะให้ผลดีกว่าการสุ่มเลือกโดยตรงจากข้อมูลเริ่มต้นที่ไม่มีการวัดความหนาแน่น ในประเด็นนี้วิเคราะห์ต่อไปได้ว่าเมื่อ

ข้อมูลเป็น 2 มิติ (ทั้งข้อมูลขนาดเล็กและขนาดใหญ่) เทคนิค DBS เมื่อใช้ร่วมกับ rejection sampling จะให้ผลการจัดกลุ่มที่ดี แต่เมื่อข้อมูลมีมิติเพิ่มขึ้นเป็น 3 มิติ (ทั้งข้อมูลขนาดเล็กและขนาดใหญ่) เทคนิค DBS เมื่อใช้ร่วมกับ simple random sampling จะให้ผลการจัดกลุ่มที่ดีกว่าเล็กน้อย

6) เมื่อใช้เทคนิค DBS ร่วมกับ rejection sampling ที่ใช้การสุ่มสองครั้ง ครั้งแรกสุ่มเลือกข้อมูลและครั้งที่สองสุ่มค่า uniform ที่มีค่าระหว่าง [0..1] เพื่อตัดสินใจว่าจะใช้ข้อมูลสุ่มนั้นหรือไม่ พบว่าการกำหนดช่วงของค่า uniform (พารามิเตอร์ range) ที่แตกต่างกันอาจให้ผลการจัดกลุ่มที่แตกต่างกันบ้าง แต่ไม่มีรูปแบบที่แน่นอน และความแตกต่างไม่มากพอที่จะมีนัยสำคัญต่อขั้นตอนการจัดกลุ่มข้อมูล

7) ในการวิเคราะห์เวลาที่ใช้ในการประมวลผล จะรวมเวลาทั้งในช่วงการสุ่มและเวลาในการจัดกลุ่มข้อมูล จากการทดลองพบว่าเมื่อข้อมูลมีขนาดเล็ก เวลาที่ใช้ในการสุ่มจะมากกว่าเวลาที่ใช้ในขั้นตอนการจัดกลุ่มมาก แต่เมื่อข้อมูลมีขนาดใหญ่เวลาที่ใช้ในช่วงของการสุ่มจะไม่เพิ่มขึ้นมากเมื่อเปรียบเทียบกับเมื่อข้อมูลมีขนาดเล็ก แต่ในข้อมูลขนาดใหญ่ โดยเฉพาะเมื่อข้อมูลมีจำนวนกลุ่มมาก ขั้นตอนการจัดกลุ่มจะใช้เวลาเพิ่มมากขึ้นอย่างเห็นได้ชัด ซึ่งข้อสังเกตนี้สอดคล้องกับลักษณะของอัลกอริทึม k-means ที่จะใช้เวลามากขึ้นเมื่อจำนวนข้อมูล จำนวนกลุ่ม และจำนวนรอบการวนซ้ำ เพิ่มสูงขึ้น

บทที่ 4
บทสรุป

สรุปผลการวิจัย

การจัดกลุ่มข้อมูล เป็นงานวิเคราะห์ข้อมูลอัตโนมัติที่มีวัตถุประสงค์ที่จะคำนวณความคล้ายคลึงของข้อมูลเพื่อจัดข้อมูลที่คล้ายกันให้อยู่ในกลุ่มย่อยเดียวกัน เมื่อได้กลุ่มย่อยแล้วมักจะมีการหาลักษณะเด่นภายในกลุ่มข้อมูลหรือตัวแทนกลุ่ม ลักษณะของตัวแทนกลุ่มนี้จะเป็นความรู้ที่สามารถนำไปใช้ประโยชน์ในการจำแนกหรือจัดกลุ่มข้อมูลใหม่ที่จะเกิดขึ้นในอนาคต การจัดกลุ่มข้อมูลโดยอัตโนมัติจัดเป็นการเรียนรู้แบบที่ไม่ต้องใช้การชี้แนะ (unsupervised learning) นั่นคือผู้ใช้ไม่ต้องให้ข้อมูลตัวอย่างว่าข้อมูลแบบใดควรจะจัดอยู่ในกลุ่มใด ผู้ใช้เพียงแต่ระบุว่าต้องการให้จัดข้อมูลเป็นกี่กลุ่ม จากนั้นโปรแกรม (ในที่นี้หมายถึงโปรแกรม k-means) จะทำการวิเคราะห์ข้อมูลวัดระยะห่างของข้อมูล และสุดท้ายจัดข้อมูลทุกตัวเข้ากลุ่มให้ได้จำนวนกลุ่มตามที่ผู้ใช้ต้องการ กระบวนการจัดกลุ่มเป็นงานที่ต้องทำซ้ำหลายรอบ สรุปขั้นตอนการทำงานได้ดังนี้

Step 1: User specifies *k* = number of clusters to partition data

Step 2: *k* records are randomly assigned to be initial cluster centers

Step 3: For each record, assign to the nearest cluster

Each cluster center “owns” subset of records

Results in *k* clusters, C*1*, C*2*, ..., C*k*

Step 4: For each of *k* clusters, find cluster <u>centroid</u>

Update cluster center location to centroid

Step 5: Repeats Steps 3 – 5 until convergence or termination

การวัดระยะห่างระหว่างข้อมูลและจุดกึ่งกลางกลุ่ม (centroids) ในขั้นตอนที่ 3 เพื่อจัดข้อมูลเข้ากลุ่มที่อยู่ใกล้ที่สุด สามารถใช้เกณฑ์การคำนวณได้หลายรูปแบบ เช่นใช้วิธีการวัดแบบยูคลิเดียน การวัดแบบซิตี้บล็อก การวัดแบบมินคอฟสกี ถ้ากำหนดข้อมูลเป็นจุด X, Y โดย X = $<x_1, x_2, \dots, x_m>$ และ Y = $<y_1, y_2, \dots, y_m>$ เมื่อ *m* คือมิติของข้อมูล วิธีการวัดระยะห่างแต่ละแบบมีวิธีการคำนวณดังนี้

$$d_{Euclidean}(X,Y) = \sqrt{\sum_i (x_i - y_i)^2}$$

$$d_{city-block}(X,Y) = \sum_i |x_i - y_i|$$

$$d_{Minkowski}(X,Y) = \sum_i |x_i - y_i|^q$$

ในงานวิจัยนี้ใช้วิธีการคำนวณแบบยูคลิเดียนเนื่องจากเมื่อข้อมูลมีหลายมิติ จะทำให้ได้ผลลัพธ์เป็นระยะห่างที่ใกล้เคียงระยะทางจริงมากที่สุด เมื่อจัดข้อมูลทุกตัวเข้ากลุ่มได้แล้ว ในขั้นตอนที่ 4 จะต้องมีการคำนวณจุดกึ่งกลางกลุ่มที่เรียกว่าเซ็นทรอยด์ (centroid) การคำนวณจุดกึ่งกลางกลุ่มจะใช้วิธีคำนวณค่าเฉลี่ยในแต่ละมิติของข้อมูลทุกตัวในกลุ่ม วิธีการคำนวณจุดกึ่งกลางของกลุ่ม j เมื่อมีข้อมูลในกลุ่ม n ตัว ($X_1, \ldots, X_n$) และข้อมูลแต่ละตัวมี m มิติ, $X_{i,d}$ = $<x_{i,1}, x_{i,2}, \ldots, x_{i,m}>$, แสดงได้ดังนี้

$$C_j = < \frac{\sum_{i=1}^{n} X_{i,1}}{n}, \ldots, \frac{\sum_{i=1}^{n} X_{i,m}}{n} >$$

จากลักษณะของการเรียนรู้แบบ unsupervised learning เวลาที่ใช้ในการทำงานจึงมักจะสูงกว่าการเรียนรู้ในแบบ supervised learning เวลาที่ใช้นี้จะผันแปรตามจำนวนมิติของข้อมูลและปริมาณของข้อมูล เมื่อข้อมูลมีปริมาณสูงมากอัลกอริทึมการจัดกลุ่มข้อมูลจะทำงานได้ช้าลง หรือถ้าข้อมูลมีปริมาณมากจนเกินไป จะไม่สามารถทำงานได้เนื่องจากปริมาณข้อมูลมีมากเกินความจุของหน่วยความจำคอมพิวเตอร์ ถ้าต้องการปรับปรุงอัลกอริทึมจัดกลุ่มข้อมูลให้สามารถทำงานกับข้อมูลปริมาณสูงมากได้จะต้องใช้วิธีลดขนาดของข้อมูล

งานวิจัยนี้ศึกษาวิธีการลดขนาดของข้อมูลด้วยเทคนิคการสุ่ม โดยคำนึงถึงลักษณะการกระจายของข้อมูล ถ้าข้อมูลมีการกระจายอย่างสม่ำเสมอการสุ่มข้อมูลจะให้น้ำหนักข้อมูลทุกตัวเท่าเทียมกัน แต่ถ้าข้อมูลมีการกระจายไม่สม่ำเสมอเช่นมีการกระจายแบบซิฟ ข้อมูลในบางช่วงมีความหนาแน่นมากกว่าข้อมูลในช่วงอื่นๆด้วยสัดส่วนความหนาแน่นที่แตกต่างกันมาก การสุ่มข้อมูลจะเพิ่มโอกาสการถูกสุ่มให้กับข้อมูลในช่วงที่มีความหนาแน่นสูง และเรียกเทคนิคการสุ่มแบบนี้ว่าการสุ่มตามความหนาแน่น (density-biased sampling, DBS) โดยการวัดความหนาแน่นของข้อมูลจะใช้เทคนิคการเลื่อนกรอบหน้าต่าง (window sliding) ซึ่งเป็นกรอบขนาดเล็กที่เคลื่อนไปบนแกนข้อมูลและวัดจำนวนจุดข้อมูลที่ปรากฏภายในแต่ละกรอบ เพื่อบันทึกเป็นค่าความหนาแน่นของกรอบหน้าต่าง การสุ่มตามความหนาแน่นจึงเป็นการคัดเลือกกรอบหน้าต่างที่มีจำนวนจุดภายในกรอบตรงตามเกณฑ์ที่กำหนด จากนั้นใช้พิกัดของกรอบหน้าต่างเหล่านี้สร้างกลับเป็นจุดข้อมูลที่มีจำนวนจุดเท่ากับค่าความหนาแน่นของกรอบหน้าต่าง

โครงการวิจัยนี้ได้ทดสอบการจัดกลุ่มข้อมูลกับข้อมูลที่ผ่านกระบวนการสุ่ม ทั้งที่เป็นการสุ่มอย่างง่ายที่ไม่มีการพิจารณาความหนาแน่นของข้อมูล และการสุ่มตามความหนาแน่นของข้อมูล ในการทดสอบได้ใช้ข้อมูลสังเคราะห์รวมทั้งหมด 16 ชุด ข้อมูลแปดชุดแรกเป็นข้อมูลสองมิติ และข้อมูลอีกแปดชุดเป็นข้อมูลสามมิติ ในข้อมูลแปดชุดแบ่งย่อยออกเป็นข้อมูลที่มีการกระจาย

อย่างสม่ำเสมอ (uniform distribution) จำนวนสี่ชุดที่มีทั้งข้อมูลขนาดเล็กและข้อมูลขนาดใหญ่ และข้อมูลที่มีการกระจายแบบซิฟ (Zipf distribution) จำนวนสี่ชุดที่มีทั้งขนาดเล็กและใหญ่เช่นเดียวกัน เทคนิคการสุ่มข้อมูลประกอบด้วย

(1) Density-biased sampling: hashing (unlimited number of bins)

(2) Density-biased sampling: reservoir and hashing

(3) Density-biased sampling: reservoir with simple random sampling

(4) Density-biased sampling: reservoir with rejection sampling

(5) Simple random sampling on the original data set (no data transformation with window sliding technique)

ผลการทดสอบพบว่าในข้อมูลที่มีการกระจายสม่ำเสมอ เทคนิคที่ (2) และ (5) ให้ผลการจัดกลุ่มที่ค่อนข้างดี กลุ่มข้อมูลที่ได้คลาดเคลื่อนไปจากกลุ่มที่แท้จริงเพียงเล็กน้อย แต่มีข้อสังเกตว่าเทคนิคที่ (5) ที่เป็นการสุ่มเลือกข้อมูลแล้วนำข้อมูลนั้นไปจัดกลุ่มด้วยโปรแกรม k-means จะใช้เนื้อที่หน่วยความจำน้อยมากเมื่อเทียบกับเทคนิคอื่นๆ ดังนั้นจึงอาจสรุปได้ว่าเมื่อข้อมูลมีการกระจายสม่ำเสมอ ไม่จำเป็นต้องทำ Density-biased sampling เพราะเทคนิคนี้จะใช้เวลาและหน่วยความจำมากกว่าการสุ่มข้อมูลแบบปกติ และผลการจัดกลุ่มที่ได้จะไม่ดีขึ้นมากอย่างมีนัยสำคัญ

แต่ในกรณีที่ข้อมูลมีการกระจายแบบซิฟ นั่นคือข้อมูลในแต่ละกลุ่มมีขนาดเล็ก-ใหญ่ต่างกันมาก และปริมาณข้อมูลในกลุ่มหรือความหนาแน่นก็แตกต่างกันมากเช่นเดียวกัน กรณีเช่นนี้พบว่าการสุ่มตามความหนาแน่น หรือ Density-biased sampling จะมีผลช่วยให้ข้อมูลที่คัดเลือกมาผ่านกระบวนการจัดกลุ่ม ได้ข้อมูลตัวแทนที่ดี ความหนาแน่นของข้อมูลที่ถูกคัดเลือกใกล้เคียงกับความหนาแน่นที่ปรากฏในกลุ่มข้อมูลดั้งเดิมที่สังเคราะห์ขึ้น เทคนิคการสุ่มที่ทำงานได้ดีกับข้อมูลที่กระจายแบบซิฟ คือเทคนิคที่ (3) และ (4) นั่นคือเทคนิคที่มีการวัดความหนาแน่นข้อมูลด้วยการทำ window sliding จากนั้นคัดเลือกข้อมูลด้วยเกณฑ์ความหนาแน่นขั้นต่ำ จากนั้นนำกรอบหน้าต่างที่มีความหนาแน่นถึงเกณฑ์แปลงข้อมูลกลับ ให้ได้จำนวนจุดข้อมูลเท่ากับค่าความหนาแน่นของกรอบหน้าต่าง ขั้นตอนต่อจากนั้นเป็นการสุ่มเลือกข้อมูลให้มีปริมาณเท่ากับเนื้อที่ reservoir โดยการสุ่มในขั้นนี้อาจจะเป็นการสุ่มแบบ simple random sampling หรืออาจจะใช้วิธี rejection sampling ก็ได้ ซึ่งจากผลการทดลองพบว่าให้ผลลัพธ์ที่ใกล้เคียงกัน แต่วิธี simple random sampling จะใช้เนื้อที่หน่วยความจำ และใช้เวลาที่น้อยกว่าเล็กน้อย

ข้อเสนอแนะ

การทดลองในโครงการวิจัยนี้ ได้ทดสอบความสามารถของเทคนิคการสุ่มข้อมูลตามความหนาแน่นในข้อมูลสองและสามมิติ ผลการทดลองเบื้องต้นพบว่าเมื่อมิติของข้อมูลสูงขึ้น การทำ window sliding จะใช้หน่วยความจำเพิ่มขึ้นในปริมาณที่สูงมาก การพัฒนาเทคนิคการสุ่มตามความหนาแน่นในอนาคตจึงควรต้องปรับปรุงวิธีการทำ window sliding ให้มีลักษณะบีบอัดข้อมูลมากขึ้นเพื่อลดปริมาณการใช้เนื้อที่หน่วยความจำให้น้อยลง หรืออาจจะปรับปรุงเทคนิค window sliding ให้มีขนาดกรอบหน้าต่างที่แปรผันได้ (adaptive window sliding) โดยช่วงที่มีข้อมูลเบาบางอาจขยายขนาดกรอบหน้าต่าง แต่ช่วงที่ข้อมูลหนาแน่นให้ลดขนาดของกรอบหน้าต่าง เพื่อให้ใช้เนื้อที่หน่วยความจำได้อย่างมีประสิทธิภาพสูงสุด

นอกจากนี้ในการทดลองได้กำหนดขนาดของกรอบหน้าต่างให้มีขนาดคงที่ที่ 2x2 จากการทดลองแปลงข้อมูลกลับแล้วจัดกลุ่มข้อมูล (k-means with window sliding) ผลการจัดกลุ่มข้อมูลที่ได้ให้ผลลัพธ์ที่ดีมาก จุดกึ่งกลางกลุ่มเคลื่อนไปเป็นระยะทางค่อนข้างน้อย จากผลการทดลองที่ได้ทำให้คาดหมายได้ว่าเมื่อมีการเปลี่ยนขนาดกรอบหน้าต่างเป็น 3x3, 4x4 หรือขนาดที่กว้างกว่านี้ น่าจะให้ผลการจัดกลุ่มข้อมูลที่แตกต่างไปจากที่นำเสนอในโครงการวิจัยนี้

ในกรณีที่ข้อมูลมีการกระจายแบบซิฟและข้อมูลมีมิติสูงมาก ผลการทดสอบข้อมูลของโครงการวิจัยนี้บ่งชี้ว่าถึงแม้ว่าจะใช้เทคนิคการสุ่มตามความหนาแน่น ผลการจัดกลุ่มที่ได้ก็จะคลาดเคลื่อนมากขึ้นกว่าข้อมูลที่มีมิติต่ำ ดังนั้นในกรณีที่ข้อมูลมีมิติสูง เช่น ข้อมูลรหัสพันธุกรรม ควรมีการปรับลดมิติข้อมูลด้วยเทคนิค feature selection (เช่น ใช้วิธี support vector machine หรือวิเคราะห์หามิติหลักด้วยเทคนิค principal component analysis) ก่อนที่จะเริ่มกระบวนการสุ่มตามความหนาแน่นเพื่อลดจำนวนข้อมูล

ซอฟต์แวร์ที่พัฒนาขึ้นนี้ใช้ภาษาเออแลง (Erlang) ที่เขียนในลักษณะ sequential เพื่อให้โปรแกรมอ่านง่ายและผู้ที่สนใจสามารถปรับปรุงแก้ไขคำสั่งได้ง่าย แต่โดยความสามารถของภาษานี้ที่อำนวยความสะดวกในด้าน concurrent programming ด้วยเทคนิคของการส่งเมสเสจติดต่อระหว่างโพรเซส การปรับปรุงวิธีการเขียนคำสั่งให้เป็นลักษณะ concurrent จะช่วยให้โปรแกรมทำงานแบบพร้อมกันหลายโพรเซส และถ้าหน่วยประมวลผลป็นแบบหลายโพรเซสเซอร์ (multi-core) การเปลี่ยนวิธีการเขียนโปรแกรมอาจจะทำให้ผลการทดลองเปลี่ยนไปจากที่ได้นำเสนอในโครงการวิจัยนี้ และแนวทางนี้เป็นแนวทางที่ผู้วิจัยอยู่ในระหว่างพัฒนาและคาดว่าจะนำเสนอผลการทดลองได้ในอนาคต

บรรณานุกรม

M. R. Anderberg, *Cluster Analysis for Applications* (New York: Academic Press, 1973).

C. Faloutsos, Y. Matias, & A. Silberschatz, Modeling skewed distributions using multifractals and the '80-20 law', *The VLDB Journal*, September 1996.

S. Guha, R. Rastogi, & K. Shim, Cure: An efficient clustering algorithm for large databases, *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, 1998, 73-84.

A.K. Jain & R.C. Dubes, *Algorithms for Clustering Data* (Englewood Cliffs, New Jersey: Prentice Hall, 1988).

G. Karypis, E.-H. Han, & V. Kumar, Chameleon: A hierarchical clustering algorithm using dynamic modeling, *IEEE Computer*, 32(8), 1999, 68-75.

G. Kollios, D. Gunopulos, N. Koudas, & S. Berchtold, Efficient biased sampling for approximate clustering and outlier detection in data sets, *IEEE Transactions on Knowledge and Data Engineering,* 15(5), 2003, 1170-1187.

J.B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probabilisty*, Vol.1, 1967, 281-297.

C. Palmer & C. Faloutsos, Density biased sampling: An improve method for data mining and clustering, *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, 2000, 82-92.

R.J. Roiger & M.W. Geatz, *Data Mining: A Tutorial Based Primer* (Boston: Addison Wesley, 2003).

J. Sander, M. Ester, H.-P. Kriegel, & X. Xu, Density-based clustering in spatial databases: The algorithm gdbscan and its application, *Data Mining and Knowledge Discovery*, 2(2), 1998, 169-194.

M. Schroeder, *Fractals, chaos, power laws: Minutes from an infinite paradise* (New York: W.H. Freeman and Company, 1991).

A. Stehl, J. Ghosh, & R. Mooney, Impact of similarity measures on web-page clustering, *Proceedings of the Workshop of Artificial Intelligence for Web Search, 17th National Conference on Artificial Intelligence*, 2000.

J.S. Vitter, Random sampling with a reservoir, *ACM Transaction on Mathematical Software*, 11(1), 1985, 37-57.

G.K. Zipf, *Human behavior and principle of least effort: An introduction to human ecology* (Cambridge, MA : Addison Wesley, 1949).

ภาคผนวก

ภาคผนวก ก

รหัสต้นฉบับของโปรแกรมการสุ่มตามความหนาแน่นเพื่อการจัดกลุ่มข้อมูล

```
% ====================================================================
% Program: Density-biased sampling and K-means clustering
%
%   By: Kittisak Kerdprasop
%       School of Computer Engineering,
%       Suranaree University of Technology, Thailand
%
%   Date: May 5, 2010
%
%   Comments: This is an open source program, freely distributed as is.
%             The code is in Erlang format (R13B04).
%   Usage:
%   (1) Install Erlang (downloadable from www.erlang.org).
%   (2) Set path to the directory that contains Erlang code.
%        (This step can be done through the properties of Erlang icon.)
%   (3) Start the Erlang shell.
%   (4) Compile a program with the command:
%                    >c(dbs).
%   (5) Run a program with the command (dbs = density-biased sampling):
%                    >dbs:callMenu().
%
% ----------- view points in visual mode uses gnuplot
%               2 dimensions:  gnuplot> plot 'points.dat'
%               3 dimensions:  gnuplot> splot 'points.dat'
% ====================================================================
-module(dbs).
-export([myMain/0,clustering/3,eachWindow/3,specificDens/2,
         rejectionRandom/4,simpleRandom/3,simpleRandom1/2,
         specificDensBin/3
         ]).
-import(io,[format/1,format/2,format/3,read/1,fwrite/3]).

% --------------- generate data points in each cluster
%    User specifies number of dimensions:
%                  let's say 3
%    Example of data point specification format:
%                  [{0,3,7},{5,10,3},{20,30,5}]
%    This is a format to generate data of 4 clusters
%       cluster 1: number of data points = 7
%                   range in the x-axis is between 0-3
%                   range in the y-axis is between 0-3
%                   range in the z-axis, which is the last dimension,
%                                        is unrestricted
%       cluster 2: number of data points = 3
%                   range in the x-axis is between 5-10
%                   range in the y-axis is between 5-10
%                   range in the z-axis, which is the last dimension,
%                                        is unrestricted
%       cluster 3: number of data points = 5
%                   range in the x-axis is between 20-30
%                   range in the y-axis is between 20-30
%                   range in the z-axis, which is the last dimension,
%                                        is unrestricted
%   Note that this program sets scale on each axis to range from
%             0 to 50, and maximum dimension is 10.
%  ……
```

```
gen(Rand,{1,L,H})->[random:uniform(H-L)+Rand ];
gen(Rand,{K,L,H})->[random:uniform(H-L)+L|gen(Rand,{K-1,L,H})].

% K=Dimension  --Seed for last dimension
myDim(_,_,{_,_,0})->[];
myDim(Rand,K,{L,H,N})->[gen(Rand,{K,L,H})|myDim(Rand,K,{L,H,N-1})].

myGenGroup(K,InitGroup) ->
    lists:concat(
            lists:map(
                fun(A)->Seed=random:uniform(40),
                        format("~nSeed~w",[Seed]),
                        myDim(Seed,K,A) end,
                InitGroup)
             ).
%------------k-means clustering
% nearCentroid([1],[[2],[3],[45],[1]]). ---> [[1],[1]]
nearCentroid(Point,CentroidL)->
      LenList=lists:zip(
      lists:map
             (fun(A)->distance(Point,A)
                    end,
                    CentroidL
                  ), CentroidL),
          [{_,Centroid}|_]=lists:keysort(1,LenList),
          {Point,Centroid}.

% take firts  distinct-n element of list
take(0,_)->[];
take(N,[H|T])->[H|take(N-1,T)].

% for K-means clustering
kMeans(PL)->
   {_,N}=read('enter number of clustering>'),
   CL=take(N,sets:to_list(sets:from_list(PL))), %initial Centroid
   format("~nAllPoints=~w ~nInitial Centroid=~w~n",[PL,CL]),
   {TT,L}=timer:tc(dbs,clustering,[1,CL,PL]),T11=TT/1000000,
                  % return time in second.
   format("~n~n__Time for k-means is ~w second ,~n
           calculated Centroid=~w~n~n",[T11,L]), %write to screen
   L.  %return cluster

clustering(N,CL,PL)->
      L1=lists:map( fun(A) -> nearCentroid(A,CL) end ,PL),
      L2 = transform(CL,L1),
      NewCentroid=lists:map(fun({_,GL})->findMeans(GL) end,L2),
      N1=N+1,
      if NewCentroid==CL -> format("No cluster changes,
                                    From Loop1->stop at Loop~w,
                                    Centroid=~w",[N,NewCentroid]),
                            NewCentroid;
           N>=90 -> format("Force to stop at Loop~w,
                            Centroid=~w",[N,NewCentroid]),
                     NewCentroid; % Max iterations=90
           true -> p("NLoop",N),p("NewCentroid",NewCentroid),
                   clustering(N1,NewCentroid,PL)
      end.
```

```
p(D,V)->format("~n~s=~w~n",[D,V]). % for printing values

% transform Point-CentroidList to Centroid-PointList
% transform([[1]],[{[2],[1]},{[3],[1]}]). -->[{[1],[[2],[3]]} ]
transform([],_)->[];
transform([C|TC],PC)->[{C,t1(C,PC)} |transform(TC,PC)].

t1(_,[])->[] ;
t1(C1,[H|T])->{P,C}=H,
      if C1==C -> [P|t1(C1,T)];
      C1=/=C -> t1(C1,T)
      end.

distance([],[])-> 0;
distance([X1|T1],[X2|T2])->
            math:sqrt((X2-X1)*(X2-X1)+distance(T1,T2) ).

% findMeans([[1,2],[3,4]]). --> [2.0,3.0]
findMeans(PointL) ->
      [H|_]=PointL ,
      Len=length(H),
      AllDim=lists:reverse(allDim(Len,PointL)),
      lists:map(fun(A)-> mymean(A) end, AllDim ).

eachDimList(_,[]) ->[];
eachDimList(N,[H|T]) ->[lists:nth(N,H)|eachDimList(N,T)].

allDim(0,_)->[];
allDim(D,L)->[eachDimList(D,L)|allDim(D-1,L)].

mymean(L)->lists:sum(L)/length(L).

% eachWindow(2,[[0,1],[1,0]],[[0,0],[0,2]]).
% ---> [{[0,0],2},{[0,2],0}]

eachWindow(_,_,[])->[];
eachWindow(Del,AllPoint,[Now|StreamT])->
            Lc = count(Now,Del,AllPoint),
            Sum= lists:sum(Lc),
            [{Now,Sum}| eachWindow(Del,AllPoint,StreamT)].

% gen back slidingWindows -> POINTs
%   + WindowsList,+Den

genWin2P([])->[];
genWin2P([{P,Den}|T])->
        if Den>=1 -> dup(center(P),Den)++genWin2P(T);
             true -> genWin2P(T)
          end.

center([])->[];
center([H|T])->[H+1|center(T)].

dup(_,0)->[];
dup(L,N)->[L|dup(L,N-1)] .
```

```
%------------ sum of euclidean distanct
dis(P,C) -> ZipPC= lists:zip(P,C),
            math:sqrt(lists:sum([(X-X1)*(X-X1) ||{X,X1} <- ZipPC])).

minDistance(P,CL)-> lists:min([dis(P,Y)|| Y<-CL]).

sumDistance(PL,CL)->lists:sum([minDistance(X,CL)||X<-PL]).
%------------
count(_,_,[])->[];
count(L,Del,[H|T]) ->In=in(H,L,[X+Del||X<-L]),
            if In ->[1|count(L,Del,T)];
                  true->[0|count(L,Del,T)]
            end.

  % in([2,3],[0,0],[2,2]) --> true
in([],_,_) -> true;
in([X|T],[L|TL],[U|TU]) ->
      if L =< X,X <U -> in(T,TL,TU);
      true ->false
      end.

% print Points to text file for gnuplot
printFile(OutF,Dim,[H|T])->
  case Dim of
   2-> fwrite(OutF,"~n~w ~w",H),printFile(OutF,Dim,T);
   3-> fwrite(OutF,"~n~w ~w ~w",H),printFile(OutF,Dim,T);
   _-> fwrite(OutF,"~n%No output file/Error Dimension ~w > 3",[Dim]),
       true
   end ;

printFile(_OutF,_Dim,[])->true.

      % ---- Generate data points
myGenL(50)->[];  % 50 = Upper bound on axis scaling, lower bound = 0
myGenL(Now)->[Now|myGenL(Now+2)]. % 2 = window step size

    % number of dimensions is bounded at 10
myZip(1)-> L=myGenL(0), [[X1]||X1<-L];
myZip(2)-> L=myGenL(0), [[X1,X2]||X1<-L,X2<-L];
myZip(3)-> L=myGenL(0),
 [[X1,X2,X3]||X1<-L,X2<-L,X3<-L];
myZip(4)-> L=myGenL(0),
 [[X1,X2,X3,X4]||X1<-L,X2<-L,X3<-L,X4<-L];
myZip(5)-> L=myGenL(0),
 [[X1,X2,X3,X4,X5]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L];
myZip(6)-> L=myGenL(0),
 [[X1,X2,X3,X4,X5,X6]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L,X6<-L];
myZip(7)-> L=myGenL(0),
 [[X1,X2,X3,X4,X5,X6,X7]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L,X6<-L,X7<-L];
myZip(8)-> L=myGenL(0),
 [[X1,X2,X3,X4,X5,X6,X7,X8]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L,X6<-L,X7<-
L,X8<-L];
myZip(9)-> L=myGenL(0),
 [[X1,X2,X3,X4,X5,X6,X7,X8,X9]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L,X6<-
L,X7<-L,X8<-L,X9<-L];
myZip(10)-> L=myGenL(0),
```

```
 [[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10]||X1<-L,X2<-L,X3<-L,X4<-L,X5<-L,X6<-
L,X7<-L,X8<-L,X9<-L,X10<-L].
% --------------------

specificDens([],_)->[];
specificDens([{P,D}|T],Den)->
  if D>=Den -> [{P,D}|specificDens(T,Den)];
      true -> specificDens(T,Den)
      end.

% take Bin elements
specificDensBin(Bin,L,Den) -> take(Bin,specificDens(L,Den)).

%--- Simple Random with density bias (no parameter SAMPLING)
simpleRandom(_,_,0)->[];
simpleRandom(WindowL,Dens,Bin)->
    Nth=random:uniform(length(WindowL)),
    {P,D}=lists:nth(Nth,WindowL),
    if D>=Dens -> [{P,D}|simpleRandom(WindowL,Dens,Bin-1)]  ;
      true-> simpleRandom(WindowL,Dens,Bin)
    end.

simpleRandom1(_,0)->[];
simpleRandom1(AllP,Bin)->
    Nth=random:uniform(length(AllP)),
    P=lists:nth(Nth,AllP),
    [P | simpleRandom1(AllP,Bin-1)].

rejectionRandom(_,_,_,0)->[];
rejectionRandom(Para,WindowL,Dens,Bin)->
    Nth=random:uniform(length(WindowL)),
    Par=random:uniform(),
    {P,D}=lists:nth(Nth,WindowL),
    if (0.5-Para)=<Par,Par=<(0.5+Para),D>=Dens ->
      [{P,D}|rejectionRandom(Para,WindowL,Dens,Bin-1)]  ;
      true-> rejectionRandom(Para,WindowL,Dens,Bin)
    end.
 % ---------------------

callWindowSliding(Dim,AllP)-> % initialization
   Stream=myZip(Dim),
            % window sliding  -- running time computation
   {TT,L}=timer:tc(dbs,eachWindow,[Dim,AllP,Stream]),T11=TT/1000000,
            % return in second.
   format("~nTime for window sliding of dimension ~w is
            ~w second ~n",[Dim,T11]),   % write to screen
   L .      % and return Windows List

callDensity (WindowList,Dens)-> % menu 3
   {T,Ld}=timer:tc(dbs,specificDens,[WindowList,Dens]),Len=length(Ld),
                       T1=T/1000000,    % time returns in second.
   format("~nFrom MENU3 has time ~w seconds,
             Length(MAX BIN)=~w~n",[T1,Len]),
   Ld .

callDensityRes (WindowList,Dens)-> % menu 4
   {_,Bin}=read('enter number of Bin>'),
```

```
   {T,Ld}=timer:tc(dbs,specificDensBin,[Bin,WindowList,Dens]),
    T1=T/1000000, % return in second.
    format("~nFrom DensityReservoir(MENU4) has time ~w
             seconds,Length=~w~n",[T1,length(Ld)]),
   Ld.

callSimpleRand1(AllP) -> % menu 7
    {_,Bin}=read('enter number of Bin>'),
    {T,RandList}=timer:tc(dbs,simpleRandom1,[AllP,Bin]),
    T1=T/1000000,  % return in second.
    format("~nFrom MENU7 simpleRandom has time ~w seconds,
             length=~w~n",[T1,length(RandList)]),
    RandList.

callSimpleRand(WindowList,Dens) -> % menu 5
  {_,Bin}=read('enter number of Bin>'),
  WindowL=sets:to_list(sets:from_list(WindowList)), % to distinct List
  {T,RandList}=timer:tc(dbs,simpleRandom,[WindowL,Dens,Bin]),
  T1=T/1000000,  % return in second.
  format("~nFrom MENU5 simpleRandom has time ~w seconds,length=~w~n",
           [T1,length(RandList)]),
  RandList.

callReservoir(WindowList,Dens)->  % menu 6
   {_,Bin}=read('enter number of Bin>'),
   WindowL=sets:to_list(sets:from_list(WindowList)), % to distinct List
   {_,Para}=read('enter rejection Parameter0.5+->'),
   {T,RandList}=timer:tc(dbs,rejectionRandom,[Para,WindowL,Dens,Bin]),
                      T1=T/1000000,   % return in second.
   format("~nTime from MENU6 Reservoir with density ~w  =
            ~w seconds ~n",[Dens,T1]),
   RandList.

callMenu()->
    {_,Dim}=read('enter Dimension (max=10)>'),
    format("~nPoints have ~w dimensions",[Dim]),
    format("~nMainMenu ~n Select points to be generated~n"),
          % generate four clusters
    format("1.Uniform (200pt):[{0,3,30},{5,9,70},
                               {10,15,50},{20,25,50}]~n"),
    format("2.Zipf (200pt):[{0,3,10},{5,8,10},
                            {10,20,80},{20,35,100}] ~n"),
    format("3.Uniform (5000pt):[{0,5,1000},{7,12,1500},
                                {15,19,1000},{20,25,1500}]~n"),
    format("4.Zipf (5000pt):[{0,5,100},{7,10,200},
                             {11,20,2000},{20,40,2700}]~n"),
          % generate eight clusters
    format("5.Uniform (200pt):[{0,3,25},{5,7,25},{10,15,22},{17,22,28},
                     {24,28,20},{30,35,20},{35,40,30},{40,45,30}]~n"),
    format("6.Zipf (200pt):[{0,2,10},{5,7,10},{10,15,12},{17,20,8},
                     {20,30,100},{30,35,40},{35,39,10},{40,43,10}]~n"),
    format("7.Uniform (5000pt):[{0,4,600},{5,9,700},{10,15,550},
                                {19,22,600},{24,28,600},{30,35,650},
                                {35,40,700},{40,45,600}]~n"),
    format("8.Zipf (5000pt):[{0,3,50},{5,7,60},{10,25,1000},
                             {17,20,200},{18,35,1200},{22,25,100},
                             {35,40,790},{40,45,600}]~n"),
```

```
    {_,MENU}=read('enter >'),
    InitGroup= lists:nth(MENU,[
        [{0,3,30},{5,9,70},{10,15,50},{20,25,50}]
       ,[{0,3,10},{5,8,10},{10,20,80},{20,35,100}]
       ,[{0,5,1000},{7,12,1500},{15,19,1000},{20,25,1500}]
       ,[{0,5,100},{7,10,200},{11,20,2000},{20,40,2700}]
       ,[{0,3,25},{5,7,25},{10,15,22},{17,22,28},
         {24,28,20},{30,35,20},{35,40,30},{40,45,30}]
       ,[{0,2,10},{5,7,10},{10,15,12},{17,20,8},
         {20,30,100},{30,35,40},{35,39,10},{40,43,10}]
       ,[{0,4,600},{5,9,700},{10,15,550},{19,22,600},
         {24,28,600},{30,35,650},{35,40,700},{40,45,600}]
       ,[{0,3,50},{5,7,60},{10,25,1000},{17,20,200},
         {18,35,1200},{22,25,100},{35,40,790},{40,45,600}]
         ]),
     AllP=myGenGroup(Dim,InitGroup),
     format("~nGenerated Points are in points.dat"),
     {_,OutF}=file:open("points.dat",[write]),
     io:fwrite(OutF,"#Comment: File:points.dat /
               Generate all ~wPoints=",[length(AllP)]),
     printFile(OutF,Dim,AllP),
     format("~n%OutPutFile:points.dat~n
                Generate all Points=~w",[AllP]),
     file:close(OutF),p(closeFile,exit),
     WindowList=callWindowSliding(Dim,AllP),
     RealCentroid=kMeans(AllP),
     format("~nRealCentroid with K-means=~w~n",[RealCentroid]),
     mymain(Dim,AllP,WindowList,RealCentroid).

mymain(Dim,AllP,WindowList,RealCentroid)->
    format("~n~nPoints have ~w dimensions",[Dim]),
    format("~nMenu (all choices use the same set of points)~n"),
    format("0. Generate a new set of points~n"),
    format("1. K-means~n"),
    format("2. K-means of Window sliding~n"),
    format("3. Density biased , Hashing~n"),
    format("4. Density biased , Reservoir+Hashing~n"),
    format("5. Density biased , Reservoir+Simple Random~n"),
    format("6. Density biased , Reservoir+Rejection Sampling~n"),
    format("7. K-means , Random~n"),
    format("11. Exit~n"),
    {_,Menu}=read('enter >'),
    case Menu of
    0-> callMenu() ; % generate data points
    1 -> % K-means
         Mem=erts_debug:size(AllP),MemByte =Mem*4,
         format("~nRealCentroid with K-means=~w~n",[RealCentroid]),
         format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
         mymain(Dim,AllP,WindowList,RealCentroid);
    2 -> % K-means of sliding window
         GenL=genWin2P(WindowList),
         EstimateCentroid=kMeans(GenL),
         Sum=sumDistance(EstimateCentroid,RealCentroid),
         Mem=erts_debug:size(GenL),MemByte =Mem*4,
         format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
         p(distance___To___RealCentroid,Sum),
         mymain(Dim,AllP,WindowList,RealCentroid);
```

```
    3 -> % Window sliding + Hashing + K-means
       {_,Den}=read('enter Window Sliding Density>'),
       GenL=genWin2P(callDensity(WindowList,Den)), % gen back to Points
       EstimateCentroid=kMeans(GenL),
       Sum=sumDistance(EstimateCentroid,RealCentroid),
       Mem=erts_debug:size(GenL),MemByte =Mem*4,
       format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
       p(distance___To___RealCentroid,Sum),
       mymain(Dim,AllP,WindowList,RealCentroid);
    4-> % Window sliding + Reservoir&Hashing + K-means
      {_,Den}=read('enter Window Sliding Density>'),
       GenL=genWin2P(callDensityRes(WindowList,Den)),
       EstimateCentroid=kMeans(GenL),
       Sum=sumDistance(EstimateCentroid,RealCentroid),
       Mem=erts_debug:size(GenL),MemByte =Mem*4,
       format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
       p(distance___To___RealCentroid,Sum),
       mymain(Dim,AllP,WindowList,RealCentroid);
    5-> % Window sliding + Reservoir&SimpleRandomSampling + K-means
       {_,Den}=read('enter Window Sliding Density>'),
       GenL=genWin2P(callSimpleRand(WindowList,Den)),
       EstimateCentroid=kMeans(GenL),
       Sum=sumDistance(EstimateCentroid,RealCentroid),
       Mem=erts_debug:size(GenL),MemByte =Mem*4,
       format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
       p(distance___To___RealCentroid,Sum),
       mymain(Dim,AllP,WindowList,RealCentroid);
    6-> % Window sliding + Reservoir&RejectionSampling + K-means
       {_,Den}=read('enter Window Sliding Density>'),
       GenL=genWin2P(callReservoir(WindowList,Den)),
       EstimateCentroid=kMeans(GenL),
       Sum=sumDistance(EstimateCentroid,RealCentroid),
       Mem=erts_debug:size(GenL),MemByte =Mem*4,
       format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
       p(distance___To___RealCentroid,Sum),
       mymain(Dim,AllP,WindowList,RealCentroid);
    7-> % Random + K-means
       GenL=callSimpleRand1(AllP),
       EstimateCentroid=kMeans(GenL),
       Sum=sumDistance(EstimateCentroid,RealCentroid),
       Mem=erts_debug:size(GenL),MemByte =Mem*4,
       format("~n__Memory_Usage=~w Bytes~n",[MemByte]),
       p(distance___To___RealCentroid,Sum),
       mymain(Dim,AllP,WindowList,RealCentroid);
    11-> true ;

    _other -> format("Please choose 1-7~n"),
              mymain(Dim,AllP,WindowList,RealCentroid)
    end.

      % -------- main module ---------------

myMain()-> _= file:delete("points.dat"),callMenu().

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

ภาคผนวก ข

ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

- K. Kerdprasop, N. Kerdprasop and P. Sattayatham (2005). Weighted k-means for density-biased clustering. *Lecture Notes in Computer Science*, Volume 3589 Data Warehousing and Knowledge Discovery (DaWak), August, pp.488-497.

- K. Kerdprasop and N. Kerdprasop (2005). A metaheuristic method to projective clustering. *Proceedings of the Joint Conference on Computer Science and Software Engineering (JCSSE)*, Burapha University, Chonburi, Thailand, November 17-18, pp. 86-90.

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

Weighted K-Means for Density-Biased Clustering

Kittisak Kerdprasop[1], Nittaya Kerdprasop[1], and Pairote Sattayatham[2]

[1] Data Engineering and Knowledge Discovery Research Unit,
School of Computer Engineering, Suranaree University of Technology,
111 University Avenue, Nakhon Ratchasima 30000, THAILAND
{kerdpras,nittaya}@ccs.sut.ac.th
http://www.sut.ac.th/engineering/computer/faculty/nittaya
[2] School of Mathematics, Suranaree University of Technology
111 University Avenue, Nakhon Ratchasima 30000, THAILAND
pairote@ccs.sut.ac.th

Abstract. Clustering is a task of grouping data based on similarity. A popular k-means algorithm groups data by firstly assigning all data points to the closest clusters, then determining the cluster means. The algorithm repeats these two steps until it has converged. We propose a variation called weighted k-means to improve the clustering scalability. To speed up the clustering process, we develop the reservoir-biased sampling as an efficient data reduction technique since it performs a single scan over a data set. Our algorithm has been designed to group data of mixture models. We present an experimental evaluation of the proposed method.

1 Introduction

Clustering is the automatic grouping of data based on similarity. There exists a large number of clustering techniques, but the most classical and popular one is the k-means algorithm [1]. Given a data set containing *n* objects, k-means partitions these objects into *k* groups. Each group is represented by the centroid of the cluster. Once cluster representatives are selected, data objects are assigned to the nearest centers. The algorithm iteratively selects new better representatives and reassigns data objects until no change is made. At this point the algorithm is said to converge. Even though k-means is an effective clustering algorithm, it can sometimes converge to a local optimum. Many methods [2,3,4,5] have been developed to extend the k-means with the common objective of avoiding converging to a bad local optimum. Some methods [6,7,8] search for the best initialization because k-means is known to be sensitive to initial point selection. Other research [9] seeks for the global optimum, at the cost of computation. These researches try to solve the problem of sub-optimal clustering and estimation the appropriate number of clusters [10,11].

Another difficulty of clustering with k-means is that it fails to identify clusters with large variation in sizes since original large clusters tend to be split. Clustering algorithms, such as DBSCAN [12] and CURE [13], have been developed to overcome this kind of difficulty. DBSCAN associates a data point with its density obtained by counting the number of points in a region of radius ε. The algorithm discovers clusters

A Min Tjoa and J. Trujillo (Eds.): DaWaK 2005, LNCS 3589, pp. 488-497, 2005.

by connecting regions with sufficient high density, a *MinPts* threshold. DBSCAN works well in spatial clustering, but it is sensitive to the selection of ε and *MinPts* and it fails to efficiently discover clusters with highly different densities. CURE algorithm represents a cluster by a set of points, instead of a single representative. Once the representative points are chosen, the algorithm then shrinks these points toward the centroid of the cluster according to a shrinking factor. CURE is an iterative hierarchical-based clustering that works well with discovering cluster of different sizes, but it is sensitive to the selection of representatives and shrinking factor. Moreover, with very large data set, these algorithms degrade considerably.

When clustering massive data set, data reduction is an effective technique to speed up the algorithm. Sampling [14,15,16] is a powerful data reduction paradigm to remedy the inherent complexity of clustering. Uniform random sampling in which every data point has the same probability of being selected has been used extensively in data mining and databases [17,18,19,20]. In the case of data sets with large variation in cluster sizes, density biased sampling [21,22,23] tends to be a better scheme. In density biased sampling, the probability that a data point will be included in the sample is varied by the density of a cluster.

Recent researches [21,22,23] propose several techniques to density biased sampling. Our work also follows this path with a step further on extending the k-means algorithm to work with a weighted sample. We propose an algorithm on density biased sampling based on the reservoir technique and a weighted k-means algorithm to cluster a data sample augmented with weights. The proposed algorithms are explained in Sections 2 and 3, respectively. We present the experimental results in Section 4. The conclusion and our future work are discussed in Section 5.

2 Data Reduction Biased by Density

On scalable popular and successful clustering methods such as k-means to work against large data sets, many algorithms like BIRCH [24] and CLARANS [14] employ the sampling technique to minimize data sets. In BIRCH, a CF-tree structure is built after an initial random sampling step. The CF-tree is used as a summarized data structure with statistical representations of space regions stored on leaf nodes. After the phase of CF-tree building, any clustering algorithm can be applied to the leaf nodes. CLARANS also uses uniform sampling to derive initial representative objects for the clusters.

The sampling technique used in these algorithms is uniform random sampling, which assigns every object the same probability of being included in the sample. But many data sets in real life do not follow the uniform distribution scheme. It instead seems to follow the Zipf's distribution [25], for instance, income and population distribution. In these data sets, some areas such as large metropolitan area have much higher population density than the small cities. If all the populations have equal opportunity of being selected as a representative, sparse areas may be missed and not be included in the sample.

2.1 Density-Biased Sampling

Density biased sampling [21] is a sampling technique that takes into account the different sizes of the groups. Small groups or sparse regions are assigned higher probability to be included in the sample than the large groups or dense regions. By biasing the sampling process, small clusters will not be missed or overlooked as outliers.

Recent advancement on clustering very large data sets in which summarized data structure is even too big to fit into main memory, sampling is independently applied to the data set prior to the subsequent clustering phase. Palmer and Faloutsos [21] develop a non-uniform sampling method for clusters that differ very much in size and density. Their method is a generalization of uniform random sampling in that every group of data sets can be assigned different probability of being drawn. When sampling is biased by group density, smaller groups are oversampling, whereas larger groups are under-sampling. Since clusters are not known a priori, Palmer and Faloutsos combine the phase of density information extraction with the biased sampling phase using the hash-based approach. They argue that the inherent collision problem of any hash-based approach will not dramatically degrade the sample.

Nevertheless, their method is significantly affected by noise due to the tendency of oversampling noisy area. Our approach adopts the reservoir technique to eliminate the collision problem of hash-based approach and it is independent on the assumption regarding cluster distribution to avoid the impact of noise.

2.2 Density-Biased Reservoir Sampling

We propose a novel approach of adapting reservoir technique [26,27] to perform a density biased sampling on large data sets. Our algorithm can obtain a desired sample through a single data set scan. The proposed method is simpler and requires less resource than the hash-based method [21].

A reservoir-sampling algorithm [26,27] is a simple, unbiased random sampling algorithm for drawing a sample of size n without replacement from a population of size N ($N \geq n$). Vitter [26] has developed a one-pass reservoir-sampling algorithm when the population size (N) is unknown and cannot be determined efficiently. The term "reservoir" defines a storage area j ($j \geq n$, but mostly $j = n$) to store the potential candidates of the sample. The j reservoirs is initialized to store the first j records of the file, that is, all areas of the reservoir pool are initially filled up. Then the algorithm starts scanning the remaining part of the file with a randomly skipping step. The random picked record is evaluated whether to replace the existing one in the reservoir pool. If it passes the test, the position in the reservoir is also randomly selected. The process stops when the end of file has been reached and the records in the reservoir form a simple random sample of the population. The general procedure of reservoir-sampling algorithm [27,28] is given in Figure 1.

The time complexity of the algorithm is shown [26,27] to be O(n (1+ log(N/n)). In the reservoir-sampling algorithm, each record of the file is assigned a uniform (0,1) random number. When the reservoir is needed to be updated, each record in the reservoir has the same chance to be replaced by the new record.

Algorithm Reservoir sampling
Input: a sequential file of N population
Output: a random sample of size n ($n \leq N$)

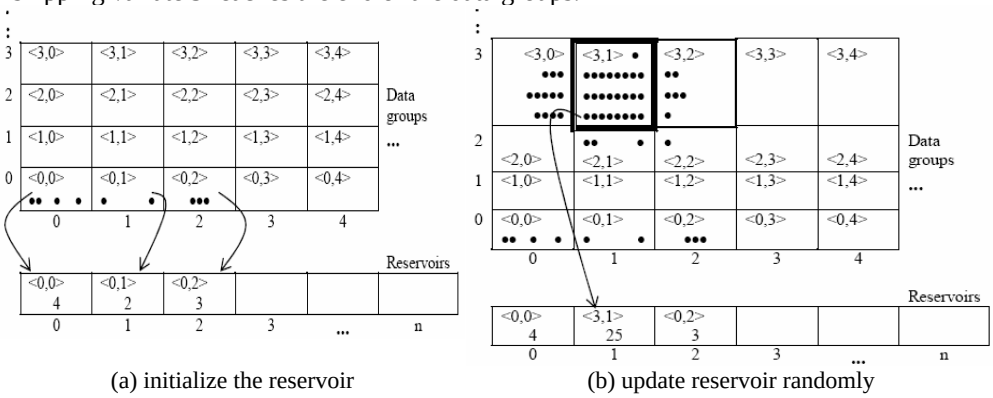
1) Initialize the reservoir $X_1, ..., X_n$ to be the first n records of the file
2) Initialize W to be the largest value in a sample of size n from the uniform distribution on the interval (0, 1)
3) While not eof do
4) Generate the random variate S to denote the number of records to be skipped over before a new record can enter the reservoir
5) If (not eof) Then Search for the next potential record to be in the reservoir
6) Else return $X_1, ..., X_n$
7) Update X and W

Fig. 1. Reservoir-sampling algorithm

Our sampling algorithm generalizes the reservoir scheme for the case of data with different density distribution. In our proposed method, the initial step of partitioning data into groups resembles that of Palmer and Faloutsos [21]. But our subsequent steps are not based on hashing scheme in order to avoid the effect of noise and collision problems.

After the initial step of dividing the data space into bins of equal size, the information of the first n groups are put into the n reservoirs residing in main memory (see Figure 2a). The collected information includes the number of points in each group and the id of the group.

The algorithm performs a single scan on a data set in a random manner controlled by a random variate S with the distribution W. The density biasing (step 7 in Figure 3) is achieved through the consideration of two consecutive data groups. If the density difference of the two data groups is above some threshold δ (i.e., detecting cluster edge) or the sum of density on both groups is above the threshold value ε (i.e., avoiding noisy cases), then the denser group is a candidate to be included in a sample. This new candidate is put into a reservoir pool at a random position (the reservoir update is pictorially shown in Figure 2b). The density-biased sampling proceeds until the skipping variate S reaches the end of the data groups.



(a) initialize the reservoir (b) update reservoir randomly

Fig. 2. Density biasing in a reservoir scheme

Algorithm Density-biased reservoir sampling

Input: a data set of N objects
Output: a density-biased sample of size n ($n \leq N$) associated with weight w

1) Partition data into g groups (with group-id $1,2,...,g$), $g \geq n$
2) Initialize the reservoir $X_1, ..., X_n$ to be the first n <group-id, density>-pairs of the data groups
3) Set $W \leftarrow \exp(\log(\text{random}()) / n)$ // initialize W that will be used in the // generation step of random variate S
4) Set $S \leftarrow \lfloor \log(\text{random}()) / \log(1-W) \rfloor$
5) While $S < g$ do
6) Read data groups g_S and g_{S+1} // read two consecutive data groups
7) If $(\|\text{density}(g_S) - \text{density}(g_{S+1})\| > \delta)$ OR $((\text{density}(g_S) + \text{density}(g_{S+1})) > \varepsilon)$
// δ and ε are predefined density threshold values
Then $X_{1+\lfloor n * \text{random}() \rfloor} \leftarrow$ <group-id, density> of maximum density$\{g_S, g_{S+1}\}$
// randomized the reservoir area to be updated
8) $W \leftarrow W * \exp(\log(\text{random}()) / n)$ // update W for the skipping process
9) $S \leftarrow \lfloor \log(\text{random}()) / \log(1-W) \rfloor$ // generate S to denote the number of // groups to be skipped over
10) Return $X_1, ..., X_n$

Fig. 3. Density-biased reservoir sampling algorithm

3 Weighted K-Means Algorithm

The classical k-means algorithm [1] is a fast method to perform clustering. The algorithm consists of a simple re-estimation procedure as outlined in Figure 4.

Algorithm K-means

Input: a set of n data points, and the number of clusters (K)
Output: centroids of the K clusters

1) Initialize the K cluster centers
2) Repeat
Assign each data point to its nearest cluster center
3) Recompute the cluster centers using the current cluster memberships
4) Until there is no further change in the assignment of the data points to new cluster centers

Fig. 4. K-means algorithm

The original n data points to be clustered are contained in the dataset $X = \{x_1, ..., x_n\}$. The k-means algorithm partitions n data points into K sets. The assignment of a data point x_i to its nearest cluster center c_j (step 2) is decided on the basis of the membership function, $m(c_j|x_i)$. The function returns either one of the {0,1} values: $m(c_j|x_i) = 1$ if $j = argmin_k||x_i - c_k||^2$; it is zero, otherwise. In step 3, the new centroids of clusters can be computed from all data points x_i in the cluster. The objective function J of the algorithm is to minimize the sum of error squared, $J = \Sigma_{i = 1:n} min_{j \in \{1..k\}} || x_i - c_j ||^2$.

In k-means algorithm, every data point has equal importance in locating the centroid of the cluster. This property does no longer hold in the case of density-biased sample clustering, for which each data point represents varied density in the original data. Therefore, the clustering algorithm has to consider a weight associated with each data point in the computation of cluster centers. The proposed extension to the k-means algorithm is called weighted k-means. Figure 5 outlines the algorithm.

Algorithm Weighted k-means

Input: a set of n data points obtained from the density-biased reservoir sampling, and the number of clusters (K)
Output: centroids of the K clusters

1) Initialize the K cluster centers

2) Repeat
Assign each data point to its nearest cluster center according to the membership function,

$$m(c_j|x_i) = \frac{|| x_i - c_j ||^{-p-2}}{\Sigma_{j = 1:k} || x_i - c_j ||^{-p-2}}$$

3) For each center c_j, recompute the cluster center c_j using the current cluster memberships and weights,

$$c_j = \frac{\Sigma_{i = 1:n}\ m(c_j|x_i)\ w(x_i)\ x_i}{\Sigma_{i = 1:n}\ m(c_j|x_i)\ w(x_i)}$$

where $w(x_i)$ is a weight associated with each data point

4) Until there is no reassignment of data points to new cluster centers

---

Fig. 5. Weighted k-means algorithm

The membership function in the weighted k-means algorithm resembles that of the k-harmonic means algorithm [5]. Zhang [5] also introduces the weight function, $w(x_i)$, in his algorithm to accelerate the recomputation of the new centroids in the next iteration. The weight function in our algorithm, however, is introduced for the different purpose. It represents the density of the original data points.

4 Experiments and Results

We perform two sets of experiments to test the quality of our sampling method, which is the step prior to clustering, and to measure the quality of the weighted k-means algorithm.

4.1 Performance of Density-Biased Reservoir Sampling

We evaluate the performance of the proposed reservoir-based density bias sampling method against the hash-based sampling method [21]. The efficiency regarding memory usage of our reservoir-based sampling method is obviously better than the hash-based method. In the hashing scheme, some amount of memory is needed to store the hashing table in addition to the memory required for storing the drawn sample. Thus, it requires twice the amount of memory comparative to those required by our method.

Effectiveness of the proposed sampling method is examined by measuring the quality of a sample with respect to the number of correctly found clusters. We run clustering using the k-means algorithm. We use a synthetic data generator to generate d-dimensional data sets having *k* clusters and *N* data points. We vary d from 2 to 5, *k* from 2 to 10, and *N* from 5,000 to 100,000.

The measurement *Number of Clusters found* (NC) is the metric defined in [21]. NC is calculated by comparing the distances of the cluster centers found by the clustering algorithm with the true cluster centers. We say that the cluster is found if the calculated distance is less than a predefined threshold (e.g., 0.001).

The results in Figure 6 show the NC when run clustering on various sample sizes with the presence of noise. The reported results are observed from the experiments using 3-dimensional data set having 7 clusters. One cluster contains 50,000 points and the other six clusters contain 500 points. The results obtained from other experiments on data sets with different dimensions, various number of clusters, and varied number of data points are conformed with the one presented in Figure 6, so we omit them for brevity. The experimental results reveal the efficiency of the biased reservoir method especially in the presence of noise.

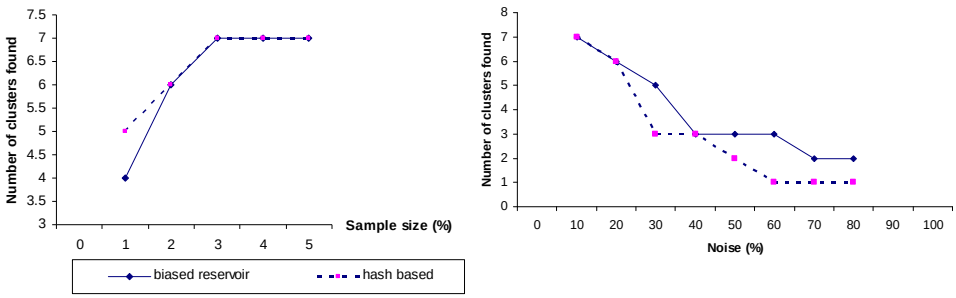


Fig. 6. Finding clusters of 3-dimensional data on various sample sizes, in the presence of noise

4.2 Performance of Weighted K-Means Algorithm

We evaluate the quality of the weighted k-means algorithm against the k-means algorithm by using the squared objective function. Lower value of a squared objective function reflects a better quality on clustering. The experiments perform on the syntactic data sets explained in Section 4.1. The initialization step randomly selects data points as initial cluster centroids. We also consider running time of both algorithms.

The performance evaluation as shown on top of Figure 7 is obtained from running k-means and weighted k-means algorithms on 3-dimensional data sets of sizes varied from 5000, 10000, 20000, 35000, 55000, 75000, to 100000 data points. The number of clusters is set to be 10. The experiments are performed on the PC computer with CPU speed 800 MHz, memory 512 MB. Since all data points are used in weighted k-means algorithm, the weight function is set to be 1. The parameter p in the membership function is set to be 1.3.

The comparison on clustering quality and running time shown at the bottom of Figure 7 reveals the efficiency of running weighted k-means on density-biased sample. The experiments are performed on 10% sample of data with two methods of sampling: simple random sampling (RS) and density-biased reservoir sampling (DBS). The weight function of the weighted k-means algorithm is varied according to the density of the original data.

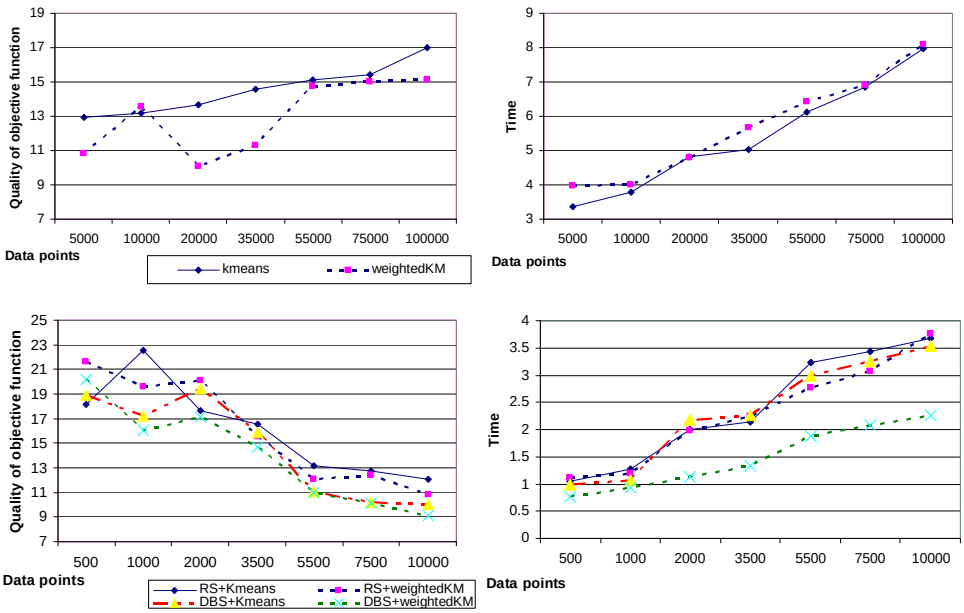


Fig. 7. Performance comparison of weighted k-means against k-means (left) and the running time comparison (right), results on top are experiments running on the whole data set while results at the bottom obtained from running on the sample data

5 Conclusions

The k-means is the simplest and most commonly used clustering algorithm. The simplicity is due to the use of squared error as the stopping criteria, which tends to work well with isolated and compact clusters. Its time complexity depends on the number of data points to be clustered and the number of iteration. We propose a variation of the k-means to better work with a large data set having much difference in cluster density. Our intuition idea is that to cope with massive data set, sampling should be the efficient data reduction method. Since the original data is assumed to be much varied in cluster sizes, density-biased sampling is an appropriate method to preserve the density.

We propose a density biased sampling technique based on the reservoir method. The inherent advantage of efficient memory usage in the reservoir scheme is adopted and extended with the additional capability of dealing with data that are much different in density distribution. The proposed technique is designed to lessen the effect of noise as it is the case in the hash-based approach. The experimental results have shown that the proposed method is as good as the hash-based method in discovering correct number of clusters. Our method, moreover, is less sensitive to noisy data even when the percentage of noise is greater than 20.

We also develop the weighted k-means algorithm to better cluster a sample data biased by its density. The results demonstrate the efficiency of the algorithm. The evaluation of the proposed methods on real large databases and the consideration of outliers are our future work.

Acknowledgements

This research has been supported by grants from the Thailand Research Fund (TRF, MRG4780170), and the National Research Council. The Data Engineering and Knowledge Discovery Research Unit is fully supported by the research grants from Suranaree University of Technology.

References

1. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol.1. University of California Press (1967) 281-297
2. Hamerly, G., Elkan, C.: Alternatives to the k-means algorithm that find better clusterings. In Proceedings of 11th ACM CIKM International Conference on Information and Knowledge Management (2002) 600-607
3. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York (1981)
4. Pellog, D., Moore, A.: Accelerating exact k-means algorithms with geometric reasoning. In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999) 277-281
5. Zhang, B.: Generalized k-harmonic means - boosting in unsupervised learning. Technical Report HPL-2000-137. Hewlett-Packard Labs (2000)
6. Bradley, P.S., Fayyad, U.M.: Refining initial points for k-means clustering. In Proceedings of the 15th International Conference on Machine Learning (1998) 91-99

7. Meila, M., Heckerman, D.: An experimental comparison of model-based clustering methods. Machine Learning 42(2001) 9-29
8. Pena, J., Lozano, J., Larranaga, P.: An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognition Letters 20(1999) 1027-1040
9. Likas, A., Vlassis, N., Verbeek, J.: The global k-means clustering algorithm. Technical Report IAS-UVA-01-02. Computer Science Institute, University of Amsterdam, The Netherlands (2001)
10. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. In Proceedings of the 17th International Conference on Machine Learning (2000) 727-734
11. Sand, P., Moore, A.: Repairing faulty mixture models using density estimation. In Proceedings of the 18th International Conference on Machine Learning (2001)
12. Sander, J., Ester, M., Kriegel, H.-P., Xu, X.: Density-based clustering in spatial databases: The algorithm GDBSCAN and its application. Data Mining and Knowledge Discovery 2(1998) 169-194
13. Guha, S., Rastogi, R., Shim, K.: CURE: An efficient clustering algorithm for large databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data (1998) 73-84
14. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In Proceedings of International Conference on Very Large Data Bases (1994) 144-155
15. Zhou, S., Zhou, A., Cao, J., Wen, J., Fan, Y., Hu., Y: Combining sampling technique with DBSCAN algorithm for clustering large spatial databases. In Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (2000) 169-172
16. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: C^2P: Clustering based on closest pairs. In Proceedings of International Conference on Very Large Data Bases (2001) 331-340
17. Singh, G., Rajagopalan, S., Lindsay, B.: Random sampling techniques for space efficient of large data sets. In Proceedings of ACM SIGMOD International Conference on Management of Data (1999)
18. Toivonen, H.: Sampling large databases for association rules. In Proceedings of International Conference on Very Large Data Bases (1996) 134-145
19. Thompson, S.K., Seber, G.A.F.: Adaptive Sampling. John Wiley & Sons, New York (1996)
20. Olken, F., Rotem, D.: Sampling from spatial databases. In Proceedings of International Conference on Data Engineering (1993) 199-208
21. Palmer, C., Faloutsos, C.: Density biased sampling: An improved method for data mining and clustering. In Proceedings of ACM SIGMOD International Conference on Management of Data (2000) 82-92
22. Nanoupoulos, A., Theodoridis, Y., Manolopoulos, Y.: An efficient and effective algorithm for density biased sampling. In Proceedings of 11th International Conference on Information and Knowledge Management (2002) 63-68
23. Kollios, G., Gunopulos, D., Koudas, N., Berchtold, S.: Efficient biased sampling for approximate clustering and outlier detection in large data sets. IEEE Transactions on Knowledge and Data Engineering 15(2003) 1-18
24. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In Proceedings of ACM SIGMOD International Conference on Management of Data (1996) 103-114
25. Zipf, G.K.: Human Behavior and Principle of Least Effort: An Introduction to Human Ecology. Addison Wesley, Cambridge, MA (1949)
26. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software 11(1985) 37-57
27. Li, K.-H.: Reservoir-sampling algorithms of time complexity O(n(1 + log(N/n))). ACM Transactions on Mathematical Software 20(1994) 481-493
28. Devroye, L.: Non-Uniform Random Variate Generation. Springer-Verlag, New York (1986)

Proceedings of The Joint Conference on Computer Science and Software Engineering,
Burapha University, Chonburi, Thailand, November 17-18, 2005, pp.86-90

A Metaheuristic Method to Projective Clustering

Kittisak Kerdprasop and Nittaya Kerdprasop
Data Engineering and Knowledge Discovery (DEKD) Research Unit
School of Computer Engineering, Suaranaree University of Technology
e-mail: {kerdpras, nittaya}@sut.ac.th

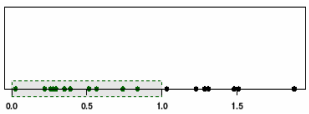
Abstract

Projective or subspace clustering is a clustering technique for high dimensional data with the inherent sparsity of data points. To overcome the unreliable measure of similarity among sparse data points in high dimensions, all data points are projected to a lower dimensional subspace. The conventional clustering algorithms can then be applied to the transformed data set. However, the clustering algorithms do not handle well the situation that different clusters are formed in different subspaces. We, thus, propose a metaheuristic method to iteratively compute projective clusters. The underlying technique of our metaheuristic is the principal component analysis. Our proposed method is embedded in the standard k-means clustering algorithm.

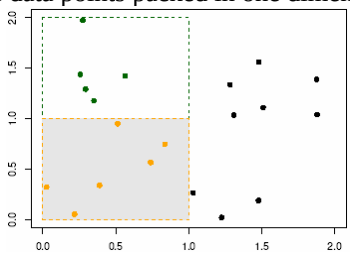
Key-Words: Projective clustering, Metaheuristic

1. Introduction

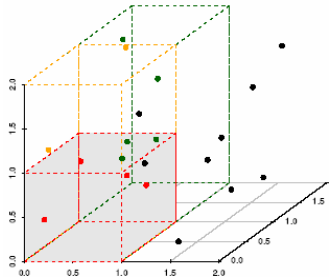
Clustering is a widely used technique to discover homogeneous groups, or clusters, of data according to a certain similarity measure such that data of one cluster are similar to each other whereas data of different clusters are dissimilar. Many algorithms have been designed [11] to compute a partition on full-dimensional data set. While these approaches work successfully on low-dimensional data sets, their efficiency decreases significantly in higher dimensional space [10, 13]. In high dimensional data, some dimensions tend to be redundant or irrelevant. Massive dimensions can confuse the clustering algorithms. It is also difficult to group similar data points in very high dimensions because the distance between any two data points becomes almost the same [5, 9] due to the increase in sparsity of the data set as shown in Figure 1. The most difficult problem on clustering high dimensional data is that different clusters may exist in different subspaces of different dimensions [4], pictorially shown in Figure 2.



(a) 11 data points packed in one dimension



(b) adding one more dimension results in stretching data points apart and decreasing the population down to 6 data points in a lower-left bin



(c) with three dimensions, there exists only 4 data points in a lower-left bin

Figure 1. The effect of increasing dimensions on data grouping

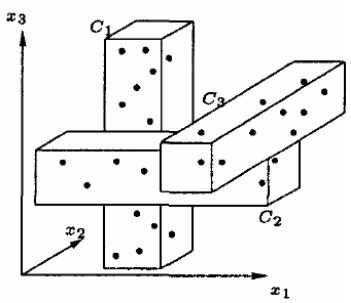


Figure 2. Three clusters in three different subspaces

A possible solution to these problems is to use dimensionality reduction or feature selection techniques. By means of dimensionality reduction, one first reduces the dimensions of the original data set by removing less important dimensions or by transforming the original data set into a lower dimensional space. The conventional clustering algorithms can then be applied to the new data set. However, an attempt to reduce dimensions of all data points results in significant information loss.

Recognizing the need for an efficient algorithm for clustering high dimensional data, the concept of *subspace clustering* or *projective clustering* has thus been proposed [1, 2, 3, 4]. The goal of projective clustering is to find clusters embedded in lower dimensional subspaces. It can minimize the information loss in the process of dimensionality reduction by projecting those high dimensional data points into different lower dimensional subspaces for different clusters.

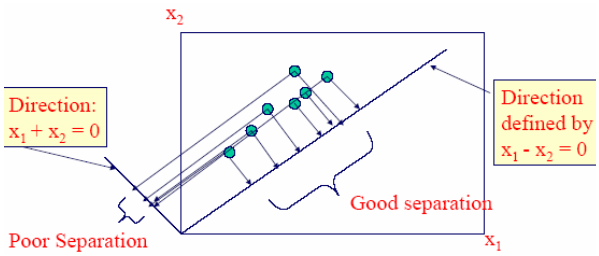
Statistical methods such as Principal Component Analysis (PCA) [12] can effectively reduce dimensions of the original data by projecting all points onto a subspace so that the information loss is minimized. Then, a standard clustering method can be used in this subspace. However, PCA does not work well when different subsets of data points embedded in different lower-dimensional subspaces. We, thus, propose the metaheuristic method to iteratively apply PCA aiming at transforming the original data set into various lower-dimensional subspaces. The term *metaheuristic* [8] refers to a high-level approximate algorithm that guides an underlying heuristic to efficiently produce high-quality solution, which in our specific domain is the most informative lower dimensional subspace. The subsequent steps of clustering process then compute a partitioning of data points into disjoint groups.

We briefly explain the concept of PCA as a major approach to dimensionality reduction in Section 2. The proposed method of applying metaheuristic to the conventional clustering algorithm is explained in Section 3. The experimental results shown in Section 4 verify the efficiency of the proposed method. Conclusion remarks are presented in Section 5.

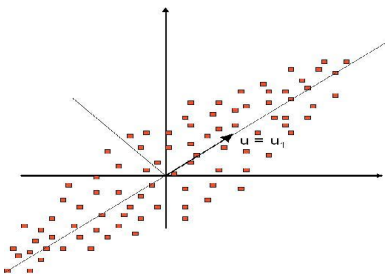
2. Dimensionality reduction

Dimensionality reduction is an important preprocessing step for unsupervised clustering, especially on high dimensional data set. There are two major approaches to dimensionality reduction: feature selection and feature transformation. Feature selection is a process of finding a minimum subset of original features that satisfies some criteria, such as information measure, and no new feature to be generated. Feature transformation methods, on the other hand, transform data from the original *d*-dimensional feature space to a new *k*-dimensional feature space (*k* << *d*) through some functional mapping.

Principal component analysis [12, 14], sometimes called the Karhunen-Loeve (KL) transformation [4], is a widely used method for feature transformation to reduce the number of dimensions of a data set. Figure 3 sketches the idea of PCA to project two dimensional data onto a one dimensional subspace. PCA finds basis vector for a subspace which maximizes the variance retained in the projected subspace, i.e., the direction or vector defined by x_1-x_2 = 0 as shown in Figure 3(a).



(a) linear projection of data to a lower subspace



(b) the first principal component and the second principal component in the orthogonal direction

Figure 3. Dimensionality reduction by PCA

Given N data points $x_1,...,x_n$ in *d*-dimensional space, PCA projects the correlated high dimensional data onto a lower dimensional space. The transformation process looks for a unit vector u_1 (as shown in Figure 3(b)) such that on average the squared length of the data projection along the u_1 is maximal:

$$u_1 = \arg\max_{\|u\|=1} E\left[\left(u^T x\right)^2\right]$$

The next principal component is the one that maximizes the residual variance:

$$u_k = \arg\max_{\|u\|=1} \mathrm{E}\left[\left(x - \sum_{i=1}^{k-1}\left(u_i^T x\right)u_i\right)^2\right]$$

The principal components for the data vectors are given by $C_i = u_i x$; u_i is called eigenvector which is the vector that shows the direction of maximal variance

of the data. To find the eigenvector u_i, we first calculate the covariance matrix:

$$C = E[xx^T].$$

Then find out the eigenvalues λ_i and eigenvectors u_k from the covariance matrix:

$$Cu_k = \lambda_k u_k$$

Eigenvalue is a measure of the proportion of the variance explained by the corresponding eigenvector.

3. Projective clustering and metaheuristic

Our approach aims at performing clustering in low dimensional subspaces, instead of the original high dimensional space where clusters are not well-separated. The intuitive idea is to reduce the dimensions using PCA. Since the embedded clusters may lie in different subspaces, the subspace initially obtained using PCA does not necessary coincide with the subspace spanned by the k cluster centers. Therefore, we propose to iteratively perform PCA and clustering on the reduced subspace until the convergence criteria has been reached. The algorithm can be defined as follows.

Algorithm Clustering with metaheuristic

Input: a set of data vectors X = [x_1, ..., x_n] of *d* dimensions and the number of cluster (*k*)

Output: a set of cluster centers C_v = [v_1, ..., v_k] and the relevant attributes

Steps:

1. *Initialization:* Refine the initial points for clustering (as proposed in [7]) on sample data, and center the data matrix X so that the value of each variable is subtracted for that variable.
2. Do the first dimension reduction using PCA to obtain the *q*-dimensional subspace, *q* << *d*.
   (Metaheuristic is applied here.)
3. While not convergence
   3.1 Run clustering algorithm on the *q*-dimensional subspace to obtain clusters.
   3.2 Use cluster membership to construct the *k* cluster centroids in the original space.
   3.3 Compute the span of *k* centroids using singular value decomposition.
   3.4 Apply PCA to obtain a new *q*-dimensional subspace. (Metaheuristic is applied here.)
4. Return a set of cluster centers associated with relevant attributes

Figure 4. Projective clustering with metaheuristic

We propose the method for iterative high dimensional clustering on the basis of fuzzy k-means [6]. A prior probability of the cluster can be computed as:

$$\alpha_i = \frac{1}{n}\sum_{j=1}^{n}\gamma_{i,j},$$

where γ is the degree of membership.

The cluster centers are determined from

$$v_i^x = \frac{\sum_{k=1}^{n}(\gamma_{i,k})^m (x_k - W_i\langle y_{i,k}\rangle)}{\sum_{k=1}^{n}(\gamma_{i,k})^m}$$

where the expectation of the latent variables is

$$\langle y_{i,k}\rangle = M_i^{-1}W_i^T(x_k - v_i^x),$$

the q×q matrix $M_i = \sigma_{i,x}^2 I + W_i^T W_i$ and the fuzzy weighting component $m = 2$. The new value of W_i can be computed from

$$\tilde{W}_i = F_i W_i (\sigma_{i,x}^2 I + M_i^{-1}W_i^T F_i W_i)^{-1}.$$

The covariance matrix F_i can be computed by

$$F_i = \frac{\sum_{k=1}^{n}(\gamma_{i,k})^m (x_k - v_i^x)(x_k - v_i^x)^T}{\sum_{k=1}^{n}(\gamma_{i,k})^m}.$$

The new value of $\sigma_{i,x}^2$ is

$$\sigma_{i,x}^2 = \frac{1}{q}tr(F_i - F_i W_i M_i^{-1}\tilde{W}_i^T).$$

The fuzzy covariance matrix is

$$A_i = \sigma_{i,x}^2 I + \tilde{W}_i\tilde{W}_i^T.$$

The square distance measurement, D^2, for the fuzzy k-means is defined as the product of three terms: prior probability of the cluster, the distance between the k^{th} data point and the centroid v_i of cluster i, and the distance between the cluster prototype and the data in the subspace. The objective function, J, of the clustering is

$$J = \sum_{i=1}^{c}\sum_{k=1}^{n}(\gamma_{i,k})^m D^2 + \sum_{k=1}^{n}\lambda_i(\sum_{i=1}^{c}\gamma_{i,k} - 1).$$

Metaheuristic

In the dimensionality reduction steps (steps 2 and 3.4 in Figure 4), we use a simple and fast method to determine the number of principal dimensions to be retained at each level. For component i, the dimension q_i to be retained in the corresponding sub-components in the next level is considered from the

two criteria: the proportion of variance of the first r components and the size of important variance.

The proportion of variance of the first r components can be computed from the summation of the first r variances divided by the sum of all variances. We set projections accounting for over 85% of the total variance as a threshold, that is,

$$\frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i} > 0.85.$$

The size of important variance is a simple measurement to discard principal components that have sample variances below $\overline{\lambda}$,

$$\overline{\lambda} = \frac{1}{d}\sum_{i=1}^{d}\lambda_i \ .$$

The intuitive idea is that if $\lambda_i < \overline{\lambda}$, then the i^{th} principal direction is less interesting than average.

Our method of determining how many principal components to retain is to consider from the proportion of variance and the size of importance variance,

$$(\frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i} > 0.85) \text{ OR } (\lambda_i > \overline{\lambda}).$$

With the proposed criteria, each cluster component can have potentially different dimensionality.

4. Experimental evaluation

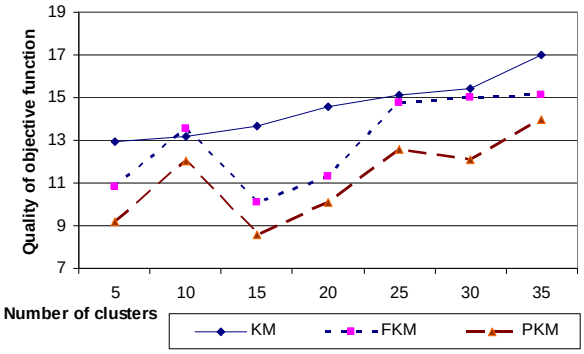
We compare our proposed projective clustering (PKM) with the fuzzy k-means (FKM) and k-means (KM) algorithms. The experiments are performed on the Pentium IV 1.0 GHz machine with 512 MB of main memory. We generate the synthetic data sets of 50,000 points with varied dimensions up to 100. To assess the quality of a clustering algorithm we use the distance metric

$$\sqrt{\sum_{i=1}^{n}\min_{j\in\{1..k\}} \| x_i - c_j \|^2}.$$

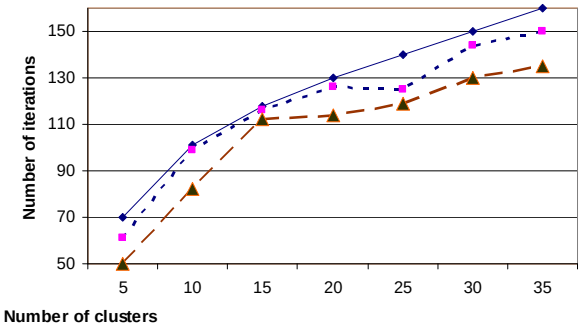
The performance is evaluated on the clustering quality, the number of iteration toward convergence criteria, and the running time. The results are shown in Figure 5.

5. Conclusions

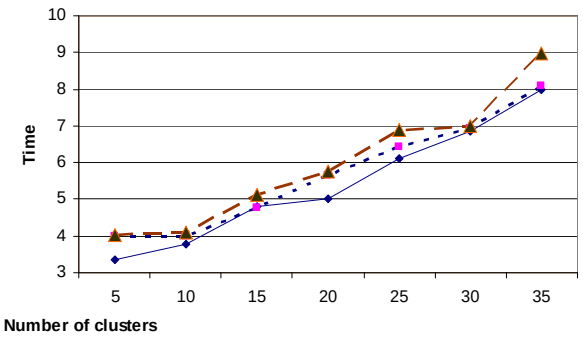
We have introduced a new method for clustering high dimensional data using the concept of projective clustering. The algorithm is based on the fuzzy partitioning clustering algorithm. The key to the effectiveness of finding clusters on different subspaces is due to the power of PCA. The main justification of powerful dimension reduction is that PCA uses singular value decomposition which gives the best low rank approximation to original data. We apply a metaheuristic to the PCA process to project data into different subspaces for the effectiveness of discovering clusters embedded in different layers. The experimental results confirm the efficiency of our proposed method. Running experiments on real data is an essential step toward the practicality improvement of the method.



(a) a comparison on clustering quality (the lower is the better)



(b) a comparison on number of iteration



(c) a comparison on running time

Figure 5. Performance comparison of projective clustering (PKM), k-means (KM), and fuzzy k-means (FKM)

6. Acknowledgements

This research has been supported by grants from the Thailand Research Fund (TRF, MRG4780170), and the National Research Council. The Data Engineering and Knowledge Discovery (DEKD) Research Unit is fully supported by the research grants from Suranaree University of Technology.

7. References

[1] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park, "Fast algorithms for projected clustering", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 1999, pp. 61-72.

[2] C. Aggarwal and P. Yu, "Finding generalized projected clusters in high dimensional spaces", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp.70-81.

[3] C. Aggarwal and P.Yu, "Redefining clustering for high-dimensional applications", *IEEE Transactions on Knowledge and Data Engineering*, 14(2), 2002, pp.210-225.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 1998, pp.94-105.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful?", *Proceedings of 7th Int. Conf. on Database Theory*, 1999, pp.217-235.

[6] J. Bezdek and J. Dunn, "Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions", *IEEE Transactions on Computers*, 1975, pp.835-838.

[7] P. Bradley and U. Fayyad, "Refining initial points for k-means clustering", *Proceedings of 15th Int. Conf. on Machine Learning*, 1988, pp.91-99.

[8] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers and Operations Research*, 13, 1986, pp.533-549.

[9] A. Hinneburg, C. Aggarwal, and D. Keim, "What is the nearest neighbor in high dimensional spaces?", *Proceedings of 26th Int. Conf. on Very Large Data Bases*, 2000, pp.506-515.

[10] A. Hinneburg and D. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high dimensional clustering", *Proceedings of 25th Int. Conf. on Very Large Data Bases*, 1999, pp.506-517.

[11] A. Jaine and R. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[12] I. Jolliffe, *Principal Component Analysis*, Springer Verlag, 1986.

[13] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review", *SIGKDD Explorations*, 6(1), 2004, pp.90-105.

[14] K. Yeung and W. Ruzzo, "Principal component analysis for clustering gene expression data", *Bioinformatics*, 17(9), 2001, pp.763-774.

A Monte Carlo Method to Data Stream Analysis

Kittisak Kerdprasop, Nittaya Kerdprasop, and Pairote Sattayatham

***Abstract*—**Data stream analysis is the process of computing various summaries and derived values from large amounts of data which are continuously generated at a rapid rate. The nature of a stream does not allow a revisit on each data element. Furthermore, data processing must be fast to produce timely analysis results. These requirements impose constraints on the design of the algorithms to balance correctness against timely responses. Several techniques have been proposed over the past few years to address these challenges. These techniques can be categorized as either data-oriented or task-oriented. The data-oriented approach analyzes a subset of data or a smaller transformed representation, whereas task-oriented scheme solves the problem directly via approximation techniques. We propose a hybrid approach to tackle the data stream analysis problem. The data stream has been both statistically transformed to a smaller size and computationally approximated its characteristics. We adopt a Monte Carlo method in the approximation step. The data reduction has been performed horizontally and vertically through our EMR sampling method. The proposed method is analyzed by a series of experiments. We apply our algorithm on clustering and classification tasks to evaluate the utility of our approach.

***Keywords*—**Data Stream, Monte Carlo, Sampling, Density Estimation.

I. Introduction

DATA analysis is the process of computing various summaries and derived values from collected data. Data mining can be viewed as an intelligent data analysis aiming at extracting valuable knowledge from large amounts of information stored in data repositories [1], [3]. The techniques used in data mining have been adopted from the areas of machine learning and statistics, but scalable to deal with the problem of huge repositories of information. The recent advances in hardware and software have enabled the rapid generation of continuous stream of information such as customer click streams, telephone records, retail chain transactions, web page visits, and so on. Mining stream data that grow at an unlimited rate poses a new challenge to researchers and practitioners in the area of data mining [1], [9].

Data stream is defined as massive amounts of data continuously generated at a rapid rate, possibly time-varying and unpredictable [2], [9]. Major characteristics of data streams are the continuously online arrival of data elements, uncontrolled order of such elements upon arrival, variable sizes, and a one-time processing of an element before it is discarded or archived due to the massive size of data that far exceeds the storage capacity. The requirements of timely analysis and efficient memory usage constrain most data stream mining algorithms to sacrifice accuracy of the analysis results for the fast and feasible processing.

Development of approximation algorithms [5], [13] is a direct solution to the problem of data stream mining. However, the large volumes of data continuously arriving in a stream could eventually make the algorithms inefficient. A more practical solution is to apply a data reduction technique along with the approximation algorithms. Data summarization techniques, such as wavelet analysis [10] and histogram [2], have been proposed as synopsis data structures to provide a summary presentation of data. The issue of dynamic space allocation as the underlying data distribution changes over time is a fundamental problem of these approaches. Data stream analysis by choosing a subset of the incoming stream is another class of techniques for producing approximate results. Sampling is a statistical-based technique widely used to scale up the mining algorithms [7]. Nevertheless, in the context of data stream in which the data size is unknown, simply applying a sampling method cannot give reliable approximation.

We, therefore, propose a Monte Carlo method to draw representatives from data stream. Monte Carlo simulation is a widely used method to produce a good approximation to the true value or quantity. Our algorithm has been designed to produce data elements from which the approximate analysis is close to the exact one. We perform cluster and classification analyses on several data sets to verify the reliability of the method.

The paper is organized as follows. Section 2 presents the theoretical background of a general Monte Carlo method. Section 3 sketches the draft idea of density estimation from a sample. Our proposed method that is efficiently applicable to data stream analysis is explained in Section 4. Some of the experimental results from cluster and classification analyses over the reduced data stream are shown in Section 5. We conclude in Section 6 with a discussion for future work.

This work was supported by the Thailand Research Fund under Grant MRG4780170, the National Research Council, and Suranaree University of Technology for the sponsorship of Data Engineering and Knowledge Discovery Research Unit.

Kittisak Kerdprasop is with the School of Computer Engineering, and the director of Data Engineering and Knowledge Discovery Research Unit Suranaree University of Technology, Nakhon Ratchasima 30000, Thailand (e-mail: kerdpras@ sut.ac.th).

Nittaya Kerdprasop is with the School of Computer Engineering, and the member of Data Engineering and Knowledge Discovery Research Unit Suranaree University of Technology, Nakhon Ratchasima 30000, Thailand (e-mail: nittaya@ sut.ac.th).

Pairote Sattayatham is with the School of Mathematics, Suranaree University of Technology, Nakhon Ratchasima 30000, Thailand (e-mail: pairote@ sut.ac.th).

II. Principles of Monte Carlo

Monte Carlo method is a class of stochastic algorithm for simulating the behavior of physical or mathematical systems [11], [14]. The term stochastic implies that the methods are non-deterministic in which they are based on the use of random numbers and probability statistics to investigate problem. To understand the method of Monte Carlo, it is useful to think of it as a general technique of numerical integration. Suppose we need to evaluate the *d*-dimensional integral of a function *f* over the unit interval

$$\Phi = \int_0^1 \int_0^1 \dots \int_0^1 f(x_1, x_2, \dots, x_n)\, dx_1\, dx_2 \dots dx_n = \int_{(0,1)^d} f(x)\, dx. \quad (1)$$

The integral is a non-random problem, but the Monte Carlo method represents the integral as an approximation problem by introducing a random vector *U* that is uniformly distributed between 0 and 1. Applying the function *f* to *U*, we obtain a random variable $f(U)$ with expectation

$$E[f(U)] = \int_{(0,1)^d} f(x)\phi(x)\, dx \quad (2)$$

where ϕ is the probability density function of *U*. Since the value of ϕ on the region of integration is 1, equation (2) becomes

$$E[f(U)] = \int_{(0,1)^d} f(x)\, dx \quad (3)$$

Equations (1) and (3) allow us to represent the integral Φ as a probabilistic expression as follow:

$$\Phi = E[f(U)] \quad (4)$$

To estimate Φ, we need a mechanism for drawing points U_1, U_2, ...,U_n. Applying function *f* to each of these *n* random points yields *n* independent and identically distributed (*iid*) random variables $f(U_1)$, $f(U_2)$, ..., $f(U_n)$, each with expectation Φ and standard deviation σ. Averaging the results produces the *Monte Carlo estimator*

$$\hat{\Phi} = \frac{1}{n}\sum_{i=1}^{n} f(U_i) \quad (5)$$

which is an unbiased estimator for Φ with the error $\hat{\Phi}$ - Φ approximately normally distributed with mean 0 and standard deviation $\sigma/\sqrt{n}$.

The form of the standard error $\sigma/\sqrt{n}$ is an important property of Monte Carlo methods. First, it tells us that if we increase the number of our samples by a factor of four, we will half the standard error. Second, standard error does not depend on the dimensionality of the integral. A Monte Carlo estimator based on *n* draws from the domain $[0,1]^d$ still have the form $\sigma/\sqrt{n}$ for all dimensions *d*. Most techniques of numerical integration such as the trapezoidal rule degrade in convergence rate with increasing dimensions.

We consider a Monte Carlo method to be useful in the domain of data stream analysis in which the number of data is overwhelming and the exact data distribution is unknown. The focus of our study is to generate samples from a stream data which is a prior step to data modeling and analysis. Once the samples have been successfully drawn, the characteristics of stream can be estimated. We concentrate on the sampling problem because it can provide a satisfactory estimation which will be proven through experimentations on cluster and classification analyses.

III. Sampling Method and Density Estimation

Basically the Monte Carlo method employs any technique of statistical sampling to approximate solutions to quantitative problems. With Monte Carlo method, a large system can be sampled in a number of random configurations, and that data can be used to describe the system as a whole. The efficiency of the method depends largely on the ability to draw samples effectively. For a particular domain of stream data, we consider the rejection sampling method. Rejection sampling, or acceptance-rejection sampling, is a sampling method first introduce by Von Neumann [16]. This method is used in cases where a target distribution, *f(x)*, is too complicate for us to sample from it directly.

Suppose we have a simpler distribution, *g(x)*, which we can evaluate and generate samples from, then the difficult sampling problem can be avoided by sampling from *g(x)* instead. By generating a uniform random variable *u* from the interval [0,1], we accept *x* if the condition $u \le f(x) / Cg(x)$ holds; otherwise reject the value of *x* and repeat the sampling step. Posing the restriction $Cg(x) \ge f(x)$ for some $C > 1$, we say that *Cg* envelopes *f*. The validation of this method is the envelope principle. When simulating the point (*x*, *v*) where $v = u*Cg(x)$, we produce a uniform simulation over the subgraph of $Cg(x)$. Accepting only points such that $u \le f(x) / Cg(x)$ then produces points (*x*, *v*) uniformly distributed over the subgraph of *f(x)* and thus, marginally, a simulation from *f(x)*.

Rejection sampling will work best if *g* is a good approximation to *f*. However, in a high-dimensional problem the value of *C* needs to be chosen very large to ensure the requirement *Cg(x) > f(x)*, for all *x*. The result is an enormous rejection rate.

The difficulty of applying rejection sampling method directly to the problem of data stream analysis is that we do not know beforehand where the modes of *f* are located or how high they are. In other words, we do not know the exact characteristics of the target density. We thus propose to apply the EM (Expectation-Maximization) technique [6] to approximate the density *f(x)*.

We consider multi-dimensional stream data as mixtures of Gaussian, or normal, probability density functions (pdf). Gaussian mixtures [8], [12] are combinations of Gaussian distributions written as

$$g(x) = \sum_{i=1}^{K} p_i f(x \mid \theta_i) \quad (6)$$

A random variable *x* denotes independent observation in *K* mixture components. The p_i's are the mixing proportions, $0 < p_i < 1$ for all $i = 1, \dots, K$, and $p_1 + \dots + p_K = 1$. The $f(x|\theta_i)$ denotes the density of a *d*-dimensional Gaussian distribution

with mean vector μ and covariance matrix Σ, that is θ = (μ,Σ), and the Gaussian pdf is given by [4], [15]

$$g_{(\mu,\Sigma)}(x) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\Sigma)}} \exp\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\} \quad (7)$$

By varying the number of Gaussians *K*, the mixing proportions p_i, and the parameter θ_i of each Gaussian density function, Gaussian mixtures can be used to describe any complex pdfs (Fig. 1).

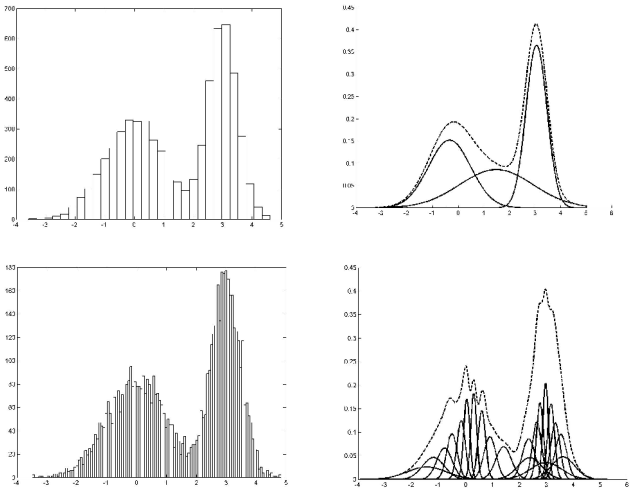


Fig. 1 one dimensional Gaussian mixture densities for K = 3 (first row) and K = 30 (second row). The left column shows the histogram of Gaussian density, the right column gives the corresponding Gaussian mixture pdf

In stream data a mixture density $p_i f(x|\theta_i)$ has been observed with unknown parameters θ_i and p_i. To find these parameters to optimally fit a mixture model for a given set of data, the EM algorithm [6], [12], [15] can be used. The EM algorithm is a broadly applicable approach to the iterative computation of maximum likelihood estimates. For a set of *iid* samples X = { x_1, ..., x_N }drawn from a data generation model $f_{(\mu,\Sigma)}(x_i)$, thus the resulting density for the samples is

$$\prod_{i=1}^{N} f_{(\mu,\Sigma)}(x_i) = L(\theta \mid x). \quad (8)$$

The likelihood function $L(\theta \mid x)$ is the likelihood of the parameters given the data. In the maximum likelihood problem, the goal is to find θ that maximizes L, that is $\arg\max_\theta L(\theta \mid X)$. In the Gaussian case, the computation of the exponential can be avoided by maximizing $log\ (L(\theta \mid x))$ instead of $L(\theta \mid x)$.

The EM algorithm is an approach to find the maximum of likelihood functions in incomplete data problems. Let X be observed data, Z be unobserved data, and Y = X∪Z be full data set. The probability distribution of Z depends on X and the unknown parameter θ. Given an initial parameter $\theta^{(0)}$, The EM algorithm produces a sequence $\{\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, ...\}$ that converges to a stationary point of the likelihood function.

IV. EMR Sampling

In our particular case of data stream analysis, we assume that the observed data have a normal distribution. Given a specific number of models, the EM algorithm is applied to estimate the mean of each model. These mean values have been scaled up to produce an upper bound for the underlying partially observed target density. The idea of the proposed method is illustrated in Fig. 2. The target function is represented as a one-dimensional 3-Gaussian mixtures (the three solid lines at the bottom of Fig. 2) from which we want to draw samples. The density $E(x)$ is estimated with the upper bound requirement that $E(x) > f(x)$ for all x. $E'(x)$ is the approximation (shown as a thick dash line in Fig. 2) of the unknown target density. A broad distance of E and E' (e.g., at x = 1) represents a rejecting area, whereas a narrow distance (e.g., at x = 6.5) is an acceptance one.

It should be noted that EM requires a pre-specified number of K components to be incorporated into the mixture models. According to our proposed method, a suitable number should be selected by a user. To cope with multi-dimensional problem, we propose to use a statistical method – principal component analysis (PCA) – to reduce the complicated problem to a simpler two-dimensional problem. That is, we take into account only the first and second major components of the data set. The two-dimensional data are used to train the EM algorithm to estimate parameters μ and Σ of the Gaussian mixture models. The estimated Gaussian pdf is a distribution E (as shown in Fig. 2). To sample from the estimated density we scale up this distribution to obtain an approximate E', which is a simpler distribution that we can evaluate and generate samples from. The outline of our EMR sampling algorithm is illustrated in Fig. 3. The subroutine *Density_Estimator* to approximate the density function has been shown in Fig. 4.

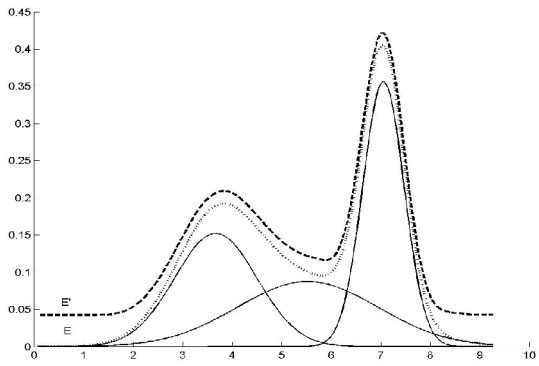


Fig. 2 EM-based rejection (EMR) sampling

From the estimated density E and the rough approximate E', we perform rejection sampling with the decision criteria $\{E(x)/(\sqrt{d} * E'(x))\} \geq u$, when u is a uniform variable distributed between 0 and 1, and d is a dimensionality of the data. The input from stream data has been taken one by one. The data item that satisfies the criteria will be included in the

sample until the specified sample size is completely filled up. Then the sampled data set is a representative of the whole stream. Any analysis methods can now be performed on this set.

Input: - a *d*-dimensional data set *D* with *N* points
- an integer *K* to specify the number of models, and
- a sample size *SS*

Output: - a sample set *S* drawn from the mixture models

// Data preprocessing steps //

1. If $d > 0$ then Apply PCA to obtain 1st and 2nd components
2. Transform *D* to a two-dimensional data set *X*

// Density estimation with EM and getting a rough pdf $E'(X)$ *//*

3. Set max_iteration = max{50, *d*K*}
4. ($E(X)$, $E'(X)$) = *Density_Estimator* (*X*, *K*, max_iteration)
5. Set count = 0
6. While count < *SS* *// Sampling steps //*
7. Sample *x* from $E(X)$
8. Generate *u* from U(0,1)
9. If $u \le E(x)/(\sqrt{d} * E'(x))$
   then Accept *x*, add it to *S*, and increment count
10. Return *S*

Fig. 3 EMR sampling algorithm

Density_Estimator (*X*, *K*, *max_iteration*)

1. Initialize parameter θ = (μ, Σ) for each of *K* Gaussian models by running K-means
2. Initialize the prior probabilities $P(m_k)$ of each model *m* to $1/K$, $k = 1,..., K$
3. Repeat
4. Compute the probability

$$P(m_k^{(i)} \mid x_n, \theta^{(i)}) = \frac{P(m_k^{(i)} \mid \theta^{(i)}) \bullet p(x_n \mid \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_j P(m_j^{(i)} \mid \theta^{(i)}) \bullet p(x_n \mid \mu_j^{(i)}, \Sigma_j^{(i)})}$$

5. Update means μ_k, variances Σ_k, and priors *P*

$$\mu_k^{(i+1)} = \frac{\sum_{n=1}^{N} x_n P(m_k^{(i)} \mid x_n, \theta^{(i)})}{\sum_{n=1}^{N} P(m_k^{(i)} \mid x_n, \theta^{(i)})}$$

$$\sum_k^{(i+1)} = \frac{\sum_{n=1}^{N} P(m_k^{(i)} \mid x_n, \theta^{(i)}) (x_n - \mu_k^{(i+1)})(x_n - \mu_k^{(i+1)})^T}{\sum_{n=1}^{N} P(m_k^{(i)} \mid x_n, \theta^{(i)})}$$

$$P(m_k^{(i+1)} \mid \theta^{(i+1)}) = \frac{1}{N} \sum_{n=1}^{N} P(m_k^{(i)} \mid x_n, \theta^{(i)})$$

6. Until the *max_iteration* has been reached or the joint likelihood of all data with respect to all the models is greater than the lower boundary criterion $CL(\theta)$

$$L(\theta) \ge CL(\theta) = \sum_{k=1}^{K} \sum_{n=1}^{N} P(m_k \mid x_n, \theta) \log p(x_n \mid \theta)$$

7. Return $\theta_i = (\mu_k, \Sigma_k)$ for $k = 1, ..., K$, and a rough $\theta_i' = (\mu^r{}_k, \Sigma^r{}_k)$ from *r* iterations, $r < 10$

Fig. 4 Density-Estimator algorithm

V. Experimentations

A. Evaluation of Density Estimator

The objective of our initial experiments is to empirically evaluate the closeness of the estimated density to the real one. The closeness is determined by comparing the Euclidean distance of the estimated mean vector $\hat{\mu}$ to the original mean vector μ, and comparing the estimated covariance matrix $\hat{\Sigma}$ to the original covariance matrix Σ. We use a synthetic data generator to produce two-dimensional Gaussian mixtures. The number of mixture models, number of points in each model, original mean vector and covariance matrix are input parameters.

We vary the number of models from 2 to 20 with 50 to 1,000 data points in each model. To properly initialize the component means for the θ-parameter learning, we find the approximate mean points by running max{50, *d*K*} iterations of k-means algorithm [17]. Component elements and main diagonal covariance matrix elements are also initialized accordingly, and off-diagonal matrix elements are constrained to zero. Some of our experimental results on the accuracy of our density estimator compared with the simple uniform sampling are illustrated in Table 1. The EMR sampling results are compared against the uniform sampling which always assumes a single Gaussian model. The efficiency of the sampling methods is evaluated on the basis of the closeness of the estimated $\theta_i = (\mu_i, \Sigma_i)$ to the original means and covariance matrices of the generative models. The μ-differences and Σ-differences are averaged from *K* models. The experimental results confirm the applicability of the EMR approach toward the problem of θ-parameter approximation. The estimated means and variances are very close to the original parameter values.

TABLE I
Experimental Results of EMR Sampling from Various Mixing of Gaussian Models

Number of Mixture Models	EMR Sampling		Uniform Sampling	
	μ-difference	Σ-difference	μ-difference	Σ-difference
2	0.000113	0.000901	0.088772	0.144793
6	0.000425	0.001527	0.090213	0.231109
8	0.000961	0.001599	0.098055	0.271645
12	0.000987	0.001938	0.200137	0.430098
14	0.001017	0.001991	0.299873	0.456131
16	0.001025	0.002007	0.300159	0.513772
18	0.001328	0.002031	0.330011	0.720001
20	0.001414	0.002508	0.460101	0.935644

B. Cluster and Classification Analyses

To verify the utility of the proposed method on the real-world data we run the k-means clustering algorithm [17] on various sampled data from the UCI repository [http://www.ics.uci.edu/~mlearn/ MLRepository.html]. We test our algorithm on four data sets: Wisconsin diagnostic breast cancer (466 data points, 2 classes), diabetes (512 data points, 2 classes), DNA (2000 data points, 3 classes), and satellite image (4435 data points, 6 classes). In each data set, we assume that the class labels are correct clusters to be found by the k-means algorithm. By assuming prior knowledge about known clusters, we can evaluate the error rate of the cluster learning.

On evaluation the efficiency of the Monte Carlo approach we simulate a data stream by generating several samples for each data set. In our experiments we observe the performance of cluster learning on increasing samples varied from 1%, 5%, 10%, 15%, ... ,50%, and the complete data set. The experimental results are shown in Fig. 5. The clustering results reveal the efficiency of the proposed method that only around 10-25% sampling size is sufficient for the accurate learning of data clusters.

The classification task has been performed on the same experimental setting with the C4.5 algorithm [17]. the results are shown in Fig. 6.

VI. Conclusion

In this paper we propose a technique of Monte Carlo estimation to analyze major characteristics of data stream. At the sampling phase of the Monte Carlo method we propose the EMR sampling algorithm to efficiently draw representative samples from data containing mixture models. We propose to apply the expectation-maximization technique to estimate the means and variances of the mixture models. The algorithm *Density_Estimator* produces two density functions, E and E'. The distance of E and E' at each sampling point is a decision criteria for either sample acceptance or rejection. A narrow distance among the two estimated densities tends to the acceptance case if the distance ratio is greater than the generated uniform random variable from the interval [0, 1].

The experimental results verify the utility of the proposed *Density_Estimator* algorithm and the EMR sampling method. The clustering and classification experimentations on real-world data also confirm the efficiency of our method. We plan to further our study on skewed data in which the distributions are not uniformly distributed.

References

[1] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for clustering evolving data streams," in *Pro. Very Large Data Bases*, 2003.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Model and issues in data stream systems," in *Pro. ACM PODS*, 2002.

[3] M. Berthold and D.J. Hand, *Intelligent Data Analysis: An Introduction*. Springer-Verlag, 2003.

[4] J. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," Dept. Electrical Engineering and Computer Science, University of California Berkeley, Technical Report TR-97-021, 1998.

[5] G. Coremode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically," in *Pro. ACM PODS*, 2003.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society B*, vol. 39, pp. 1-22, 1977.

[7] P. Domingos and G. Hulten, "A general method to scaling up machine learning algorithms and its application to clustering," in *Pro. ICML*, 2001.

[8] M. A. T. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, pp. 381–396, 2002.

[9] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data stream: A review," *SIGMOD Record*, vol. 34, pp. 18–26, 2005.

[10] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "One-pass wavelet decompositions of data streams," *IEEE Trans. Knowledge and Data Engineering*, vol. 15, pp. 541–554, 2003.

[11] D. Mackay, "Introduction to Monte Carlo," in *Learning in Graphical Models*, M. Jordan, Ed. MIT Press, 1996, pp. 175-204.

[12] J. M. Marin, K. Mengersen, and C. Robert, "Bayesian modelling and inference on mixtures of distributions," in *Handbook of Statistics*, vol. 25, Elsevier-Science, 2005.

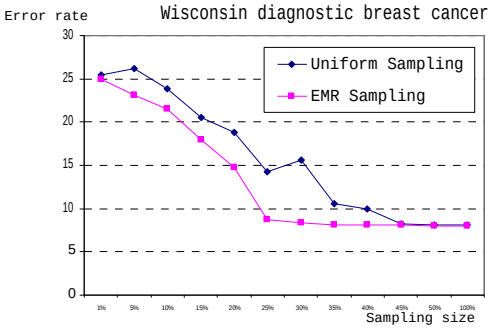
[13] S. Muthukrishnan, "Data streams: Algorithms and applications," in *Proc. ACM-SIAM Symposium on Discrete Algorithm*, 2003.

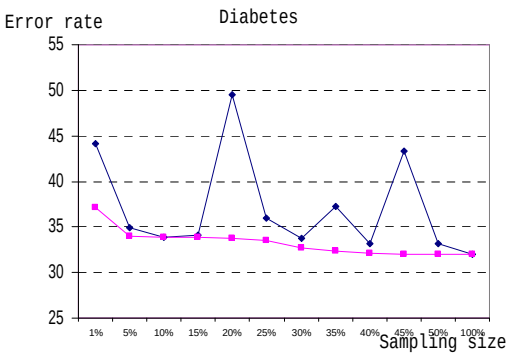
[14] R. Neal, "Probabilistic inference using Markov chain Monte Carlo methods," Dept. Computer Science, University of Toronto, Technical Report CRG-TR93-1, 1993.

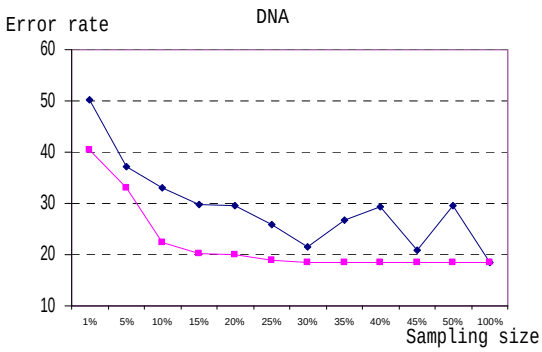
[15] B. Resch, "A tutorial for the course computational intelligence," Available: http://www.igi.tugraz.at/lehre/CI

[16] J. von Neumann, "Various techniques used in connection with random digits," *Applied Mathematics Series*, vol. 12, National Bureau of Standards, Washington, D.C., 1951.

[17] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.







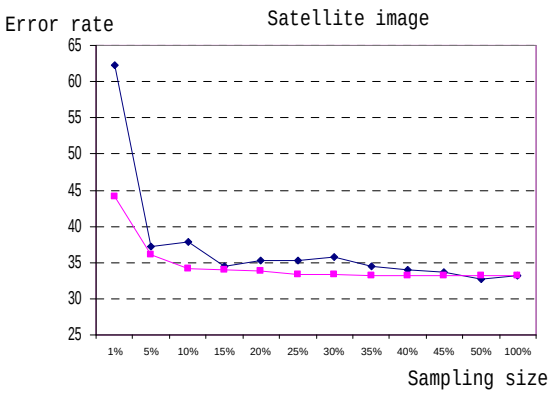
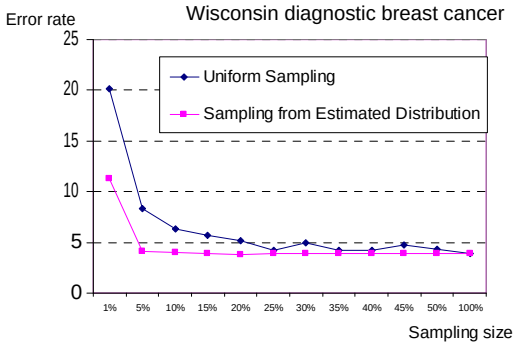
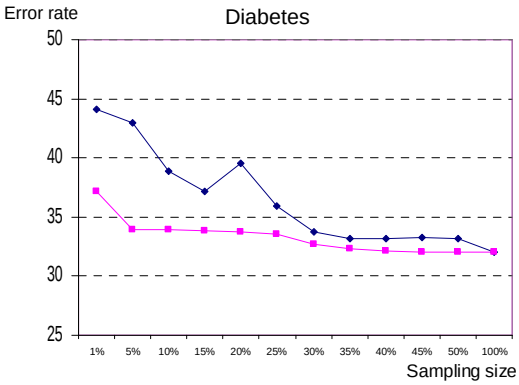
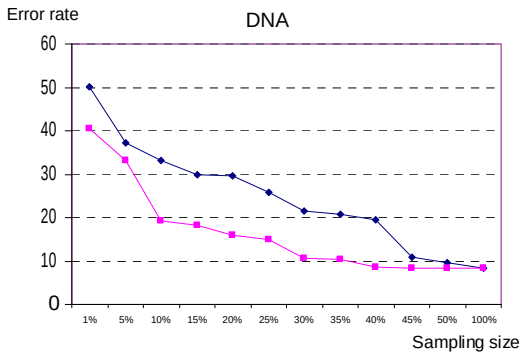


Fig. 5 Clustering results on four data sets







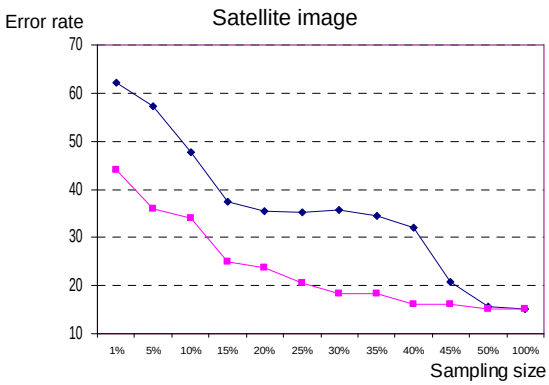


Fig. 6 Classification results on four data sets

ประวัติผู้วิจัย

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา ประเทศสหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของทบวงมหาวิทยาลัย (หรือสำนักงานคณะกรรมการการอุดมศึกษาในปัจจุบัน) โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง “Active database rule set reduction by knowledge discovery” หลังสำเร็จการศึกษาได้ปฏิบัติงานในตำแหน่งอาจารย์ ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ปัจจุบันดำรงตำแหน่งหัวหน้าหน่วยปฏิบัติการวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้ (Data Engineering and Knowledge Discovery Research Unit – DEKD) สำนักวิชาวิศวกรรมศาสตร์ ดำเนินการวิจัยประยุกต์เกี่ยวกับการออกแบบและพัฒนาระบบเหมืองข้อมูลประสิทธิภาพสูงที่สามารถทนต่อข้อมูลรบกวน และการวิจัยพื้นฐานเกี่ยวกับเทคนิคการจัดกลุ่มข้อมูล และเทคนิคการวิเคราะห์ข้อมูลขนาดใหญ่ด้วยฮิวริสติก โดยมีผลงานวิจัยตีพิมพ์ในวารสารวิชาการและเอกสารการประชุมวิชาการ จำนวนมากกว่า 30 เรื่อง ในสาขาระบบฐานความรู้ ฐานข้อมูลแอคทีฟ ฐานข้อมูลนิรนัย การทำเหมืองข้อมูลและการค้นหาความรู้