การลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นเพื่อการจัดกลุ่มข้อมูลขนาดใหญ่

นายธรรมศักดิ์ เธียรนิเวศน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2548

ISBN 974-533-521-5

A DENSITY-BASED DATA REDUCTION FOR CLUSTERING ON LARGE DATA SETS

Thammasak Thianniwet

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2005

ISBN 974-533-521-5

การลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นเพื่อการจัดกลุ่มข้อมูลขนาดใหญ่

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
ประธานกรรมการ

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

(รศ. ดร.นิตยา เกิดประสพ)
กรรมการ

(ผศ. ดร.คะชา ชาญศิลป์)
กรรมการ

(รศ. ดร.เสาวณีย์ รัตนพานี)
รองอธิการบดีฝ่ายวิชาการ

(รศ. น.อ. ดร.วรพจน์ ขำพิศ)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ธรรมศักดิ์ เธียรนิเวศน์ : การลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นเพื่อการจัดกลุ่มข้อมูลขนาดใหญ่ (A DENSITY-BASED DATA REDUCTION FOR CLUSTERING ON LARGE DATA SETS) อาจารย์ที่ปรึกษา : ผศ. ดร.กิตติศักดิ์ เกิดประสพ, 120 หน้า.
ISBN 974-533-521-5

กระบวนการจัดกลุ่มข้อมูลอัตโนมัติบนชุดข้อมูลที่มีขนาดใหญ่มากๆ เป็นกระบวนการที่ต้องใช้เวลาและสิ้นเปลืองหน่วยความจำเป็นจำนวนมาก การลดขนาดข้อมูลเป็นแนวทางหนึ่งที่จะช่วยแก้ปัญหานี้ได้ งานวิจัยนี้จึงมุ่งที่จะศึกษาค้นคว้า ตลอดจนพัฒนาวิธีการลดขนาดข้อมูลด้วยเทคนิคการสุ่มข้อมูลที่มีความทนทานต่อข้อมูลรบกวน สำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ที่มีการกระจายแบบไม่ปกติ และสามารถสร้างชุดข้อมูลสุ่มแบบต่อเนื่องได้ภายในการอ่านข้อมูลเพียงรอบเดียว จากการวิเคราะห์หาลักษณะเด่นของอัลกอริธึมสุ่มข้อมูล RVS, DBS, และ DBRVS ที่เคยมีนักวิจัยท่านอื่นเสนอไว้ พบว่าอัลกอริธึม DBS เป็นอัลกอริธึมที่มีความแม่นตรงสูง การสุ่มข้อมูลด้วยอัลกอริธึม DBS เพียง 2% สามารถให้ผลลัพธ์ของกลุ่มข้อมูลได้เทียบเท่ากับข้อมูลทั้งหมด อีกทั้งยังสามารถลดเวลาในการจัดกลุ่มข้อมูลได้มากกว่า 95% แต่เมื่อข้อมูลที่ใช้มีข้อมูลรบกวนปะปนอยู่ คุณภาพของการสุ่มข้อมูลของอัลกอริธึม DBS กลับลดลงอย่างเห็นได้ชัด ซึ่งแสดงให้เห็นว่าอัลกอริธึม DBS มีความอ่อนไหวต่อข้อมูลรบกวนสูงเช่นกัน

งานวิจัยนี้จึงได้เสนออัลกอริธึมสุ่มข้อมูล DBSPACE ซึ่งพัฒนาขึ้นเพื่อเพิ่มศักยภาพในการทนทานต่อข้อมูลรบกวน โดยการพิจารณาค่าความเป็นปึกแผ่นของข้อมูล ซึ่งบริเวณที่มีความน่าจะเป็นที่จะเป็นกลุ่มข้อมูลที่สนใจจะมีค่าความเป็นปึกแผ่นของข้อมูลสูงกว่าบริเวณที่มีความน่าจะเป็นที่จะเป็นข้อมูลรบกวน

จากผลการวิจัยพบว่า อัลกอริธึม DBSPACE สามารถให้ผลลัพธ์ที่ดีเทียบเท่ากับอัลกอริธึม DBS ในกรณีที่ข้อมูลปราศจากข้อมูลรบกวน และสามารถให้ผลลัพธ์ที่ดีกว่า ในกรณีที่ข้อมูลมีข้อมูลรบกวนปะปนอยู่

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา _____

ปีการศึกษา 2548 ลายมือชื่ออาจารย์ที่ปรึกษา _____

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม _____

THAMMASAK THIANNIWET : A DENSITY-BASED DATA REDUCTION FOR CLUSTERING ON LARGE DATA SETS. THESIS ADVISOR : ASST. PROF. KITTISAK KERDPRASOP, Ph.D., 120 PP. ISBN 974-533-521-5

DATA MINING/DATA REDUCTION/SAMPLING/CLUSTERING/ DENSITY-BIASED/COMPACTNESS

Determining clusters in large data sets takes a very long time and consumes many resources. Data reduction is an important step to increase the efficiency of determining clusters in large data sets. Our work is intended to examine and develop the appropriate sampling technique as a data reduction scheme for clustering that requires only a single data set scan. From the experimental results performed on three sampling algorithms (RVS, DBS, and DBRVS), we found that DBS is the most accurate sampling algorithm. A 2% DBS sample of the original data set can produce the same result as the whole original data set and also help reduce time to find the clusters by over 95%. However, it shows sensitivity on a noisy data set.

Our research is intended to propose the DBSPACE algorithm which is developed to gain the potential of noise tolerance by considering the compactness within the region of data. The region with more compactness will be a good area from which representative sample should be drawn.

The results of this research showed that, DBSPACE sample can produce the result as accurate as DBS sample drawn from clean data sets. It can produce the better result while the original data set is surrounded by many noises.

School of Computer Engineering

Academic Year 2005

Student's Signature

Advisor's Signature

Co-advisor's Signature

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณ บุคคลดังต่อไปนี้ ที่ได้กรุณาให้คำปรึกษา แนะนำ ช่วยเหลือ อย่างดียิ่ง ทั้งในด้านวิชาการ และ ด้านการดำเนินงานวิจัย

- ผู้ช่วยศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ, อาจารย์ที่ปรึกษาวิทยานิพนธ์
- รองศาสตราจารย์ ดร.นิตยา เกิดประสพ, อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
- ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์ และผู้ช่วยศาสตราจารย์ ดร.คะชา ชาญศิลป์ ที่กรุณาให้คำแนะนำในการเรียบเรียงวิทยานิพนธ์ฉบับนี้
- ขอบคุณ C. Palmer ที่กรุณาเปิดเผยซอร์สโค้ดของอัลกอริธึมสุ่มข้อมูล DBS ซึ่งทำให้งานวิจัยนี้ดำเนินไปได้อย่างถูกต้อง ครบถ้วน และสมบูรณ์
- ขอบคุณ J. Handl ที่กรุณาเปิดเผยซอร์สโค้ดของอัลกอริธึมสร้างคลัสเตอร์สังเคราะห์ ซึ่งใช้เป็นเครื่องมือที่ใช้สร้างชุดข้อมูลสำหรับทดสอบประสิทธิภาพของแต่ละอัลกอริธึม
- ขอบคุณ นางสาวลักษมี โขมโนทัย และ นายธนินทร์ ระเบียบโพธิ์ ที่ช่วยตรวจทานวิทยานิพนธ์ฉบับนี้ และขอขอบคุณเพื่อนนักศึกษาบัณฑิตศึกษาทุกท่านที่ให้กำลังใจและให้การสนับสนุนงานวิจัยนี้เป็นอย่างดี อันมีส่วนทำให้วิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์

ท้ายนี้ ขอกราบขอบพระคุณบิดา มารดา ที่ให้การเลี้ยงดูอบรมและส่งเสริมการศึกษาเป็นอย่างดีมาโดยตลอด จนทำให้ผู้วิจัยประสบความสำเร็จในชีวิตตลอดมา

ธรรมศักดิ์ เธียรนิเวศน์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในยุคแห่งข้อมูลข่าวสารของโลกปัจจุบัน การทำธุรกิจต่างต้องแข่งขันกันสร้างผลกำไรโดยอาศัยเทคโนโลยีและความรู้ต่างๆ การทำเหมืองข้อมูล (data mining) เป็นศาสตร์แขนงหนึ่งที่มีจุดมุ่งหมายเพื่อค้นหาองค์ความรู้ที่มีอยู่บนฐานข้อมูลซึ่งเก็บรวบรวมไว้ในหลากหลายรูปแบบ โดยมีจุดมุ่งหมายเพื่อนำความรู้ที่อาจเป็นประโยชน์เหล่านั้นไปประยุกต์ใช้ให้เกิดประโยชน์เพื่อบรรลุถึงเป้าหมายของแต่ละองค์กรและเพื่อประกอบการตัดสินใจในการดำเนินธุรกิจต่อไป ซึ่งการทำเหมืองข้อมูลนั้นมีเทคนิคและวิธีการต่างๆ กันมากมาย โดยในแต่ละวิธีมีจุดมุ่งหมายเพื่อให้ได้มาซึ่งคำตอบของคำถามที่แตกต่างกัน

การทำเหมืองข้อมูลแบ่งออกเป็นกลุ่มกว้างๆ ตามลักษณะการนำไปใช้งานได้สองกลุ่มใหญ่ๆ คือ การทำเหมืองข้อมูลเพื่อการทำนาย (predictive data mining) ซึ่งเป็นการนำเอาความรู้ที่ได้จากการประมวลผลข้อมูลเพื่อใช้ในการทำนายข้อมูลหรือเหตุการณ์ที่จะเกิดขึ้นในอนาคต เช่น การทำเหมืองข้อมูลสำหรับการวินิจฉัยโรค จากข้อมูลการวินิจฉัยโรคของแพทย์ผู้เชี่ยวชาญซึ่งจำแนกชนิดของโรคตามลักษณะอาการของผู้ป่วย โปรแกรมทำเหมืองข้อมูลจะเรียนรู้จากข้อมูลดังกล่าวและสร้างโมเดลที่สามารถจำแนกชนิดของโรคต่างๆ จากอาการของผู้ป่วยได้ และเมื่อมีผู้ป่วยรายใหม่ต้องการตรวจวินิจฉัยโรคก็สามารถนำโมเดลที่สร้างขึ้นมาช่วยในการวินิจฉัยได้ การทำเหมืองข้อมูลอีกกลุ่มหนึ่งคือการทำเหมืองข้อมูลเพื่อการอธิบาย (descriptive data mining) ซึ่งเป็นการค้นหารูปแบบของกลุ่มข้อมูลที่มีความสัมพันธ์กัน หรือลักษณะของความสัมพันธ์ร่วมกันของข้อมูล โดยมิได้เจาะจงเพื่อหารูปแบบหรือโมเดลของข้อมูลอย่างหนึ่งอย่างใดเพื่อการทำนายเท่านั้น แต่เป็นการค้นหาทุกรูปแบบที่น่าสนใจ โดยรูปแบบหรือความรู้ต่างๆที่ซ่อนอยู่ในข้อมูลเหล่านั้นอาจเป็นองค์ความรู้ใหม่ที่เป็นประโยชน์ซึ่งถูกค้นพบอย่างไม่คาดคิดมาก่อนก็เป็นได้

การจัดกลุ่มข้อมูล (clustering) เป็นอีกสาขาหนึ่งของงานทางด้านการทำเหมืองข้อมูลเพื่อการอธิบาย โดยการหารูปแบบที่สัมพันธ์กันของข้อมูลที่มีการรวมกลุ่มกัน จุดประสงค์และเป้าหมายสำคัญของการจัดกลุ่มข้อมูลคือการค้นหาและจำแนกลักษณะเฉพาะของข้อมูลออกเป็นแต่ละกลุ่มหรือที่เรียกว่าคลัสเตอร์ (cluster) โดยมุ่งเน้นให้ข้อมูลที่อยู่กลุ่มเดียวกันมีความคล้ายคลึง

กันมากที่สุด และในขณะเดียวกัน ข้อมูลที่อยู่ต่างกลุ่มกันจะต้องมีความคล้ายคลึงกันน้อยที่สุดหรือแตกต่างกันมากที่สุดนั่นเอง

การจัดกลุ่มข้อมูลนิยมใช้เพื่อการวิเคราะห์ข้อมูลเบื้องต้นสำหรับการทำเหมืองข้อมูลด้วยเทคนิคอื่นๆ ต่อไป ตัวอย่างเช่น การวิเคราะห์สายพันธุ์ของไวรัส ในขั้นแรกจะต้องทำการจัดกลุ่มไวรัสจากข้อมูลไวรัสที่มีอยู่แต่ละตัวเพื่อหาว่าไวรัสตัวไหนบ้างที่ควรจัดให้อยู่กลุ่มเดียวกันโดยใช้อัลกอริธึมจัดกลุ่มข้อมูล (clustering algorithm) จากนั้นจึงนำข้อมูลการแบ่งกลุ่มไวรัสที่ได้มาทำการสร้างโมเดลเพื่อจำแนกไวรัสออกเป็นแต่ละกลุ่ม โดยในขั้นนี้เป็นการนำผลการวิเคราะห์ข้อมูลเบื้องต้นมาทำเหมืองข้อมูลอีกขั้นหนึ่ง เมื่อมีไวรัสตัวใหม่ถูกพบก็สามารถนำเอาโมเดลที่สร้างขึ้นนั้นมาประกอบการวิเคราะห์ถึงสายพันธุ์ กลุ่ม ตลอดจนพฤติกรรมของไวรัสที่พบได้ ซึ่งถ้าหากไม่สามารถจำแนกได้ว่าไวรัสตัวนั้นอยู่ในกลุ่มใด นั่นอาจหมายถึงการค้นพบไวรัสสายพันธุ์ใหม่ก็เป็นได้ ตัวอย่างการใช้ประโยชน์อื่นๆของการจัดกลุ่มข้อมูลเช่น การจำแนกกลุ่มของลูกค้าตามรูปแบบหรือพฤติกรรมการซื้อสินค้า การจัดกลุ่มเว็บเพจ การจัดกลุ่มยีนส์และโปรตีนที่มีลักษณะการทำงานคล้ายคลึงกัน การค้นหากลุ่มบริเวณที่มีแนวโน้มที่จะเกิดแผ่นดินไหวจากข้อมูลการเกิดแผ่นดินไหวเป็นต้น

แต่อย่างไรก็ตาม การจัดกลุ่มข้อมูลนั้นเป็นงานที่ต้องใช้ทั้งเทคนิคและวิธีการที่ซับซ้อน เนื่องจากจะต้องค้นหารูปแบบทั้งหมดที่เป็นไปได้ อีกทั้งข้อมูลที่ใช้มักมีขนาดใหญ่มาก จึงทำให้การค้นหากลุ่มข้อมูลทำได้ยากเพราะเป็นงานที่ต้องใช้ทั้งเวลาในการประมวลผลและสิ้นเปลืองหน่วยความจำเป็นจำนวนมาก ด้วยเหตุนี้จึงทำให้นักวิจัยจำนวนมากคิดค้นหาเทคนิคและวิธีการต่างๆ มากมายเพื่อปรับปรุงกระบวนการจัดกลุ่มข้อมูลให้มีประสิทธิภาพดียิ่งขึ้น การลดขนาดข้อมูล (data reduction) ด้วยเทคนิคที่แม่นยำและมีประสิทธิภาพเป็นอีกแนวทางหนึ่งเพื่อจัด การกับปัญหาดังกล่าว หลักการที่สำคัญของการลดขนาดข้อมูลคือการทำให้ข้อมูลตั้งต้นมีขนาดลดลงโดยสูญเสียลักษณะสำคัญของข้อมูลน้อยที่สุด เนื่องจากข้อมูลแต่ละตัวจะมีความสำคัญต่อการจัดกลุ่มข้อมูลไม่เท่ากัน ด้วยเทคนิคการเลือกข้อมูลที่ดีจะทำให้สามารถเลือกข้อมูลที่มีความสำคัญและสามารถใช้เป็นตัวแทนของข้อมูลส่วนใหญ่ได้ ข้อมูลที่มีการรวมกลุ่มกันอย่างหนาแน่นจะเป็นข้อมูลที่มีความสำคัญต่อการจัดกลุ่มข้อมูลในอนาคต ในทางกลับกันข้อมูลที่ไม่มีการรวมกลุ่มกันและมีการกระจายตัวต่างไปจากกลุ่มหรือที่เรียกว่าเอ้าท์ไลเออร์ (outlier) จะต้องถูกขจัดออกไป เพราะข้อมูลเหล่านี้จะมีผลกระทบต่อความแม่นตรงของกระบวนการจัดกลุ่มข้อมูลอัตโนมัติ นอกจากนี้ในกลุ่มข้อมูลยังอาจมีข้อมูลรบกวน (noise) ที่อยู่ในรูปของข้อมูลผิดพลาดปะปนอยู่ การลดขนาดข้อมูลลงโดยการกำจัดข้อมูลรบกวนทิ้งไป และเลือกเฉพาะข้อมูลที่สามารถเป็นตัวแทนที่ดีของข้อมูลส่วนใหญ่ไว้ จึงเป็นแนวทางหนึ่งเพื่อให้การจัดกลุ่มข้อมูลบนฐานข้อมูลขนาดใหญ่เป็นไปได้โดยราบรื่น

ดังนั้นงานวิจัยนี้จึงมุ่งที่จะศึกษาค้นคว้า ตลอดจนพัฒนาเทคนิคการลดขนาดข้อมูลที่เหมาะสมเพื่องานการจัดกลุ่มข้อมูลขนาดใหญ่ โดยการพิจารณาการรวมกลุ่มกันของข้อมูลหรือความหนาแน่นของข้อมูลเป็นหลัก โดยข้อมูลที่ใช้จริงอาจมีการกระจายของข้อมูลแบบไม่ปกติจึงทำให้ไม่สามารถใช้เทคนิคการลดขนาดข้อมูลแบบสม่ำเสมอทั่วไปได้ ฉะนั้นวัตถุประสงค์ของงานวิจัยนี้คือเพื่อพัฒนาเทคนิคการลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นที่มีความทนทานต่อข้อมูลรบกวนสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ที่มีการกระจายแบบไม่ปกติ

1.2 วัตถุประสงค์การวิจัย

1.2.1 เพื่อศึกษาเทคนิคการลดขนาดข้อมูลด้วยวิธีการสุ่มข้อมูล การลดขนาดข้อมูลแบ่งออกเป็นหลายประเภท สำหรับงานวิจัยนี้จะเน้นการลดขนาดข้อมูลด้วยวิธีการสุ่มข้อมูลเป็นหลัก

1.2.2 เพื่อศึกษาเกณฑ์และเทคนิคที่มีผลต่อความแม่นตรงในการเลือกข้อมูล เช่น การคำนวณค่าความหนาแน่น ค่าความน่าจะเป็นที่ข้อมูลใดๆ จะถูกเลือก เป็นต้น

1.2.3 เพื่อเปรียบเทียบและค้นหาเทคนิคการสุ่มที่เหมาะสมสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ โดยจะพิจารณาเวลาที่ใช้ในการสุ่มข้อมูล จำนวนหน่วยความจำที่ใช้ เวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลจากชุดข้อมูลสุ่ม ตลอดจนความแม่นตรงของคลัสเตอร์ที่พบ

1.2.4 เพื่อค้นหาเทคนิคที่จะช่วยเพิ่มประสิทธิภาพในการเลือกข้อมูลสุ่มอย่างแม่นยำสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ได้ โดยเมื่อนำชุดข้อมูลสุ่มที่ได้เข้าสู่กระบวนการค้นหากลุ่มข้อมูลจะต้องสามารถให้ผลลัพธ์ที่ถูกต้องสูงกว่าหรือใกล้เคียงกับผลลัพธ์จากชุดข้อมูลตั้งต้นมากที่สุด

1.2.5 เพื่อพัฒนาเทคนิคการลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นที่มีความทนทานต่อข้อมูลรบกวน สำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ที่มีการกระจายแบบไม่ปกติ

1.2.6 สามารถนำเทคนิคการลดขนาดข้อมูลที่พัฒนาขึ้น มาใช้เพื่อการเตรียมข้อมูลได้อย่างมีประสิทธิภาพ และให้ผลลัพธ์ของคลัสเตอร์ที่มีความถูกต้องสูงกว่าหรือใกล้เคียงกับผลลัพธ์จากชุดข้อมูลตั้งต้นมากที่สุด

1.3 ขอบเขตการวิจัย

งานวิจัยนี้มุ่งเน้นที่จะศึกษาการลดขนาดข้อมูลด้วยวิธีการสุ่มข้อมูลในแบบต่างๆ ซึ่งอาจสรุปลักษณะสำคัญของเทคนิคการสุ่มข้อมูลที่สนใจสำหรับงานวิจัยนี้ได้ดังต่อไปนี้

1.3.1 Sequential Sampling with Unknown Population Size คือ สามารถสุ่มข้อมูลแบบต่อเนื่องโดยที่ไม่ทราบจำนวนข้อมูลทั้งหมดล่วงหน้าก่อนการสุ่ม

1.3.2 Single Data Set Scan คือ อ่านข้อมูลเพียงแค่หนึ่งรอบเท่านั้น การอ่านข้อมูลขนาดใหญ่ในแต่ละรอบจะทำให้สิ้นเปลืองทั้งเวลาและหน่วยความจำเป็นจำนวนมาก ดังนั้นการจำกัดการอ่านข้อมูลให้อยู่เพียงหนึ่งรอบจึงเป็นแนวทางในการลดปัญหาดังกล่าวได้

1.3.3 Non-Uniform Distribution สามารถรองรับข้อมูลที่มีการกระจายแบบไม่ปกติได้ ซึ่งข้อมูลที่เป็นจริงอาจมีการกระจายตัวแบบ zipf คือเป็นข้อมูลที่ประกอบด้วยคลัส-เตอร์ที่ทั้งขนาดและความหนาแน่นแตกต่างกันมาก

1.3.4 High Accurate คือ มีความแม่นตรงสูง สามารถเลือกข้อมูลที่สามารถเป็นตัวแทนที่ดีของแต่ละคลัสเตอร์ได้ เพื่อให้อัลกอริธึมจัดกลุ่มข้อมูลสามารถพบคลัสเตอร์มากที่สุดเมื่อเทียบกับขนาดของชุดข้อมูลสุ่ม (%)

1.3.5 Noise Tolerance คือ ความสามารถในการทนทานต่อข้อมูลรบกวน กล่าวคือสามารถกำจัดข้อมูลรบกวน รวมถึงการค้นหาข้อมูลที่เป็น outlier อันเป็นข้อมูลที่มีผลกระทบต่อคุณภาพของคลัสเตอร์ได้

1.3.6 Efficient Resource and Memory Usage ใช้ทรัพยากรและหน่วยความจำอย่างมีประสิทธิภาพ

1.3.7 Time Reduction สามารถช่วยลดเวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลลงได้

ในงานวิจัยนี้จะทำการศึกษาเทคนิคการสุ่มข้อมูลโดยจะทดสอบด้วยชุดข้อมูลสังเคราะห์จำนวนหนึ่ง ซึ่งมีการกระจายของข้อมูลแบบไม่ปกติ โดยเกณฑ์ในการพิจารณาความเหมาะสมของแต่ละเทคนิคประกอบด้วย เวลาที่ใช้ในการสุ่มข้อมูล (time to sample) จำนวนหน่วยความจำที่ใช้ (memory usage) เวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลจากชุดข้อมูลสุ่ม (time to cluster) ตลอดจนความแม่นตรงของคลัสเตอร์ที่พบ *(number of cluster found*: *NC)* หรือจำนวนคลัสเตอร์ที่ถูกต้องและสามารถยอมรับได้จากชุดข้อมูลสุ่ม อีกทั้งยังจะศึกษาถึงความทนทานต่อข้อมูลรบกวนของแต่ละเทคนิคโดยการทดสอบกับชุดข้อมูลที่ถูกสร้างข้อมูลรบกวนเข้าไปในปริมาณต่างๆ กัน ในการทดลองจะใช้ *k*-means ซึ่งเป็นอัลกอริธึมจัดกลุ่มข้อมูลที่เป็นที่นิยมในการค้นหาคลัสเตอร์บนชุดข้อมูล ซึ่งอัลกอริธึมที่ใช้พัฒนาขึ้นบนระบบเปิดเผยซอร์สโค้ดที่ชื่อ WEKA (Frank, Hall, Holmes, Martin, Mayo, Pfahringer, Smith and Witten, 2005)

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ทำให้เกิดการพัฒนาเทคนิคการลดขนาดข้อมูลที่มีประสิทธิภาพและมีความแม่นตรงสูงขึ้น

1.4.2 สามารถนำเทคนิคที่พัฒนาขึ้นไปใช้ในขั้นตอนการลดขนาดข้อมูลได้

1.4.3 ทำให้การจัดกลุ่มข้อมูลบนฐานข้อมูลขนาดใหญ่มากเป็นไปได้ด้วยการใช้ทรัพยากรที่จำกัด

1.4.4 ทำให้การจัดกลุ่มข้อมูลใช้เวลาน้อยลง คือ สามารถลดเวลาที่ใช้ในกระบวนการทั้งหมดลงได้ แม้จะต้องสูญเสียเวลาส่วนหนึ่งไปในขั้นตอนการลดขนาดข้อมูลก็ตาม ซึ่งถือว่าเป็นส่วนน้อยเมื่อเทียบกับเวลาที่สามารถลดลงได้

1.4.5 เพิ่มความแม่นยำให้กับอัลกอริธึมจัดกลุ่มข้อมูลได้ เนื่องจากการเลือกเฉพาะข้อมูลที่สำคัญและตัดข้อมูลรบกวนที่มีผลกระทบต่อความแม่นตรงของการจัดกลุ่มข้อมูลทิ้งไป

1.4.6 สามารถใช้เทคนิคการลดขนาดข้อมูลที่ได้พัฒนาขึ้นกับข้อมูลที่ไม่ทราบจำนวนล่วงหน้า และข้อมูลที่มีการรับเข้ามาอย่างต่อเนื่องได้

1.4.7 สามารถใช้เทคนิคการลดขนาดข้อมูลที่ได้พัฒนาขึ้นกับข้อมูลที่มีการกระจายแบบไม่ปกติ ซึ่งเป็นลักษณะของข้อมูลที่ใช้จริงได้

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการนำเสนอวรรณกรรมและงานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ โดยในหัวข้อที่ 2.1 จะกล่าวถึงเทคนิคและวิธีการจัดกลุ่มข้อมูลในแบบต่างๆ ที่สำคัญ อันเป็นการให้ข้อมูลเบื้องต้นและสร้างความคุ้นเคยกับงานการจัดกลุ่มข้อมูล ในหัวข้อที่ 2.2 จะกล่าวถึงเทคนิคและวิธีการลดขนาดข้อมูลซึ่งเป็นการเตรียมข้อมูลให้มีขนาดที่เหมาะสม และเพื่อเป็นการเพิ่มประสิทธิภาพสำหรับงานการจัดกลุ่มข้อมูล ในหัวข้อที่ 2.3 เป็นการให้คำจำกัดความและเทคนิคที่ใช้เพื่อการระบุข้อมูลรบกวนและเอ้าท์ไลเออร์ซึ่งเป็นข้อมูลที่จะส่งผลกระทบต่อคุณภาพของคลัส-เตอร์ และในหัวข้อที่ 2.4 กล่าวถึงเทคนิคการลดขนาดข้อมูลด้วยวิธีการสุ่มข้อมูลซึ่งออกแบบมาเพื่องานการจัดกลุ่มข้อมูลโดยเฉพาะ เนื่องจากงานการจัดกลุ่มข้อมูลมีวัตถุประสงค์หลักเพื่อค้นหาและจัดกลุ่มของข้อมูลที่มีความใกล้ชิดกัน โดยกลุ่มข้อมูลอาจมีขนาดที่แตกต่างกัน ดังนั้นการใช้เทคนิคการลดขนาดข้อมูลที่เหมาะสมจึงจะสามารถให้ผลลัพธ์ของชุดข้อมูลที่มีคุณภาพ โดยชุดข้อมูลที่มีคุณภาพจะต้องสามารถใช้เป็นตัวแทนของชุดข้อมูลทั้งหมดสำหรับงานการจัดกลุ่มข้อมูลได้อย่างถูกต้องและสมบูรณ์

2.1 การจัดกลุ่มข้อมูล (Clustering Methods)

การจัดกลุ่มข้อมูลอัตโนมัติเป็นงานที่ต้องใช้เทคนิคที่ซับซ้อนเพื่อค้นหากลุ่มข้อมูลได้อย่างแม่นยำ ซึ่งกระบวนการจำเป็นต้องใช้ทั้งเวลาในการประมวลผลและสิ้นเปลืองหน่วยความจำจำนวนมาก อีกทั้งแต่ละเทคนิคยังมีข้อจำกัดที่แตกต่างกันออกไป ดังนั้นจึงมีผู้สนใจศึกษาและพัฒนาเทคนิคการจัดกลุ่มข้อมูลอย่างกว้างขวาง เนื่องจากอัลกอริธึมจัดกลุ่มข้อมูลมีอยู่มากมาย การเลือกใช้อัลกอริธึมที่เหมาะสมกับวัตถุประสงค์ของงานนั้นจึงเป็นสิ่งสำคัญ อัลกอริธึมจัดกลุ่มข้อมูลสามารถแบ่งออกเป็นประเภทต่างๆ ได้ดังนี้

2.1.1 Partitioning Methods

บนฐานข้อมูลจำนวน n เรคคอร์ด การจัดกลุ่มข้อมูลประเภทนี้จะทำการสร้าง k พาร์-ทิชัน โดยแต่ละพาร์ทิชันจะแสดงถึงข้อมูลที่ถูกแบ่งออกเป็นกลุ่ม (โดยที่ $k \leq n$) ในแต่ละกลุ่มจะประกอบไปด้วยข้อมูลอย่างน้อยที่สุด 1 เรคคอร์ด และข้อมูลแต่ละเรคคอร์ดจะต้องถูกจัดให้อยู่ในกลุ่มข้อมูลใดเพียงกลุ่มเดียวเท่านั้น (สำหรับบางเทคนิคอาจอนุโลมให้เรคคอร์ดใดๆ สามารถถูกจัด

ให้อยู่ในกลุ่มข้อมูลได้มากกว่า 1 กลุ่ม)

การจัดกลุ่มข้อมูลประเภทนี้จะต้องระบุค่า k หรือจำนวนพาร์ทิชันที่ต้องการ โดยกระบวนการจัดกลุ่มเริ่มจากการสร้างพาร์ทิชันตั้งต้น จากนั้นอัลกอริธึมจะทำการวนซ้ำเพื่อปรับพาร์ทิชันให้เหมาะสมโดยการย้ายเรคคอร์ดหรืออ็อบเจกต์จากกลุ่มหนึ่งไปยังอีกกลุ่มหนึ่งที่มีความเหมาะสมกว่า โดยที่พาร์ทิชันที่ดีจะต้องสามารถแบ่งให้ข้อมูลที่มีความใกล้ชิดกันหรือมีความสัมพันธ์กันอยู่ในพาร์ทิชันเดียวกัน ส่วนข้อมูลที่มีความแตกต่างกันจะต้องถูกจัดให้อยู่ในพาร์-ทิชันที่ไกลออกไป ซึ่งแต่ละอัลกอริธึมจะใช้เทคนิคในการพิจารณาความคล้ายคลึงกันของข้อมูลของแต่ละคลัสเตอร์แตกต่างกันออกไป เทคนิคที่เป็นที่รู้จักได้แก่

***k*-means algorithm** (Han and Kamber, 2001) ใช้ค่าเฉลี่ยของอ็อบเจกต์ที่ถูกจัดให้อยู่ในกลุ่มเดียวกันเป็นตัวแทนของทุกอ็อบเจกต์ในกลุ่มนั้น อัลกอริธึมเริ่มต้นจากการรับค่าพารามิเตอร์ k ซึ่งคือจำนวนคลัสเตอร์ที่ต้องการค้นหา จากนั้นอัลกอริธึมจะทำการสุ่มเลือกอ็อบ-เจกต์เริ่มต้นจำนวน k อ็อบเจกต์ ซึ่งแต่ละอ็อบเจกต์แสดงถึงตัวแทนของแต่ละคลัสเตอร์(ค่าเฉลี่ยหรือจุดศูนย์กลางของคลัสเตอร์หรือเซนทรอยด์: centroid) จากนั้นจะทำการจัดกลุ่มให้กับอ็อบ-เจกต์ที่เหลือ อ็อบเจกต์จะถูกจัดให้อยู่ในคลัสเตอร์เดียวกันเมื่ออ็อบเจกต์นั้นมีความคล้ายกับตัวแทนของคลัสเตอร์นั้นมากที่สุด จากนั้นจึงทำการคำนวณหาค่าเฉลี่ยของคลัสเตอร์ใหม่ และดำเนินกระบวนการเดียวกันกับอ็อบเจกต์ที่เหลือต่อไปจนกระทั่งทุกอ็อบเจกต์ถูกจัดกลุ่มอย่างสมบูรณ์และอ็อบเจกต์ไม่มีการเปลี่ยนกลุ่มอีกต่อไป คุณภาพของการจัดกลุ่มข้อมูลสามารถคำนวณได้จากค่า squared-error ดังสมการ

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

โดยที่ E แสดง sum of squared-error ของทุกอ็อบเจกต์ ให้ p แทนอ็อบเจกต์ใดๆ ของแต่ละคลัส-เตอร์ และ m_i คือตัวแทนหรือค่าเฉลี่ยของคลัสเตอร์ C_i โดยขั้นตอนของอัลกอริธึม k-means สามารถสรุปได้ดังรูปที่ 2.1

อัลกอริธึม k-means จะทำการค้นหา k คลัสเตอร์โดยพยายามทำให้ค่า squared-error มีค่าน้อยที่สุด ซึ่งอัลกอริธึมจะทำงานได้ดีเมื่อคลัสเตอร์มีการกระจายตัวออกเป็นกลุ่มอย่างชัดเจน อีกทั้งยังสามารถรองรับกับข้อมูลที่มีขนาดใหญ่ได้อย่างมีประสิทธิภาพเนื่องจากมีความซับซ้อนของอัลกอริธึม (computational complexity) เพียง $O(nkt)$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด k คือจำนวนคลัสเตอร์ และ t คือจำนวนรอบของการวนซ้ำ ซึ่งโดยปกติค่า $k \ll n$ และ $t \ll n$ ดังนั้นเวลาส่วนใหญ่จึงขึ้นตรงกับขนาดของชุดข้อมูลเป็นสำคัญ

Algorithm: *k*-means. The *k*-means algorithm for partitioning based on the mean value of the object in the cluster.
Input: The number of cluster *k* and a database containing *n* objects.
Output: A set of *k* clusters that minimizes the squared-error criterion.
Method:
(1) arbitrarily choose *k* objects as the initial cluster center;
(2) repeat
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
(5) until no change;

รูปที่ 2.1 อัลกอริธึม *k*-means (Han and Kamber, 2001, p.349)

k-means สามารถใช้งานได้ดีตราบใดที่สามารถหาค่าเฉลี่ยของคลัสเตอร์ได้ แต่ในบางกรณีซึ่งข้อมูลประกอบด้วยข้อมูลเชิงอักขระ จึงอาจต้องปรับปรุงวิธีการคำนวณหาค่าเฉลี่ยและวิธีการพิจารณาความคล้ายคลึงกันของอ็อบเจกต์เพื่อความเหมาะสม การที่ต้องระบุค่า k ให้กับอัลกอริธึมอาจทำได้ยากเมื่อผู้ใช้ไม่มีความรู้พื้นฐานเกี่ยวกับข้อมูลที่ใช้ การระบุค่า k ที่ไม่เหมาะสมอาจทำให้กระบวนการจัดกลุ่มข้อมูลต้องใช้เวลานานขึ้น อีกทั้งยังอาจได้คลัสเตอร์ที่ไม่มีคุณภาพอีกด้วย และเนื่องจากการพิจารณาความใกล้ชิดกันของอ็อบเจกต์กับคลัสเตอร์โดยการวัดระยะห่างระหว่างอ็อบเจกต์กับตัวแทนคลัสเตอร์ทำให้คลัสเตอร์ที่พบมีรูปทรงกลม (spherical-shape) ซึ่งเทคนิคนี้อาจไม่เหมาะสำหรับการค้นหาคลัสเตอร์ที่มีรูปทรงอื่นๆ (arbitrary-shape, non-convex shape) อีกทั้งยังมีความอ่อนไหวต่อข้อมูลรบกวน เนื่องจากข้อมูลรบกวนเป็นข้อมูลที่มีลักษณะต่างจากข้อมูลอื่น (มีค่าสูงหรือต่ำกว่าข้อมูลปกติ) ซึ่งอาจส่งผลให้การคำนวณค่าเฉลี่ยของอ็อบเจกต์ในคลัสเตอร์ผิดเพี้ยนได้

***k*-medoids algorithm** เป็นเทคนิคที่พัฒนาขึ้นต่อจาก *k*-means เนื่องจาก *k*-means มีความอ่อนไหวต่อข้อมูลรบกวน ดังนั้นอัลกอริธึม *k*-medoids จึงถูกออกแบบมาเพื่อขจัดข้อด้อยดังกล่าว โดยการใช้อ็อบเจกต์ที่อยู่ใกล้กับจุดกึ่งกลางของคลัสเตอร์มากที่สุดเป็นตัวแทนของทุกอ็อบเจกต์ในคลัสเตอร์นั้นแทนการใช้ค่าเฉลี่ยของคลัสเตอร์ โดยที่กระบวนการจัดกลุ่มจะคล้ายกับ *k*-means คือการจัดกลุ่มข้อมูลโดยพิจารณาความคล้ายคลึงกันของอ็อบเจกต์กับตัวแทนของคลัสเตอร์ แต่ต่างกันเพียงวิธีการหาตัวแทนของคลัสเตอร์เท่านั้น PAM (Partitioning Around Medoids) (Kaufman and Rousseeuw, 1989) เป็นอัลกอริธึมที่ใช้เทคนิค *k*-medoids โดยที่ PAM มีความซับซ้อนของอัลกอริธึม $O(k(n-k)^2)$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด k คือจำนวนคลัสเตอร์ โดย

ขั้นตอนของอัลกอริธึม *k*-medoids สามารถสรุปได้ดังรูปที่ 2.2

Algorithm: *k*-medoids. A typecal *k*-medoids algorithm for partitioning based on medoid or central objects.
Input: The number of cluster *k* and a database containing *n* objects.
Output: A set of *k* clusters that minimizes the sum of the dissimilarities of all the objects to their nearest medoid.
Method:
(1) arbitrarily choose *k* objects as the initial medoids;
(2) repeat
(3) assign each remaining object to the cluster with the nearest medoid;
(4) randomly select a nonmedoid object, o_{random};
(5) compute the total cost, S, of swapping o_j with o_{random};
(6) if S<0 then swap o_j with o_{random} to form the new set of *k* medoids;
(7) until no change;

รูปที่ 2.2 อัลกอริธึม *k*-medoids (Han and Kamber, 2001, p.353)

เทคนิค *k*-means และ *k*-medoids สามารถใช้งานได้ดีกับข้อมูลที่มีขนาดเล็กถึงปานกลางที่มีการกระจายของข้อมูลเป็นแบบทรงกลมเท่านั้น สำหรับการค้นหากลุ่มข้อมูลที่มีรูปทรงที่ซับซ้อนจำเป็นจะต้องอาศัยเทคนิคอื่นๆ เพิ่มเติม *k*-medoids เป็นเทคนิคที่ทนทานต่อข้อมูลรบกวนได้ดีกว่า *k*-means แต่อย่างไรก็ตาม *k*-medoids จำเป็นต้องใช้การคำนวณมากกว่า *k*-means ซึ่งทั้งสองเทคนิคต่างก็มีข้อจำกัดคือจำเป็นต้องให้ผู้ใช้ระบุจำนวนคลัสเตอร์ที่ต้องการค้นหา (*k*)

CLARA (Clustering LARge Application) พัฒนาขึ้นโดย Kaufman and Rousseeuw (1989) เพื่อการจัดกลุ่มข้อมูลบนฐานข้อมูลขนาดใหญ่ โดยอาศัยเทคนิคการลดขนาดข้อมูลด้วยวิธีการสุ่มซึ่งเป็นการเพิ่มประสิทธิภาพของกระบวนการจัดกลุ่มข้อมูลและเพิ่มขีดความสามารถในการรองรับกับข้อมูลจำนวนมากได้ โดยอัลกอริธึมเริ่มต้นจากการสร้างชุดข้อมูลสุ่ม จากนั้นจึงใช้ PAM เพื่อจัดกลุ่มข้อมูลบนชุดข้อมูลสุ่มนั้นแทนชุดข้อมูลทั้งหมด โดย CLARA จะทำการสร้างชุดข้อมูลสุ่มและใช้ PAM จัดกลุ่มข้อมูลบนชุดข้อมูลสุ่มนั้นซ้ำๆ เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด ซึ่งความซับซ้อนของอัลกอริธึมในแต่ละรอบคือ $O(ks^2 + k(n-k))$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด k คือจำนวนคลัสเตอร์ และ s คือขนาดของชุดข้อมูลสุ่มที่สร้างขึ้น และจากการที่ CLARA จัดกลุ่มข้อมูลจากชุดข้อมูลสุ่ม คุณภาพของคลัสเตอร์จึงขึ้นอยู่กับคุณภาพของชุดข้อมูลสุ่มที่ใช้ด้วย

CLARANS (Clustering Large Application based upon RANdomized Search) พัฒนาขึ้นโดย Ng and Han (1994) เป็นการรวมเทคนิคการสุ่มข้อมูลเข้ากับ PAM แต่ไม่เหมือนกับ

CLARA ตรงที่ CLARANS ไม่ได้ใช้เฉพาะชุดข้อมูลสุ่มเพียงชุดใดชุดหนึ่งในการค้นหาแต่ละครั้ง แต่ CLARANS จะผสานเทคนิคการสุ่มในแต่ละขั้นตอนของการค้นหา ซึ่งต่างจาก CLARA ที่ใช้เพียงชุดข้อมูลสุ่มเดียวตลอดทั้งกระบวนการ ความซับซ้อนของอัลกอริธึม CLARANS เท่ากับ $O(n^2)$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด ซึ่งมีค่าค่อนข้างสูง แต่อย่างไรก็ตาม CLARANS สามารถปรับปรุงประสิทธิภาพได้โดยใช้เทคนิค R*-trees ช่วยในการค้นหาข้อมูลให้มีประสิทธิภาพมากยิ่งขึ้น

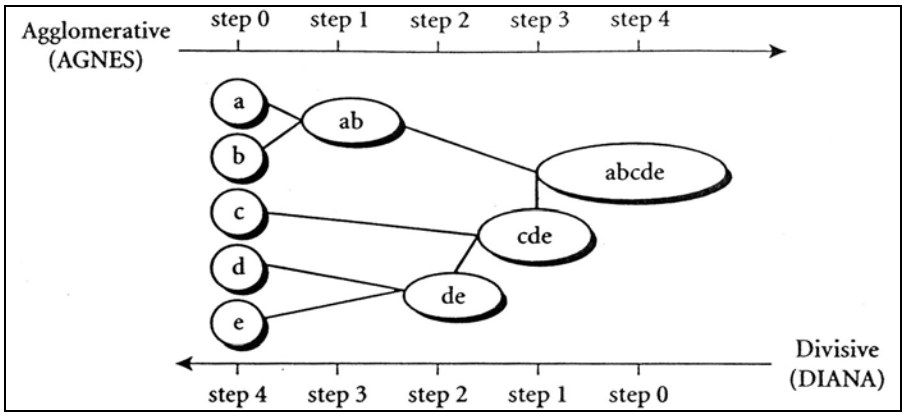
2.1.2 Hierarchical Methods

เป็นวิธีการจัดกลุ่มข้อมูลประเภทหนึ่งที่อาศัยหลักการแบ่งข้อมูลออกเป็นลำดับขั้นคล้ายกับต้นไม้ ซึ่งวิธีการแบ่งกลุ่มข้อมูลแบบนี้สามารถแบ่งออกเป็น 2 แนวทางตามลักษณะการสร้างลำดับขั้นคือ agglomerative approach กับ divisive approach

Agglomerative approach หรือ bottom-up approach เริ่มต้นโดยการให้แต่ละอ็อบเจกต์อยู่ในคลัสเตอร์ที่ต่างกัน จากนั้นจึงทำการวนซ้ำเพื่อรวมคลัสเตอร์ที่ใกล้ชิดกันเข้าด้วยกันจนกระทั่งทุกคลัสเตอร์ถูกรวมเข้าเป็นคลัสเตอร์เดียว หรือเมื่อเข้าเงื่อนไขการสิ้นสุดการค้นหา อัลกอริธึมจัดกลุ่มข้อมูลแบบลำดับขั้นส่วนใหญ่จะใช้แนวทางนี้ในการจัดกลุ่มข้อมูล แต่จะต่างกันที่เทคนิคและวิธีการคำนวณค่าความคล้ายคลึงกันระหว่างคลัสเตอร์ (intercluster similarity)

Divisive approach หรือ top-down approach เริ่มต้นโดยให้ทุกอ็อบเจกต์อยู่ในคลัสเตอร์เดียวกัน จากนั้นจึงค่อยๆ แบ่งคลัสเตอร์ออกเป็นคลัสเตอร์ที่เล็กลงมาเรื่อยๆ จนกระทั่งทุกอ็อบเจกต์ถูกแยกออกจากกันทั้งหมดหรือเมื่อเข้าเงื่อนไขการสิ้นสุดการค้นหา เช่นจำนวนของคลัสเตอร์ที่ได้ เป็นต้น

ในการแบ่งหรือรวมคลัสเตอร์แต่ละครั้งจะอาศัยการพิจารณาบนพื้นฐานของสิ่งที่ได้เรียนรู้ขณะปัจจุบันเป็นสำคัญ โดยเมื่อตัดสินใจที่จะรวมหรือแบ่งคลัสเตอร์ใดแล้วจะไม่สามารถย้อนกลับมาแก้ไขได้อีก ทำให้การตัดสินใจในแต่ละรอบใช้การคำนวณเพียงเล็กน้อยเนื่องจากไม่จำเป็นต้องพิจารณาทุกทางเลือก เพียงแต่เลือกตัดสินใจในสิ่งที่ดีที่สุด ณ ขณะนั้นเป็นสำคัญ แต่อย่างไรก็ตามเนื่องจากการไม่สามารถย้อนกลับมาแก้ไขการตัดสินใจที่ผิดพลาดได้ การตัดสินใจในแต่ละครั้งจึงต้องพิจารณาอย่างรอบคอบที่สุด รูปที่ 2.3 แสดงถึงการทำงานของ AGNES (Agglomerative NESting) ซึ่งเป็น agglomerative hierarchical clustering และ DIANA (Divisive ANAlysis) ซึ่งเป็น divisive hierarchical clustering บนชุดข้อมูลที่ประกอบด้วย 5 อ็อบเจกต์ {a, b, c, d, e}



รูปที่ 2.3 การจัดกลุ่มข้อมูลโดยใช้ AGNES และ DIANA (Han and Kamber, 2001, p.355)

โดยที่ AGNES จะเริ่มต้นจากการให้แต่ละอ็อบเจกต์ a, b, c, d, และ e อยู่ในคลัส-เตอร์ที่ต่างกัน ([a], [b], [c], [d], และ [e] ตามลำดับ) จากนั้นจึงทำการวนซ้ำเพื่อรวมคลัสเตอร์ที่มีความใกล้ชิดกันเข้าด้วยกัน ดังตัวอย่างในรูปที่ 2.3 AGNES จะทำการรวมคลัสเตอร์ [a] กับ [b] เข้าเป็นคลัสเตอร์ [ab] และรวมคลัสเตอร์ [d] กับ [e] เข้าเป็นคลัสเตอร์ [de] จากนั้นจึงรวมคลัสเตอร์ [c] กับ [de] เข้าเป็นคลัสเตอร์ [cde] และสุดท้ายคือทุกคลัสเตอร์ย่อยถูกรวมเข้าเป็นคลัสเตอร์เดียวกันทั้งหมด ([abcde]) ส่วน DIANA จะทำงานในทิศทางตรงกันข้ามกับ AGNES โดยจะเริ่มต้นจากการให้ทุกอ็อบเจกต์อยู่ในคลัสเตอร์เดียวกันทั้งหมด จากนั้นจึงทำการวนซ้ำเพื่อแบ่งคลัส-เตอร์ออกเป็นคลัสเตอร์ย่อยจนกระทั่งทุกอ็อบเจกต์ถูกแบ่งให้อยู่ต่างคลัสเตอร์กันทั้งหมด ดังตัวอย่างในรูปที่ 2.3 DIANA จะทำการแบ่งคลัสเตอร์ [abcde] ออกเป็นคลัสเตอร์ [ab] กับ [cde] และแบ่งคลัสเตอร์ [cde] ออกเป็นคลัสเตอร์ [c] กับ [de] และแบ่งคลัสเตอร์ [de] ออกเป็นคลัส-เตอร์ [d] กับ [e] และสุดท้ายคลัสเตอร์ [ab] จะถูกแบ่งออกเป็นคลัสเตอร์ [a] กับ [b] ตามลำดับ อัลกอริธึมจัดกลุ่มข้อมูลแบบลำดับขั้นที่เป็นที่รู้จักได้แก่

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) โดย Zhang, Ramakrishnan and Livny (1996) เป็นอัลกอริธึมที่ถูกออกแบบมาเพื่อให้สามารถทำงานได้บนหน่วยความจำที่จำกัด โดยที่ BIRCH จะใช้วิธีการค้นหาคลัสเตอร์บนข้อมูลสรุปที่เก็บไว้ในโครงสร้างต้นไม้สมดุล (hieght-balanced CF-tree: clustering feature tree) ที่ถูกเก็บไว้ในหน่วยความจำ โดยในแต่ละโหนดของ CF-tree ประกอบด้วยข้อมูลที่สำคัญสามส่วนคือ N, $\vec{LS}$, และ $\vec{SS}$ โดยที่ N คือจำนวนของอ็อบเจกต์ที่ถูกจัดให้อยู่ในโหนดนั้น $\vec{LS}$ คือผลรวมของอ็อบเจกต์จำนวน N และ $\vec{SS}$ คือผลรวมกำลังสองของอ็อบเจกต์จำนวน N ซึ่งสามารถอธิบายได้ด้วยเวกเตอร์

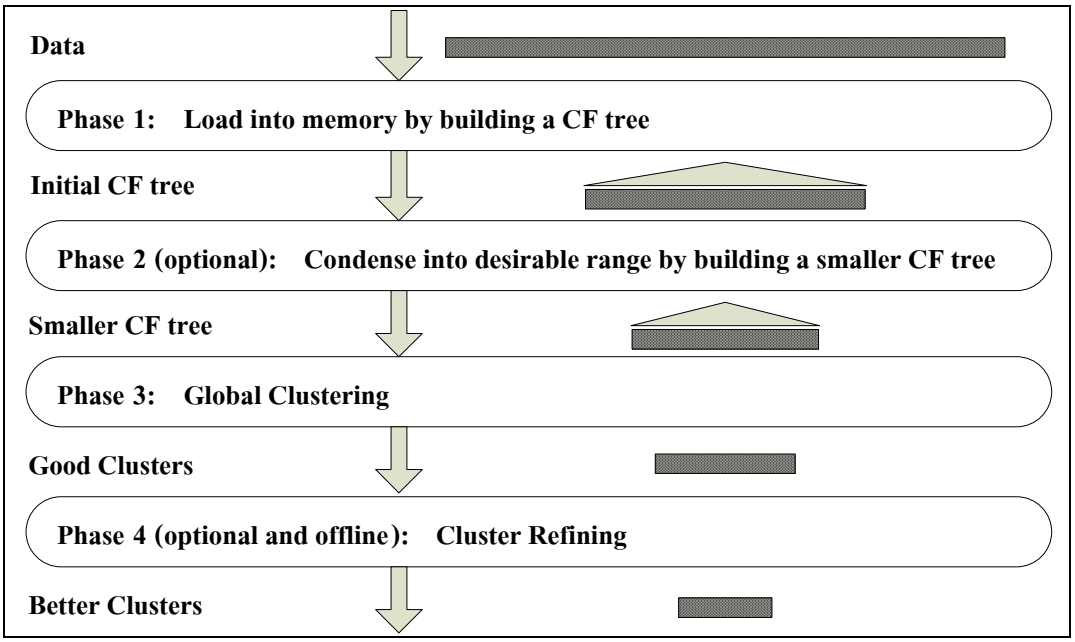
$$CF = (N, \vec{LS}, \vec{SS}) \quad | \{ \vec{LS} = \sum_{i=1}^{N} \vec{X}_i \ , \ \vec{SS} = \sum_{i=1}^{N} \vec{X}_i^{\,2} \}$$

โดยเมื่อมีการรวมกันของโหนดใบ เวกเตอร์ CF สามารถคำนวณใหม่ได้อย่างง่ายดายด้วยการนำเวกเตอร์ทั้งสองมารวมกัน วิธีการดังกล่าวทำให้ BIRCH สามารถทำงานแบบต่อเนื่องได้ (incremental) ทั้งในกรณีที่มีการรับข้อมูลใหม่เข้ามาและเมื่อต้องการรวมแต่ละคลัสเตอร์เข้าด้วยกัน

$$CF_1 + CF_2 = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, \vec{SS}_1 + \vec{SS}_2)$$

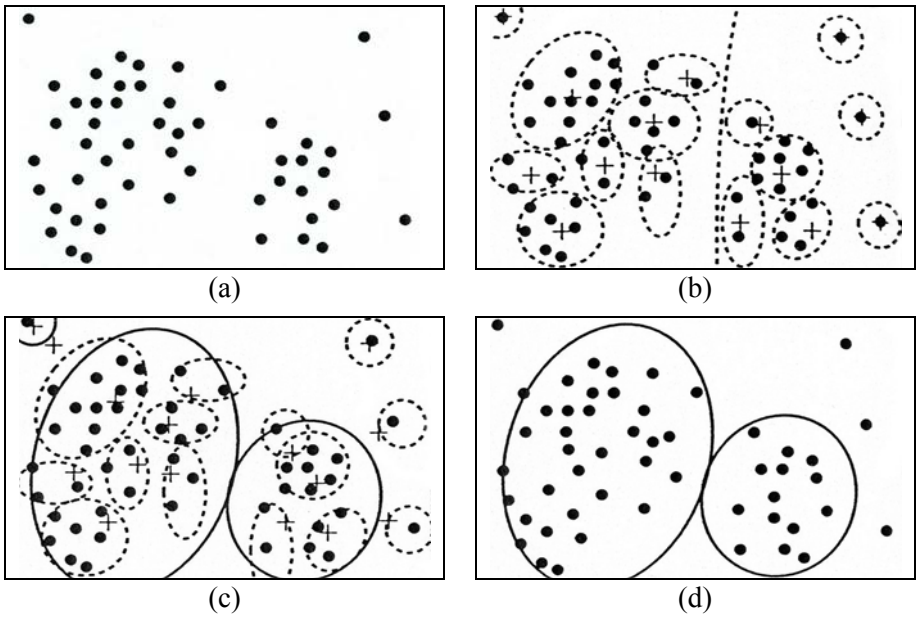
CF-tree จะประกอบด้วยค่า threshold ที่สำคัญคือค่า T ซึ่งเป็นค่าความผิดพลาดสูงสุดประจำแต่ละโหนดที่สามารถยอมรับได้ แต่ละอ็อบเจกต์จะถูกจัดให้อยู่ในโหนดใบโดยที่ค่าความผิดพลาดของโหนดใบนั้นต้องไม่เกินกว่าค่า T ที่กำหนด ค่า T ที่มากจะทำให้ได้ CF-tree ที่มีขนาดเล็กลงและใช้หน่วยความจำน้อยลง แต่จะเป็นการสรุปข้อมูลในระดับที่สูงขึ้นซึ่งอาจส่งผลต่อคุณภาพของคลัสเตอร์ที่ได้อีกด้วย ขั้นตอนของอัลกอริธึม BIRCH สามารถสรุปได้ดังรูปที่ 2.4 ซึ่งแสดงขั้นตอนการทำงานของอัลกอริธึมทั้ง 4 ขั้นตอน เริ่มต้นจากการอ่านข้อมูลทั้งหมดเพื่อทำการสร้าง CF-tree ซึ่งถือเป็นการจัดกลุ่มข้อมูลย่อย (preclustering) ก่อนขั้นตอนการจัดกลุ่มจริง ขั้นตอนที่สองเป็นขั้นตอนการลดขนาด CF-tree ที่จัดเก็บไว้บนหน่วยความจำเรียบร้อยแล้ว โดยในขั้นนี้จะมีการขจัดเอ้าท์ไลเออร์และข้อมูลรบกวนออกไปด้วย ขั้นตอนต่อมาเป็นขั้นตอนการค้นหาคลัสเตอร์ซึ่งสามารถเลือกใช้อัลกอริธึมจัดกลุ่มข้อมูลแบบใดก็ได้ที่สามารถรองรับการจัดกลุ่มข้อมูลจากข้อมูลสรุปบน CF-tree โดยจะทำการจัดกลุ่มให้กับโหนดใบของ CF-tree (แต่ละโหนดใบของ CF-tree สามารถใช้เป็นตัวแทนของข้อมูล N อ็อบเจกต์) ส่วนขั้นตอนสุดท้ายเป็นการจัดแต่ละอ็อบเจกต์ให้กับคลัสเตอร์โดยพิจารณาจากความใกล้ชิดกันกับคลัสเตอร์นั้นมากที่สุด

BIRCH สามารถทำงานได้รวดเร็วบนหน่วยความจำที่จำกัด โดยทั้งกระบวนการของ BIRCH ใช้การอ่านข้อมูลจากแฟ้มข้อมูลทั้งหมดเพียงรอบเดียวเท่านั้น (เฉพาะขั้นตอนการสร้าง CF-tree) จากนั้นจะเป็นการจัดกลุ่มข้อมูลจาก CF-tree ที่อยู่บนหน่วยความจำหลัก ความซับซ้อนของอัลกอริธึม BIRCH มีค่าเท่ากับ $O(n)$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด แต่อย่างไรตามอัลกอริธึมยังไม่สามารถค้นหาคลัสเตอร์ที่มีรูปร่างที่ซับซ้อนได้ด้วย



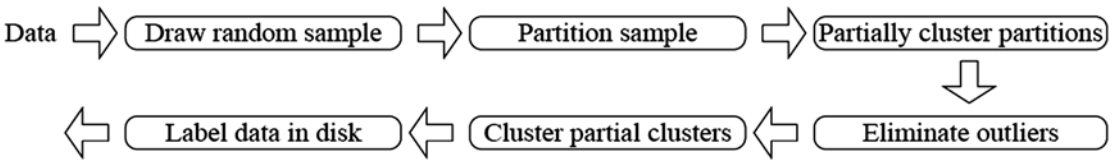
รูปที่ 2.4 ขั้นตอนโดยสรุปของอัลกอริธึม BIRCH (Zhang et al., 1996)

CURE (Clustering Using REpresentatives) โดย Guha, Rastogi and Shim (1998) เป็นอัลกอริธึมจัดกลุ่มข้อมูลแบบลำดับขั้นอีกอัลกอริธึมหนึ่ง คุณสมบัติเด่นของ CURE คือสามารถค้นหาคลัสเตอร์ที่มีรูปร่างที่ซับซ้อนท่ามกลางข้อมูลที่มีเอาท์ไลเออร์ได้ หลักการที่สำคัญของอัลกอริธึมคือ การค้นหาคลัสเตอร์จากชุดข้อมูลสุ่ม (ไม่ใช้ข้อมูลทั้งหมด) โดยการแบ่งข้อมูลออกเป็นพาร์ทิชันย่อยๆ แล้วทำการค้นหาคลัสเตอร์ย่อยๆ ในแต่ละพาร์ทิชันและกำจัดเอ้าท์ไลเออร์ออกไป จากนั้นจึงทำการค้นหาคลัสเตอร์จากตัวแทนของแต่ละคลัสเตอร์ย่อยในแต่ละพาร์ทิชันเพื่อเป็นการสร้างคลัสเตอร์ที่สมบูรณ์อีกทีหนึ่ง โดยในขั้นตอนนี้จะทำการรวมคลัสเตอร์ที่มีความใกล้ชิดกันเข้าด้วยกัน และเลื่อนตัวแทนของคลัสเตอร์เดิมเข้าหาจุดกึ่งกลางของคลัสเตอร์ใหม่โดยที่ยังใช้ตัวแทนของคลัสเตอร์เดิมนั้นเป็นตัวแทนของคลัสเตอร์ใหม่ด้วย (ใช้ตัวแทนของคลัสเตอร์ หลายตัวแทน การใช้จุดศูนย์กลางของคลัสเตอร์เพียงจุดเดียว) ด้วยเทคนิคดังกล่าวทำให้ CURE สามารถค้นหาคลัสเตอร์ที่มีรูปร่างในแบบต่างๆ ได้ เพื่อความเข้าใจยิ่งขึ้นสามารถพิจารณาจากรูปที่ 2.5 ซึ่งแสดงตัวอย่างของการค้นหาคลัสเตอร์ในแต่ละขั้นตอน โดยรูปที่ 2.5(a) แสดงชุดข้อมูลสุ่มที่ได้ รูปที่ 2.5(b) ข้อมูลถูกแบ่งและค้นหาคลัสเตอร์ย่อยในแต่ละพาร์ทิชัน โดยที่ “+” แสดงตัวแทนของคลัสเตอร์ย่อย รูปที่ 2.5(c) คลัสเตอร์ย่อยถูกนำมาจัดกลุ่มอีกครั้ง โดยตัวแทนเดิมของแต่ละ คลัสเตอร์ย่อยจะถูกเลื่อนเข้าหาจุดกึ่งกลางของคลัสเตอร์ใหม่ด้วยค่า shrinking factor (α) และเส้นทึบในรูปที่ 2.5(e) แสดงขอบเขตของคลัสเตอร์ที่สมบูรณ์



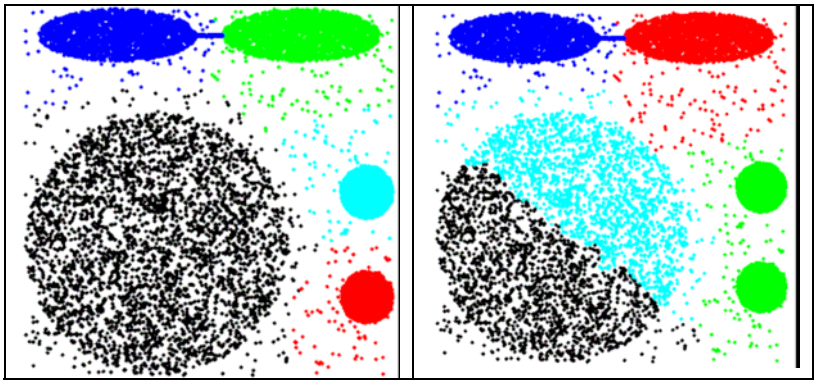
รูปที่ 2.5 การค้นหาคลัสเตอร์ของอัลกอริธึม CURE (Han and Kamber, 2001, p.360)

เนื่องจาก CURE ทำการค้นหาคลัสเตอร์จากชุดข้อมูลสุ่ม คุณภาพของข้อมูลสุ่มที่ใช้จึงจำเป็นต้องพิจารณาอย่างรอบคอบ โดย CURE ใช้วิธีการสุ่มข้อมูลแบบ random sampling โดยใช้หลักการของ Vitter (1985) ซึ่งเป็นอัลกอริธึมสุ่มข้อมูลที่เป็นที่รู้จักดีและสามารถสุ่มข้อมูลได้โดยการอ่านข้อมูลเพียงแค่รอบเดียว (รายละเอียดของอัลกอริธึมจะได้กล่าวต่อไปในหัวข้อที่ 2.4.1) แต่ปัญหาที่สำคัญคือขนาดของชุดข้อมูลสุ่มที่เหมาะสมควรเป็นเท่าไรจึงจะทำให้ชุดข้อมูลสุ่มที่ได้สามารถแสดงถึงคลัสเตอร์ได้มากที่สุด ซึ่ง CURE ใช้ Chernoff bounds (Motwani and Raghavan, 1995) ในการวิเคราะห์ว่าขนาดของชุดข้อมูลสุ่มที่เหมาะสมควรเป็นเท่าไรจึงจะทำให้ความน่าจะเป็นของการที่จะไม่พบคลัสเตอร์มีค่าน้อยที่สุด ขั้นตอนโดยสรุปของอัลกอริธึม CURE สามารถสรุปได้ดังรูปที่ 2.6



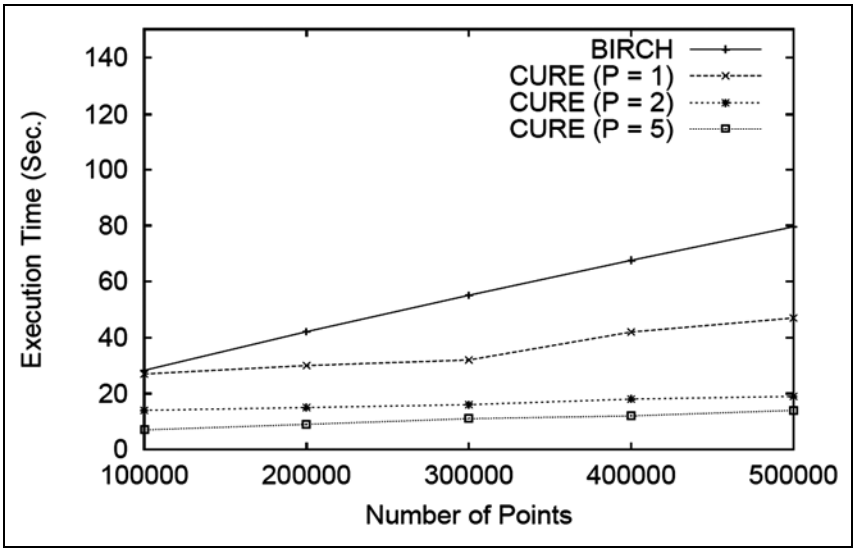
รูปที่ 2.6 ขั้นตอนโดยสรุปของอัลกอริธึม CURE (Guha et al., 1998)

จากการทดลองของ Guha et al. (1998) พบว่า CURE สามารถค้นหาคลัสเตอร์ได้ถูกต้องสูงกว่า BIRCH อีกทั้งยังใช้เวลาในการค้นหาคลัสเตอร์น้อยกว่าอีกด้วย (ดังภาพ 2.7 และ 2.8) โดยความซับซ้อนของอัลกอริธึม CURE มีค่าเท่ากับ $O(n^2 \log n)$ เมื่อ n คือจำนวนอ็อบเจกต์ของชุดข้อมูลสุ่มที่ใช้ ในกรณีที่ข้อมูลมีมิติจำนวนน้อยค่าความซับซ้อนของอัลกอริธึมสามารถลดลงเหลือ $O(n^2)$



(a) CURE (b) BIRCH

รูปที่ 2.7 ผลการค้นหาคลัสเตอร์ของ CURE และ BIRCH (Guha et al., 1998)



รูปที่ 2.8 เวลาที่ใช้ในการค้นหาคลัสเตอร์ของ CURE และ BIRCH (Guha et al., 1998)

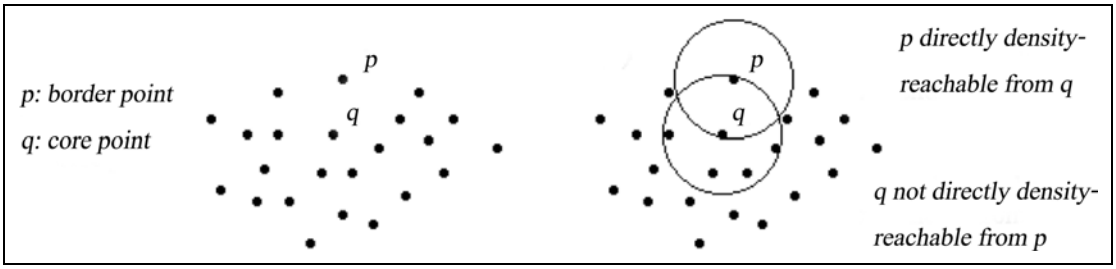
2.1.3 Density-Based Methods

เทคนิคการจัดกลุ่มข้อมูลโดยใช้วิธีการแบ่งข้อมูลเป็นพาร์ทิชันส่วนใหญ่มักใช้การหาความสัมพันธ์ของแต่ละอ็อบเจกต์โดยการพิจารณาระยะทางหรือ distance ของอ็อบเจกต์เหล่านั้น ทำให้เทคนิคดังกล่าวมีข้อจำกัดคือสามารถค้นหาได้เฉพาะคลัสเตอร์ที่มีรูปทรงกลมเท่านั้น ด้วยข้อจำกัดดังกล่าวจึงนำไปสู่การพัฒนาเทคนิคใหม่ซึ่งพิจารณาความหนาแน่นของข้อมูลเป็นเกณฑ์ในการค้นหาคลัสเตอร์ หลักการทั่วไปของเทคนิคนี้คือการแผ่ขยายขอบเขตของคลัสเตอร์ไปเรื่อยๆ ตราบใดที่ความหนาแน่นของอ็อบเจกต์มีค่าถึงค่าที่กำหนด นั่นคือแต่ละอ็อบเจกต์ของคลัส-เตอร์ใดๆ จะต้องประกอบด้วยอ็อบเจกต์ซึ่งอยู่ใกล้กันภายในรัศมีที่กำหนด (neighborhood) จำนวนไม่น้อยกว่าค่าที่กำหนด ด้วยเทคนิคนี้สามารถใช้ในการกรองข้อมูลรบกวนซึ่งเป็นข้อมูลที่มีความหนาแน่นเบาบางได้ และยังสามารถค้นหาคลัสเตอร์ที่มีรูปทรงที่ซับซ้อนได้อีกด้วย เทคนิคการจัดกลุ่มข้อมูลโดยพิจารณาความหนาแน่นของข้อมูลที่เป็นที่รู้จักได้แก่

DBSCAN พัฒนาขึ้นโดย Ester, Kriegel, Sander and Xu (1996) อัลกอริธึมจะทำการสร้างและขยายขอบเขตของคลัสเตอร์ออกไปเรื่อยๆ โดยพิจารณาความหนาแน่นที่กำหนด พารามิเตอร์ที่สำคัญได้แก่ *Eps* (ε-neighborhood) และ *MinPts* (min point) โดยที่ *Eps* คือรัศมีหรือระยะห่างที่ใช้อ้างอิงการเชื่อมถึงกันด้วยความหนาแน่น (density-connected) ของทุกอ็อบเจกต์ภายในคลัสเตอร์ และ *MinPts* คือจำนวนอ็อบเจกต์หรือ data point น้อยสุดภายในรัศมี *Eps* จาก อ็อบเจกต์ใดๆ อ็อบเจกต์ที่สามารถเชื่อมถึงกันได้ด้วยความหนาแน่นจะถูกจัดให้อยู่ในคลัสเตอร์เดียวกัน ความหนาแน่นของข้อมูลสามารถพิจารณาจากจำนวนอ็อบเจกต์ใกล้เคียง (N_{Eps}) ซึ่งก็คือจำนวนอ็อบเจกต์ทั้งหมดภายในรัศมี *Eps* จากอ็อบเจกต์หนึ่งๆ ให้ o, p และ q แทนอ็อบเจกต์ใดๆ p สามารถเชื่อมถึง q ได้ด้วยความหนาแน่นก็ต่อเมื่อมีอ็อบเจกต์ o ซึ่ง $\left|N_{Eps}(o)\right| \geq MinPts$ และ o สามารถแพร่ไปถึง (density-reachable) ทั้ง p และ q ส่วนอ็อบเจกต์ที่มีความหนาแน่นไม่เพียงพอและการเชื่อมถึงกันได้ด้วยความหนาแน่นจากบริเวณที่มีข้อมูลหนาแน่นไปไม่ถึงจะถือว่าอ็อบเจกต์นั้นเป็นข้อมูลรบกวนซึ่งมีลักษณะต่างจากข้อมูลทั่วไป (พิจารณาจากรูปที่ 2.9)

อัลกอริธึม DBSCAN เริ่มต้นจากการอ่านทุกอ็อบเจกต์มาเก็บไว้ในโครงสร้างแบบ R*-tree เพื่อเพิ่มประสิทธิภาพในการค้นหาข้อมูล จากนั้นจึงอ่านข้อมูลทีละอ็อบเจกต์ อ็อบเจกต์ใดที่มีความหนาแน่นเบาบางจะถูกจัดให้เป็นข้อมูลรบกวน ส่วนอ็อบเจกต์ใดที่มีความหนาแน่นถึงเกณฑ์ก็จะทำการค้นหาและจัดให้ทุกๆ อ็อบเจกต์ที่อ็อบเจกต์นั้นสามารถเชื่อมถึงได้ด้วยความหนาแน่นอยู่ในคลัสเตอร์เดียวกัน ด้วยเทคนิคดังกล่าวทำให้สามารถค้นหาคลัสเตอร์ที่มีรูปร่างที่ซับซ้อนได้ โดยจะเห็นได้ว่าคลัสเตอร์ที่ได้จะอยู่ในบริเวณที่มีข้อมูลปรากฏอยู่อย่างหนาแน่นซึ่งถูกแบ่งออกจากกันด้วยบริเวณที่มีความหนาแน่นของข้อมูลน้อยกว่า (Kollios, Gunopulos, Koudas and

Berchtold, 2003) ความซับซ้อนของอัลกอริธึม DBSCAN มีค่าเท่ากับ $O(n \cdot \log n)$ เมื่อ n คือจำนวนอ็อบเจกต์ทั้งหมด



รูปที่ 2.9 Density-reachability and Density-connectivity (Ester et al., 1996)

แม้ว่าอัลกอริธึม DBSCAN จะสามารถค้นหาคลัสเตอร์ที่มีรูปร่างซับซ้อนได้ แต่การหาค่าพารามิเตอร์ *Eps* และ *MinPts* ที่เหมาะสมยังเป็นเรื่องที่ยาก และ DBSCAN ยังไม่สามารถแยกคลัสเตอร์สองคลัสเตอร์ที่มีการเชื่อมกันด้วยเส้นของข้อมูลที่มีความหนาแน่นได้ (สองคลัสเตอร์ด้านบนดังแสดงไว้ในรูปที่ 2.7) อีกทั้งอัลกอริธึมยังใช้เวลามากสำหรับการค้นหาคลัสเตอร์บนฐานข้อมูลที่มีขนาดใหญ่ เนื่องจากจะต้องพิจารณาทุกอ็อบเจกต์บนฐานข้อมูล (Guha et al., 1998)

จากอัลกอริธึมจัดกลุ่มข้อมูลดังที่ได้ยกตัวอย่างมาแล้วข้างต้น ผู้พัฒนาต่างพยายามศึกษาและพัฒนาเทคนิคเพื่อให้อัลกอริธึมสามารถค้นหาคลัสเตอร์ที่มีรูปร่างต่างๆ ได้อย่างแม่นยำ โดยคำนึงถึงผลกระทบของข้อมูลรบกวนและเอ้าท์ไลเออร์ด้วย ดังได้กล่าวไว้แล้วในรายละเอียดของหลายๆ อัลกอริธึม อีกทั้งอัลกอริธึมที่ดีควรจะสามารถรองรับข้อมูลที่มีขนาดใหญ่ได้ ซึ่งหลายๆ อัลกอริธึมได้ผสมผสานเทคนิคการลดขนาดข้อมูลเข้าไปในขั้นตอนต่างๆ ของการจัดกลุ่มข้อมูล เช่น BIRCH ใช้ CF-tree ซึ่งเป็นการลดขนาดข้อมูลซึ่งใช้เทคนิค data summarization หรือการสรุปข้อมูล ส่วน CURE ใช้เทคนิคการสุ่มข้อมูล ทั้งนี้เพื่อเป็นการลดเวลาและหน่วยความจำที่ต้องสิ้นเปลืองไประหว่างกระบวนการจัดกลุ่มข้อมูลโดยการเลือกพิจารณาเฉพาะข้อมูลที่สำคัญเท่านั้น ซึ่งการเลือกใช้เทคนิคการลดขนาดข้อมูลที่เหมาะสมจะสามารถเลือกข้อมูลที่สามารถเป็นตัวแทนที่ดีของชุดข้อมูลทั้งหมดได้

2.2 การลดขนาดข้อมูล (Data Reduction)

ในการทำเหมืองข้อมูลนั้น เนื่องจากข้อมูลที่ใช้ประกอบด้วยข้อมูลหลากหลายรูปแบบ ทั้งข้อมูลเชิงตัวเลข (numerical value) และข้อมูลเชิงอักขระ (categorical value) อีกทั้งยังอาจปะปนไปด้วยข้อมูลที่ไม่สมบูรณ์ เช่นการขาดหายไปของข้อมูลบางส่วน ตลอดจนข้อมูลรบกวน อีกทั้งข้อ

มูลที่ใช้ยังต้องสอดคล้องกับข้อจำกัดของอัลกอริธึมจัดกลุ่มข้อมูลด้วย ซึ่งอัลกอริธึมจัดกลุ่มข้อมูลแต่ละอัลกอริธึมมีวัตถุประสงค์และข้อกำหนดสำหรับข้อมูลที่ใช้แตกต่างกัน อัลกอริธึมจัดกลุ่มข้อมูลบางอัลกอริธึมอาจมีข้อจำกัดเรื่องความสามารถในการรองรับข้อมูลที่มีขนาดใหญ่ การใช้ข้อมูลที่มีขนาดเหมาะสมจึงเป็นสิ่งที่ต้องคำนึงถึงด้วยเช่นกัน โดยที่ผู้ใช้จำเป็นจะต้องเลือกอัลกอริธึมที่เหมาะสมกับชุดข้อมูลนั้นๆ จึงจะทำให้สามารถบรรลุวัตถุประสงค์ตามต้องการได้ การทำเหมืองข้อมูลจากข้อมูลที่ไม่ได้ผ่านการเตรียมข้อมูลอาจเกิดปัญหาขึ้นในระหว่างกระบวนการได้ เช่น หน่วยความจำไม่เพียงพอ หรือใช้เวลานานจนไม่สามารถนำผลการวิเคราะห์มาใช้ประโยชน์ได้ทันเวลา และอาจทำให้ความรู้ที่ได้จากข้อมูลเหล่านั้นผิดเพี้ยนไปได้ ดังนั้นการเตรียมข้อมูลก่อนการทำเหมืองข้อมูลจึงเป็นขั้นตอนหนึ่งที่สำคัญและมิอาจถูกละเลยได้

การลดขนาดข้อมูลเป็นกระบวนการหนึ่งในขั้นตอนการเตรียมข้อมูล นั่นคือการทำให้ข้อมูลตั้งต้นมีขนาดลดลงโดยสูญเสียลักษณะสำคัญของข้อมูลน้อยที่สุดและสูญเสียความถูกต้องของผลลัพธ์น้อยที่สุด เนื่องจากข้อมูลแต่ละตัวจะมีความสำคัญต่อการจัดกลุ่มข้อมูลไม่เท่ากัน ด้วยเทคนิคการเลือกข้อมูลที่ดีจะทำให้สามารถเลือกข้อมูลที่มีความสำคัญและสามารถใช้เป็นตัวแทนของข้อมูลส่วนใหญ่ได้ การลดขนาดข้อมูลสามารถทำได้ดังวิธีการต่อไปนี้

2.2.1 Data Summarization

Data summarization คือการลดขนาดข้อมูลด้วยการสรุปข้อมูล ยกตัวอย่างเช่น ข้อมูลการขายของห้างร้านแห่งหนึ่งซึ่งได้จากข้อมูลการขายสินค้า ณ จุดขาย และผู้บริหารต้องการทราบรูปแบบของการขายโดยสรุปในแต่ละปี ดังนั้นข้อมูลที่ใช้จึงควรเป็นข้อมูลสรุปของแต่ละปีแทนการใช้ข้อมูลทุกเรคคอร์ด ณ จุดขาย

2.2.2 Dimensionality Reduction

Dimensionality reduction คือการลดขนาดข้อมูลโดยการตัดแอททริบิวต์หรือมิติของข้อมูลที่ไม่มีความเกี่ยวข้อง หรือมีความเกี่ยวข้องน้อย หรือไม่มีความสำคัญต่อการค้นหารูปแบบที่สนใจ เช่น การค้นหารูปแบบที่มีผลต่อการสำเร็จการศึกษาตามกำหนดภายในระยะเวลา 4 ปีการศึกษาของนักศึกษาระดับปริญญาตรีของมหาวิทยาลัยเทคโนโลยีสุรนารี ชื่อและรหัสของนักศึกษาอาจเป็นแอททริบิวต์ที่ไม่มีความเกี่ยวข้องสำหรับการค้นหารูปแบบที่สนใจ เพราะชื่อและรหัสนักศึกษาไม่มีผลต่อการสำเร็จการศึกษา แต่ในทางกลับกัน สาขาวิชา ระดับคะแนนของแต่ละรายวิชา เกรดเฉลี่ยในแต่ละเทอม และเกรดเฉลี่ยสะสมของนักศึกษา จะเป็นสิ่งที่สามารถเชื่อมโยงไปสู่สิ่งที่ต้องการค้นหาได้ ซึ่งจุดประสงค์หลักของการลดขนาดข้อมูลด้วยเทคนิคนี้คือการเลือกเซ็ตของแอททริบิวต์ที่เล็กที่สุดที่มีความน่าจะเป็นที่จะสามารถให้ผลลัพธ์ของรูปแบบที่น่าสนใจได้

เทียบเท่ากับการใช้ทุกแอททริบิวต์ การทำเหมืองข้อมูลกับเซ็ตของแอททริบิวต์ที่ถูกเลือกไว้แล้วนั้น ยังมีส่วนช่วยลดจำนวนแอททริบิวต์ที่ปรากฏในเงื่อนไขของรูปแบบที่ค้นพบ ซึ่งทำให้รูปแบบที่ได้กระชับและสามารถทำความเข้าใจได้ง่ายขึ้นอีกด้วย

2.2.3 Data Compression

Data compression คือการแปลงข้อมูลหรือการบีบอัดข้อมูลให้มีขนาดเล็กลงแต่ยังคงสามารถแสดงถึงข้อมูลทั้งหมดได้ ถ้าข้อมูลต้นฉบับสามารถถูกสร้างกลับได้จากข้อมูลบีบอัดโดยไม่มีการสูญเสียรายละเอียดของข้อมูลไปเทคนิคที่ใช้ดังกล่าวจะถูกเรียกว่า lossless เช่นการบีบอัดเอกสารซึ่งจะต้องสามารถรักษารายละเอียดของคำทุกคำไว้ได้อย่างครบถ้วนและสมบูรณ์ แต่ในทางตรงกันข้ามหากข้อมูลสามารถสร้างกลับได้เพียงบางส่วนของข้อมูลต้นฉบับโดยสูญเสียรายละเอียดของข้อมูลบางส่วนไปเทคนิคที่ใช้ดังกล่าวจะถูกเรียกว่า lossy เช่นการบีบอัดรูปภาพเป็นแบบ JPEG ซึ่งถาพที่ถูกบีบอัดด้วยเทคนิคนี้จะมีคุณภาพลดลงตามอัตราการบีบอัดที่ใช้ ส่วนเทคนิคการบีบอัดแบบ lossy แบบอื่นๆ ได้แก่ Wavelet Transform(WT) และ Principal Component Analysis(PCA) เป็นต้น

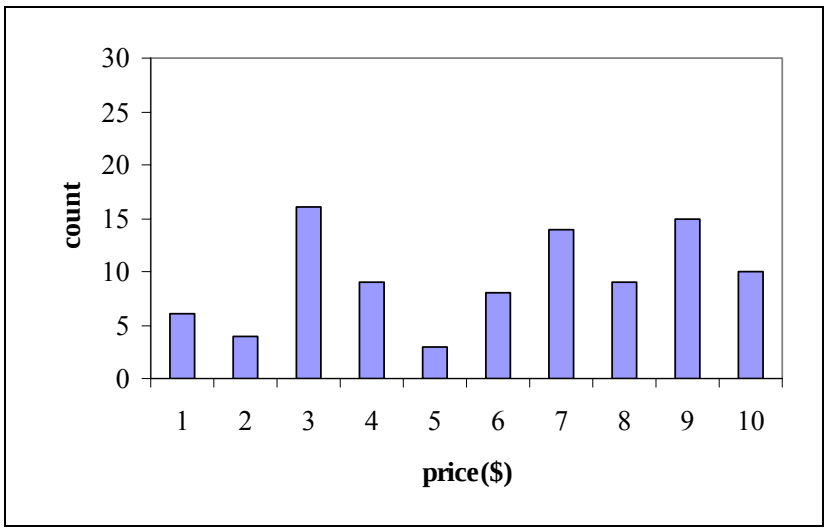
2.2.4 Numerosity Reduction

Numerosity reduction คือการลดขนาดข้อมูลโดยการเลือกใช้รูปแบบที่ง่ายขึ้นในการแสดงถึงข้อมูลแทนการใช้ข้อมูลจริงทั้งหมด ซึ่งเทคนิคนี้อาจเป็นได้ทั้งแบบ parametric และ nonparametric สำหรับวิธีแบบ parametric จะใช้โมเดลในการอธิบายข้อมูลส่วนใหญ่โดยจะเก็บเฉพาะพารามิเตอร์ของโมเดลที่สร้างขึ้นแทนข้อมูลทั้งหมด เช่น เทคนิค regression และ log-linear models เป็นต้น ส่วนอีกวิธีคือ nonparametric จะใช้วิธีการลดขนาดหรือจำนวนข้อมูลที่ใช้แสดงถึงข้อมูลเดิม โดยอาจเลือกใช้วิธีการกระจายข้อมูลแบบฮิสโทแกรม (histogram method) การเลือกข้อมูลด้วยเทคนิคคลัสเตอร์ (clustering method) หรือการเลือกข้อมูลด้วยเทคนิคการสุ่มข้อมูล (sampling)

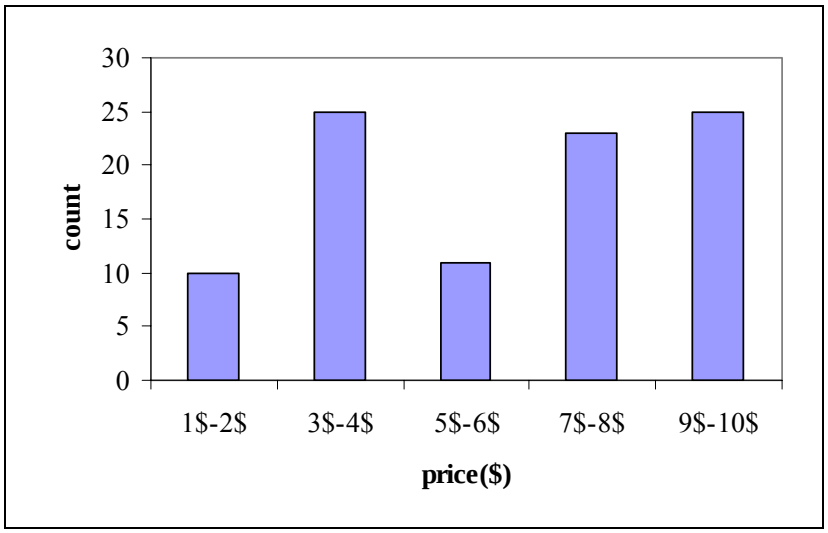
Regression and Log-Linear Models เทคนิคนี้จะสร้างและใช้โมเดลที่สร้างขึ้นในการประมาณข้อมูล โดยในเทคนิค linear regression ข้อมูลจะถูกนำมาสร้างเป็นโมเดลหรือสมการเชิงเส้นตรง เช่น การใช้สมการเชิงเส้น $Y = \alpha+\beta X$ แทนข้อมูลทั้งหมด โดยจะเก็บเฉพาะพารามิเตอร์ของสมการไว้เท่านั้น

Histograms เป็นเทคนิคการลดขนาดข้อมูลที่นิยมใช้อีกเทคนิคหนึ่ง โดยการใช้ช่วงของข้อมูลและความถี่ของข้อมูลที่อยู่ในช่วงนั้นแทนข้อมูลเดิม สำหรับการสร้างฮิสโทแกรมบนแอททริบิวต์ใดๆ แอททริบิวต์นั้นจะถูกแบ่งออกเป็นช่วงของข้อมูลย่อยๆที่ไม่มีการซ้อนทับกัน (buckets) จากนั้นจะทำการหาความถี่ของข้อมูลซึ่งมีค่าอยู่ในแต่ละช่วงข้อมูลนั้น โดยถ้าช่วงข้อมูล

ใดแสดงค่าของข้อมูลเดี่ยวๆ จะเรียกช่วงข้อมูลนั้นว่า singleton buckets (ดังรูปที่ 2.10) สำหรับฮิสโทแกรมที่มีความกว้างของช่วงข้อมูลเท่ากันจะเรียกว่า equiwidth (ดังรูปที่ 2.11) ส่วนฮิสโทแกรมอีกชนิดหนึ่งคือ equidepth จะกำหนดให้ข้อมูลในแต่ละช่วงมีความถี่เท่ากัน โดยในแต่ละช่วงจะประกอบด้วยข้อมูลที่มีค่าใกล้เคียงกันซึ่งได้จากการเรียงลำดับตามค่าของแอททริบิวต์นั้นๆ

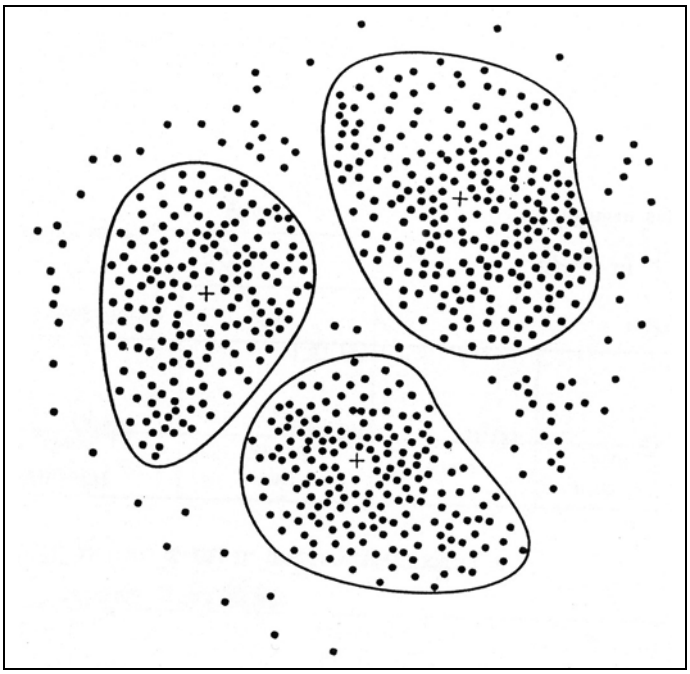


รูปที่ 2.10 ฮิสโทแกรมของราคาสินค้าที่เป็น singleton buckets



รูปที่ 2.11 ฮิสโทแกรมที่มีความกว้างของช่วงข้อมูลเท่ากันของราคาสินค้า

Clustering เป็นเทคนิคที่พิจารณาข้อมูลแต่ละเรคคอร์ดเสมือนวัตถุ (object) ซึ่งมีหลักการเหมือนกับการจัดกลุ่มข้อมูล คือจะทำการแบ่งข้อมูลออกเป็นแต่ละคลัสเตอร์ โดยจะจัดให้วัตถุที่มีความคล้ายคลึงกันอยู่ในคลัสเตอร์เดียวกัน และวัตถุที่อยู่ต่างคลัสเตอร์กันจะมีความคล้ายคลึงกันน้อยที่สุด ซึ่งความเหมือนหรือต่างกันสามารถเปรียบเทียบได้กับความใกล้ชิดกันของวัตถุใดๆ โดยใช้ระยะทางเป็นตัวชี้วัด คุณภาพของแต่ละคลัสเตอร์สามารถอธิบายได้จากเส้นผ่านศูนย์กลางของคลัสเตอร์ (diameter) ซึ่งแสดงระยะห่างมากสุดของวัตถุสองชิ้นที่อยู่ในคลัสเตอร์เดียวกัน ในขณะเดียวกันระยะห่างระหว่างจุดศูนย์กลางของคลัสเตอร์ (centroid) กับวัตถุภายใน คลัสเตอร์เดียวกันยังสามารถใช้อธิบายถึงคุณภาพของคลัสเตอร์ได้อีกทางหนึ่ง โดยจะแสดงถึงระยะ ห่างเฉลี่ยของวัตถุทุกชิ้นกับจุดศูนย์กลางของคลัสเตอร์นั้น โดยที่แต่ละคลัสเตอร์จะมีตัวแทนที่สามารถแทนวัตถุทุกชิ้นของคลัสเตอร์นั้นได้ เช่น การใช้จุดศูนย์กลางคลัสเตอร์แทนคลัสเตอร์นั้น สำหรับบางเทคนิคตัวแทนของคลัสเตอร์อาจมีได้หลายตัวแทน ทั้งนี้ขึ้นอยู่กับความเหมาะสมของแต่ละเทคนิคที่เลือกใช้ สำหรับการลดขนาดข้อมูลด้วยเทคนิคนี้จะใช้ตัวแทนของแต่ละคลัสเตอร์แทนข้อมูลทั้งหมดที่อยู่ในคลัสเตอร์นั้น ดังรูปที่ 2.12 ซึ่งแสดงข้อมูลตัวอย่างที่ประกอบด้วย 3 คลัสเตอร์ โดยจุดศูนย์กลางของคลัสเตอร์แสดงด้วยเครื่องหมาย “+” ซึ่งแทนตัวแทนของข้อมูลภายในคลัสเตอร์

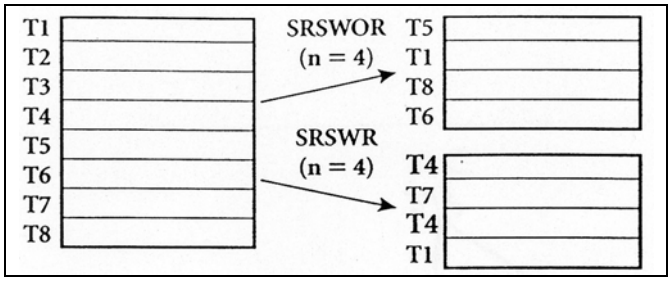


รูปที่ 2.12 แสดงข้อมูลตัวอย่างที่ประกอบด้วย 3 คลัสเตอร์ (Han and Kamber, 2001, p.128)

Sampling เป็นเทคนิคการลดขนาดข้อมูลด้วยวิธีการสุ่มข้อมูล โดยการใช้ชุดข้อมูลที่ได้จากการสุ่มแทนชุดข้อมูลตั้งต้นที่มีขนาดใหญ่มาก ให้ D แทนชุดข้อมูลขนาดใหญ่ที่ประกอบด้วยข้อมูล N เรคคอร์ด ตัวอย่างเทคนิคการสุ่มข้อมูลในแบบต่างๆ บน D แสดงไว้ดังต่อไปนี้

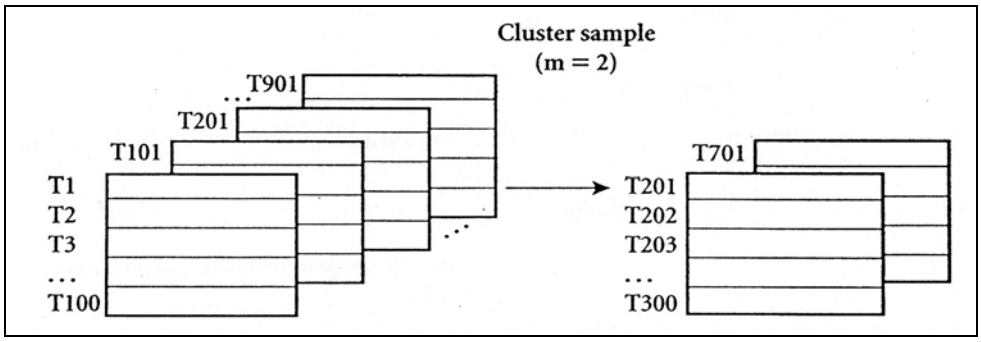
1) การสุ่มอย่างง่ายแบบไม่ใส่กลับ (Simple Random Sampling without Replacement: SRSWOR) เป็นการหยิบข้อมูลจำนวน n ของ N เรคคอร์ดจาก D โดยที่ความน่าจะเป็นที่เรคคอร์ดใดๆ จะถูกหยิบมีค่าเท่ากับ $1/N$ เท่ากันหมดทุกเรคคอร์ด นั่นคือทุกเรคคอร์ดมีโอกาสที่จะถูกเลือกเท่าๆ กัน (ดูรูปที่ 2.13)

2) การสุ่มอย่างง่ายแบบใส่กลับ (Simple Random Sampling with Replacement: SRSWR) เทคนิคนี้คล้ายกับ SRSWOR ต่างกันแต่เพียงทุกครั้งที่หยิบเรคคอร์ดใดๆ จาก D จะทำการบันทึกเรคคอร์ดนั้นไว้ จากนั้นจึงคืนเรคคอร์ดนั้นเข้าใน D เหมือนเดิม ซึ่งหมายความว่าเมื่อมีการหยิบครั้งต่อไป เรคคอร์ดที่เคยถูกหยิบขึ้นมาแล้วนั้นยังมีโอกาสที่จะถูกหยิบขึ้นมาซ้ำได้อีกนั่นเอง (ดูรูปที่ 2.13)



รูปที่ 2.13 การสุ่มข้อมูลแบบ SRSWOR และ SRSWR (Han and Kamber, 2001, p.131)

3) การสุ่มแบบคลัสเตอร์ (cluster sampling) เป็นการสุ่มอีกรูปแบบหนึ่งซึ่งอาศัยคุณสมบัติของคลัสเตอร์ร่วมกับการสุ่มอย่างง่าย ทำได้โดยการจัดกลุ่มหรือแบ่งกลุ่มข้อมูลใน D ออกเป็น M กลุ่มย่อยหรือที่เรียกว่าคลัสเตอร์ที่ไม่มีการซ้อนทับกัน จากนั้นจึงทำการสุ่มเลือกคลัส-เตอร์ m คลัสเตอร์ โดยที่ $m<M$ ตัวอย่างเช่น การสุ่มข้อมูลที่เก็บไว้บนฐานข้อมูล ซึ่งข้อมูลแต่ละเรคคอร์ดจะถูกอ่านขึ้นมาทีละเพจในแต่ละครั้ง โดยที่แต่ละเพจสามารถเทียบได้กับคลัสเตอร์ เมื่ออ่านข้อมูลมาในแต่ละเพจจะทำการสุ่มว่าจะเลือกข้อมูลเพจนั้นหรือไม่ แล้วจึงอ่านข้อมูลใน เพจต่อไป ทำเช่นนี้จนกระทั่งได้ชุดข้อมูลสุ่มครบถ้วนตามต้องการ (ดูรูปที่ 2.14)



รูปที่ 2.14 ตัวอย่างการสุ่มข้อมูลแบบคลัสเตอร์ (Han and Kamber, 2001, p.131)

4) การสุ่มโดยใช้สัดส่วนคงเดิม (stratified sampling) เป็นการสุ่มที่มีลักษณะคล้ายกับการสุ่มแบบคลัสเตอร์ แต่ต่างกันที่วิธีการสุ่มซึ่งจะสุ่มข้อมูลที่อยู่ในคลัสเตอร์หรือกลุ่มนั้นๆ แทนการสุ่มเอาข้อมูลทั้งกลุ่มออกมา ทำได้โดยการแบ่งข้อมูลบน D ออกเป็นกลุ่มย่อยที่ไม่มีการซ้อนทับกันซึ่งเรียกว่า stratum การสุ่มข้อมูลจาก D ทำได้โดยการสุ่มข้อมูลอย่างง่ายในแต่ละ stratum ด้วยการสุ่มดังกล่าวจะสามารถทำให้มั่นใจได้ว่าจะได้ตัวแทนของข้อมูลทุกกลุ่มแม้ว่ากลุ่มข้อมูลนั้นจะมีขนาดเล็กกว่าก็ตาม ยกตัวอย่างเช่น การสุ่มโดยใช้สัดส่วนคงเดิมบนแฟ้มข้อมูลลูกค้า ซึ่งแต่ละ stratum ถูกสร้างขึ้นสำหรับทุกช่วงอายุของลูกค้า เช่น 1-20, 20-40, 40-60, 60-80 และ 81 ปีขึ้นไป ซึ่งบางช่วงของอายุอาจมีจำนวนลูกค้าน้อยมาก (81 ปีขึ้นไป) ด้วยการสุ่มวิธีนี้จะสามารถทำให้มั่นใจได้ว่าชุดข้อมูลสุ่มที่ได้จะปรากฏตัวแทนของข้อมูลทุกกลุ่มแม้ว่ากลุ่มข้อมูลนั้นจะมีขนาดเล็กก็ตาม เนื่องจากเทคนิคดังกล่าวเป็นเทคนิคที่มีความสำคัญต่องานวิจัยนี้และเพื่อให้เกิดความชัดเจนยิ่งขึ้น จะแสดงตัวอย่างการสุ่มเป็นขั้นตอนดังรูปที่ 2.15

สำหรับการลดขนาดข้อมูล เทคนิคการสุ่มเป็นเทคนิคหนึ่งที่ง่ายและสามารถใช้เพื่อหาคำตอบโดยประมาณของข้อสอบถามแบบสรุปได้เป็นอย่างดี และขนาดของข้อมูลสุ่มยังสามารถแสดงถึงค่าความเชื่อมั่นซึ่งแสดงถึงความถูกต้องของผลลัพธ์ที่ต้องการได้อีกด้วย โดยเมื่อต้องการความถูกต้องที่สูงขึ้นก็สามารถเพิ่มขนาดของข้อมูลสุ่มได้ตามต้องการ ซึ่งการสุ่มด้วยเทคนิคที่มีประสิทธิภาพจะสามารถสร้างชุดข้อมูลสุ่มที่มีขนาดเล็กได้โดยที่ยังคงความถูกต้องได้ใกล้เคียงกับข้อมูลเดิมทั้งหมด

การสุ่มข้อมูลโดยใช้สัดส่วนคงเดิมบนแฟ้มข้อมูลลูกค้า จำนวน 13 คน จากข้อมูลของลูกค้าทั้งหมด 27 คน ซึ่งมีอายุดังต่อไปนี้

13,15,16,16,19,20,20,21,22,22,25,25,25,25,30,33,33,35,35,35,35,36,40,45,46,52,70.

โดยแบ่งกลุ่มตามช่วงอายุดังนี้ (a) Age<22 และ (b) Age >= 22

1. แบ่งข้อมูลออกเป็นกลุ่ม a และ b

   (a) Age<22: (8 เรคคอร์ด)

   13, 15, 16, 16, 19, 20, 20, 21

   (b) Age >= 22: (19 เรคคอร์ด)

   22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70

2. หาจำนวนของข้อมูลที่จะต้องทำการสุ่มออกมาจากข้อมูลของแต่ละกลุ่ม (a และ b) เมื่อต้องการสุ่มข้อมูลจำนวนเท่ากับ 13

   จำนวนที่จะต้องสุ่มจากแต่ละกลุ่ม หาได้จากวิธีการเทียบบัญญัติไตรยางค์ จะได้ว่า

   จำนวนที่จะต้องสุ่มจากกลุ่ม a เท่ากับ (13 x 8) / 27 = 3.9 = 4 เรคคอร์ด

   จำนวนที่จะต้องสุ่มจากกลุ่ม b เท่ากับ (13 x 19) / 27 = 9.1 = 9 เรคคอร์ด

3. ทำการสุ่มตามจำนวนที่ได้คำนวณไว้แล้วข้างต้น จะได้

   สุ่มจาก (a) 13, 16, 19, 20

   สุ่มจาก (b) 22, 25, 25, 33, 35, 35, 36, 45, 52

ซึ่งจะได้ผลลัพธ์คือ 13, 16, 19, 20, 22, 25, 25, 33, 35, 35, 36, 45, 52

รูปที่ 2.15 ตัวอย่างการสุ่มข้อมูลแบบสัดส่วนคงเดิม

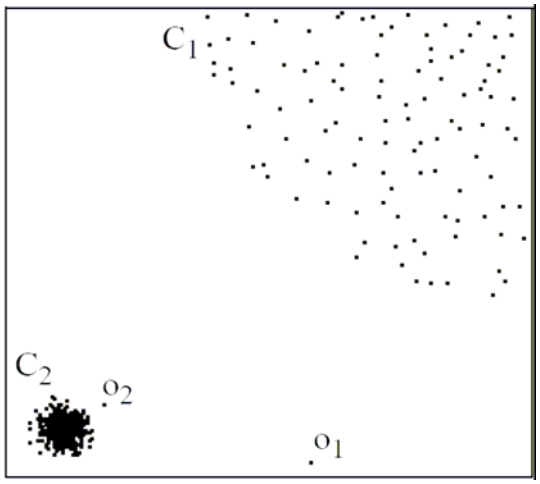
2.3 ข้อมูลรบกวนและเอ้าท์ไลเออร์ (Noise and Outlier)

การทำเหมืองข้อมูลเป็นการค้นหาองค์ความรู้ที่อยู่ภายในข้อมูล ดังนั้นขนาดของข้อมูลที่มากพอจึงจะทำให้ความรู้ที่ได้มีความน่าเชื่อถือและใกล้เคียงกับความเป็นจริงมากที่สุด ซึ่งข้อมูลที่ใช้ในงานการทำเหมืองข้อมูลนั้นอาจประกอบด้วยข้อมูลจากหลายแหล่ง ด้วยวิธีการเก็บรวบรวมข้อมูลที่แตกต่างกันออกไป ซึ่งข้อมูลที่ได้อาจมาจากการบันทึกโดยตรงจากผู้ใช้งานระบบ ข้อมูลบางส่วนอาจเป็นข้อมูลที่นำเข้ามาจากระบบอื่นโดยที่อาจมีรูปแบบของข้อมูลที่แตกต่างกัน ซึ่งเมื่อนำข้อมูลเหล่านั้นมารวมกันก็จำเป็นจะต้องทำให้ข้อมูลทุกส่วนมีรูปแบบที่เหมาะสมต่อไป แต่อย่างไรก็ตามในข้อมูลจำนวนมากอาจมีข้อมูลบางส่วนที่มีค่าหรือรูปแบบที่แตกต่างไปจากข้อมูลทั่วไป เรียก

ข้อมูลเหล่านั้นว่าเอ้าท์ไลเออร์ (outlier)

เอ้าท์ไลเออร์อาจเกิดขึ้นได้ในหลายลักษณะ อาทิ การป้อนข้อมูลที่ผิดพลาดของผู้ใช้งานระบบโดยตรง ตัวอย่างเช่น ข้อมูลอายุของพนักงานในบริษัทที่แสดงเป็น -999 ซึ่งสามารถเกิดขึ้นได้จากการที่โปรแกรมตั้งค่าเริ่มต้นของข้อมูลดังกล่าวไว้ด้วยค่านั้นและปราศจากการแก้ไขให้ถูกต้องโดยพนักงาน ในขณะที่เอ้าท์ไลเออร์บางส่วนอาจเป็นข้อมูลที่ถูกต้องแต่มีรูปแบบหรือค่าของข้อมูลต่างไปจากข้อมูลส่วนใหญ่ ตัวอย่างเช่น เงินเดือนของผู้บริหารของบริษัทแห่งหนึ่ง ซึ่งเป็นธรรมดาที่จะต้องมีค่าของเงินเดือนที่สูงกว่าเงินเดือนของพนักงานทั่วไปในบริษัท

ในงานการจัดกลุ่มข้อมูล เนื่องจากวัตถุประสงค์หลักของอัลกอริธึมจัดกลุ่มข้อมูลคือการค้นหาคลัสเตอร์ที่เป็นไปได้ และเพื่อให้สามารถระบุคลัสเตอร์ของข้อมูลได้อย่างชัดเจน ข้อมูลที่มีรูปแบบหรือมีค่าต่างไปจากข้อมูลส่วนใหญ่หรือเอ้าท์ไลเออร์นั้นจะถูกจัดเป็นข้อมูลรบกวน(noise) ข้อมูลรบกวนจะทำให้กระบวนการค้นหาคลัสเตอร์ทำได้ลำบากขึ้น อีกทั้งยังอาจส่งผลให้คลัสเตอร์ที่ได้มีลักษณะที่ผิดเพี้ยนไปได้ ในอัลกอริธึมจัดกลุ่มข้อมูลบางอัลกอริธึมจะเพิ่มคุณสมบัติในการกรองข้อมูลรบกวนออกไป และทำการค้นหาคลัสเตอร์โดยจะไม่สนใจข้อมูลรบกวนเหล่านั้น ด้วยวิธีการดังกล่าวทำให้อัลกอริธึมยังคงสามารถค้นหาคลัสเตอร์ได้ถูกต้องแม้ข้อมูลที่ใช้ปะปนไปด้วยข้อมูลรบกวนก็ตาม ซึ่งเรียกคุณสมบัติดังกล่าวว่าคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึม (noise-tolerance)



รูปที่ 2.16 ตัวอย่างชุดข้อมูลขนาดสองมิติ

การตรวจสอบว่าข้อมูลใดเป็นเอ้าท์ไลเออร์สามารถทำได้หลายแนวทาง เช่น การค้นหาเอ้าท์ไลเออร์ด้วยวิธีการทางสถิติ (statistical-based) โดยการทดสอบสมมุติฐานว่าข้อมูลนั้นมีความน่าจะเป็นที่จะเป็นเอ้าท์ไลเออร์หรือไม่ แต่มีข้อจำกัดที่ผู้ใช้จำเป็นจะต้องทราบลักษณะของข้อมูลที่ใช้ด้วย (ลักษณะการกระจายของข้อมูลและความแปรปรวนของข้อมูล) การค้นหาเอ้าท์ไลเออร์ด้วยระยะทาง (distance-based) เป็นการการค้นหาเอ้าท์ไลเออร์จากข้อมูลข้างเคียงซึ่งอยู่ภายในรัศมีที่กำหนด โดยที่อ็อบเจกต์ที่เป็นเอ้าท์ไลเออร์จะเป็นอ็อบเจกต์ที่มีจำนวนอ็อบเจกต์ข้างเคียงต่ำกว่าจำนวนที่กำหนด ซึ่งหมายความว่าอ็อบเจกต์นั้นมีความแตกต่างจากข้อมูลทั่วไป (พิจารณาจากระยะทาง) การค้นหาเอ้าท์ไลเออร์ด้วยระยะทางใช้ได้ดีกับข้อมูลที่แต่ละคลัสเตอร์มีความหนาแน่นของข้อมูลใกล้เคียงกัน แต่ในบางกรณีที่แต่ละคลัสเตอร์มีขนาดและความหนาแน่นแตกต่างกันมากการค้นหาเอ้าท์ไลเออร์ด้วยระยะทางจะไม่สามารถค้นหาเอ้าท์ไลเออร์บางส่วนได้ พิจารณารูปที่ 2.16 เป็นตัวอย่างข้อมูลสองมิติจำนวน 502 อ็อบเจกต์ โดย 400 อ็อบเจกต์เป็นของคลัสเตอร์ C_1 100 อ็อบเจกต์เป็นของคลัสเตอร์ C_2 และอีก 2 อ็อบเจกต์คืออ็อบเจกต์ o_1 และ o_2 จากตัวอย่างนี้ C_2 เป็นคลัสเตอร์ที่มีความหนาแน่นมากกว่า C_1 โดยที่ o_1 และ o_2 เป็นอ็อบเจกต์ที่ไม่เป็นของคลัสเตอร์ใดซึ่งจะต้องจัดให้อ็อบเจกต์ทั้งสองเป็นเอ้าท์ไลเออร์ การค้นหาเอ้าท์ไลเออร์ด้วยระยะทางจะไม่สามารถระบุว่า o_2 เป็นเอ้าท์ไลเออร์ในขณะที่อ็อบเจกต์ภายในคลัสเตอร์ C_1 ไม่เป็น เนื่องจากระยะทางจาก o_2 ไปยังคลัสเตอร์ C_2 มีค่าน้อยกว่าระยะทางระหว่างบางอ็อบเจกต์ใน C_1 การกำหนดค่ารัศมีที่มีค่าน้อยกว่าระยะทางจาก o_2 ไปยังคลัสเตอร์ C_2 จะทำให้ o_2 และทุกอ็อบเจกต์ใน C_1 เป็นเอ้าท์ไลเออร์ทั้งหมด แต่การกำหนดค่ารัศมีให้มีค่ามากขึ้นก็กลับทำให้อ็อบเจกต์ของ C_1 บางส่วนถูกจัดเป็นเอ้าท์ไลเออร์อีก ดังนั้น Breunig, Kriegel, Ng and Sander (2000) จึงเสนอเทคนิคการค้นหาเอ้าท์ไลเออร์ด้วยความหนาแน่น (density-based) โดยการพิจารณาค่า local outlier factor (*LOF*) ซึ่งจะพิจารณาความหนาแน่นของอ็อบเจกต์ควบคู่กับความหนาแน่นของอ็อบเจกต์ข้างเคียง ซึ่งข้อมูลทั่วไปจะมีค่า *LOF* ใกล้เคียง 1 (นั่นคืออ็อบเจกต์นั้นมีความหนาแน่นใกล้เคียงกับความหนาแน่นของอ็อบเจกต์ในบริเวณใกล้เคียงกัน) ซึ่งอ็อบเจกต์ที่เป็นเอ้าท์ไลเออร์จะมีค่า *LOF* สูงหรือต่ำกว่า 1 ในปริมาณที่แตกต่างกัน โดยอ็อบเจกต์ที่มีค่า $|LOF\text{-}1|$ มากแสดงถึงอ็อบเจกต์นั้นมีความน่าจะเป็นที่จะเป็นเอ้าท์ไลเออร์สูงด้วย

ในความไร้ประโยชน์ของเอ้าท์ไลเออร์ก็ยังมีสิ่งที่ดีๆ แฝงอยู่เสมอ แม้ว่าเอ้าท์ไลเออร์จะทำให้การค้นหารูปแบบที่สามารถอธิบายข้อมูลส่วนใหญ่คลาดเคลื่อนไป แต่อย่างไรก็ตามกระบวนการค้นหาและตรวจสอบเอ้าท์ไลเออร์ก็เป็นสิ่งสำคัญและเป็นประโยชน์สำหรับงานบางอย่างซึ่งต้องการค้นหารูปแบบของข้อมูลที่ต่างไปจากข้อมูลทั่วไป ตัวอย่างเช่น การนำเอาเทคนิคการค้นหาเอ้าท์ไลเออร์มาใช้เพื่อตรวจสอบการทุจริต (fraud detection) เช่น การค้นหารูปแบบการใช้บริการ

บัตรเครดิตที่ผิดปกติ การวิเคราะห์รูปแบบของการตอบสนองที่ผิดปกติของผู้ป่วยต่อการรักษาโรค เป็นต้น

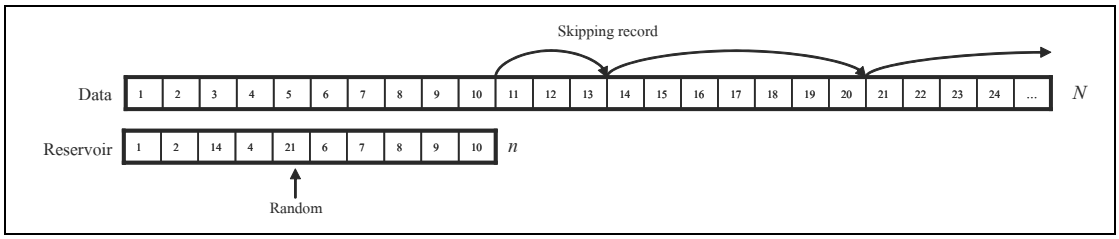
2.4 เทคนิคการสุ่มข้อมูลเพื่องานการจัดกลุ่มข้อมูล

สำหรับงานวิจัยนี้จะเน้นการพัฒนาเทคนิคการลดขนาดด้วยวิธีการสุ่มข้อมูลเป็นหลัก และเพื่อให้การสุ่มข้อมูลทำงานได้อย่างรวดเร็วและเหมาะสมสำหรับข้อมูลขนาดใหญ่ที่ต้องมีการรับข้อมูลเข้ามาอย่างต่อเนื่อง ดังนั้นเทคนิคการสุ่มข้อมูลที่เหมาะสมจึงควรเป็นเทคนิคการสุ่มข้อมูลแบบต่อเนื่องเช่นกัน โดยที่จะต้องสามารถสุ่มข้อมูลได้อย่างสมบูรณ์ภายในการสแกนข้อมูลเพียงหนึ่งรอบ

Vitter (1985) ได้เสนอเทคนิคการสุ่มข้อมูลแบบสะสม ในสมมุติฐานที่ไม่สามารถทราบจำนวนข้อมูลทั้งหมดมาก่อน เทคนิคนี้เป็นการสุ่มข้อมูลอย่างง่ายแบบไม่ใส่กลับอีกรูปแบบหนึ่ง ซึ่งนับว่าเป็นเทคนิคการสุ่มแบบต่อเนื่องที่เป็นที่รู้จักอันดับต้นๆ อันเป็นที่ยอมรับและเป็นพื้นฐานของการพัฒนาเทคนิคการสุ่มในแบบต่างๆ ทั้งนี้เทคนิคการสุ่มข้อมูลอย่างง่ายยังคงมีข้อจำกัดสำหรับข้อมูลที่มีการกระจายตัวของข้อมูลแบบไม่ปกติ เนื่องจากการสุ่มข้อมูลอย่างง่ายอาจทำให้กลุ่มข้อมูลขนาดเล็กอาจถูกละเลยไปได้ ซึ่งต่อมา Palmer and Faloutsos (2000) ได้เสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่น สำหรับข้อมูลที่มีการกระจายแบบไม่ปกติ โดยการเบี่ยงเบนค่าความน่าจะเป็นที่ใช้ในการสุ่มข้อมูลแต่ละตัว โดยเพิ่มอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกลุ่มกันอย่างเบาบาง และลดอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกลุ่มกันอย่างหนาแน่น ซึ่งด้วยเทคนิคนี้ทำให้กลุ่มข้อมูลขนาดเล็กและมีความหนาแน่นเบาบางยังคงสามารถปรากฏในชุดข้อมูลสุ่มได้ แต่อย่างไรก็ตามการที่ให้ความสำคัญกับบริเวณที่มีข้อมูลรวมกลุ่มกันอย่างเบาบางมากขึ้นนั้นอาจเป็นการเพิ่มโอกาสให้ข้อมูลรบกวนสามารถเข้ามามีบทบาทในชุดข้อมูลสุ่มได้ ซึ่งต่อมา K. Kerdprasop, N. Kerdprasop and Sun (2005) ได้เสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นแบบสะสม โดยเป็นเทคนิคที่ผสมผสานทั้งสองเทคนิคดังกล่าวไว้แล้วข้างต้นเข้าด้วยกัน อัลกอริธึมเริ่มต้นจากการแบ่งข้อมูลออกเป็นกลุ่มย่อยที่เรียงต่อกัน จากนั้นจึงใช้หลักการสุ่มข้อมูลแบบสะสมเลือกกลุ่มข้อมูลที่มีความหนาแน่นมากกว่าระหว่างสองกลุ่มข้อมูลที่อยู่ใกล้กัน โดยที่ผลต่างหรือผลรวมของความหนาแน่นของสองกลุ่มข้อมูลนั้นมีค่าเกินกว่าค่าที่กำหนดไว้ ด้วยการเลือกเอาเฉพาะกลุ่มข้อมูลที่มีความหนาแน่นมากนี่เองทำให้อัลกอริธึมสามารถกรองข้อมูลรบกวนไม่ให้ถูกเลือกเข้าไปในชุดข้อมูลสุ่มได้ จึงถือได้ว่าเทคนิคการสุ่มข้อมูลดังกล่าวเป็นเทคนิคที่ทนทานต่อข้อมูลรบกวน ซึ่งทั้งสามเทคนิคนี้จะได้กล่าวโดยละเอียดในหัวข้อถัดไป

2.4.1 เทคนิคการสุ่มข้อมูลแบบสะสม (Random Sampling with a Reservoir: RVS)

Vitter (1985) ได้เสนอเทคนิคการสุ่มแบบสะสม ซึ่งเป็นเทคนิคที่ใช้เลือกข้อมูลสุ่มจำนวน n เรคคอร์ดแบบไม่ใส่กลับ จากข้อมูลทั้งหมด N เรคคอร์ด โดยที่ไม่ทราบค่าของ N มาก่อนล่วงหน้า และ $n \leq N$ โดยสามารถแสดงผลชุดข้อมูลสุ่มได้ภายในการอ่านข้อมูลเพียงหนึ่งรอบ ดังตัวอย่างของปัญหาที่เกิดขึ้นจากการสุ่มข้อมูลที่ถูกบันทึกไว้บนแถบแม่เหล็กซึ่งมีความยาวมาก ด้วยเทคนิคปกติจะต้องทำการอ่านข้อมูลหนึ่งรอบเพื่อหาค่า N จากนั้นจึงจะเริ่มกระบวนการสุ่มข้อมูลโดยการอ่านแถบแม่เหล็กในรอบถัดไป ซึ่งการอ่านข้อมูลบนแถบแม่เหล็กในแต่ละรอบเป็นงานที่สูญเสียเวลาเป็นอันมาก ดังนั้น Vitter จึงเสนออัลกอริธึมแบบสะสม (reservoir algorithm) เพื่อรองรับกับปัญหาดังกล่าว โดยการจองพื้นที่หน่วยความจำไว้ส่วนหนึ่งเพื่อเก็บข้อมูลที่ถูกเลือกซึ่งเรียกหน่วยความจำส่วนนี้ว่าที่สะสมหรือ reservoir



รูปที่ 2.17 ภาพจำลองกระบวนการสุ่มแบบสะสม

```
{Make the first n records candidates for the sample}
for j := 0 to n - 1 do READ_NEXT_RECORD(C[M]);
        t := n;                          {t is the number of records processed so far}
while not eof do
        begin                                        {Process the rest of the records}
        Generate an independent random variate P(n, t);
        SKIP_RECORDS(P);                          {Skip over the next P records}
        if not eof then
        begin          {Make the next record a candidate, replacing one at random}
          M:= TRUNC (n * RANDOM( )); M is uniform in the range 0 ≤ M ≤ n – 1}
          READ_NEXT_RECORD(C[M])
        end
        t := t + P + 1;
end;
```

รูปที่ 2.18 อัลกอริธึมสุ่มแบบสะสม (Vitter, 1985)

หลักการพื้นฐานของ reservoir algorithm คือ กระบวนการเลือกข้อมูลจำนวน n ตัว จากข้อมูลทั้งหมด N ตัว โดยเริ่มจากการเลือกข้อมูล n ตัวแรกใส่ไว้ใน reservoir จากนั้นจึงอ่านข้อมูลถัดไปตามลำดับ โดยอัลกอริธึมจะทำการสุ่มค่าจำนวนข้อมูลที่จะกระโดดข้ามไปเพื่อเพิ่มความเร็วสำหรับการอ่านข้อมูลในแต่ละครั้ง (รูปที่ 2.17) ซึ่งข้อมูลปัจจุบันที่กำลังถูกอ่านจะถูกเก็บเข้ามาไว้ใน reservoir โดยการสุ่มเลือกข้อมูลเดิมใน reservoir และแทนที่ข้อมูลเดิมนั้นด้วยข้อมูลใหม่ ข้อมูลทั้งหมดที่ถูกเก็บไว้ใน reservoir ณ ขณะใดๆ สามารถแทนชุดข้อมูลสุ่ม ณ ขณะนั้นๆ ได้ และเมื่อการอ่านข้อมูลเสร็จสิ้น ข้อมูลที่อยู่ใน reservoir จะเป็นชุดข้อมูลสุ่มสุดท้ายของ reservoir algorithm รายละเอียดของอัลกอริธึมแสดงได้ดังรูปที่ 2.18

reservoir algorithm ได้ถูกปรับปรุงเรื่อยมาจนถึง algorithm Z ซึ่งสามารถทำงานได้ภายในเวลา $O\left(n\left(1+\log\frac{N}{n}\right)\right)$ ซึ่งรายละเอียดของ algorithm Z ได้แสดงไว้ดังต่อไปนี้

```
{(Make the first n records candidates for the sample}
for j := 0 to n - 1 do READ_NEXT_RECORD(C[j]);
t := n;                                   { t is the number of records processed so far}
{Process records using the method of Algorithm X until t is large enough}
thresh := T * n;
num := 0;                                                  {num is equal to t – n}
while not eof and (t ≤ thresh) do
   begin
   V := RANDOM( );                                                  {Generate V}
   P := 0;
   t := t + 1;     num := nun + 1;
   quot := num / t;
   while quot > V do                                                {Find min P}
      begin
      P := P + 1;
      t := t + 1;     num := num + 1;
      quot := (quot * num) / t
   end;
   SKIP_RECORDS(P);                                {Skip over the next P records}
   if not eof then
      begin                {Make the next record a candidate, replacing one at random}
      M := TRUNC(n * RANDOM( ));        {M is uniform in the range 0 ≤ M ≤ n – 1}
      READ_NEXT_RECORD(C[M ]);
      end
   end;


{Process the rest of the records using the rejection technique}
```

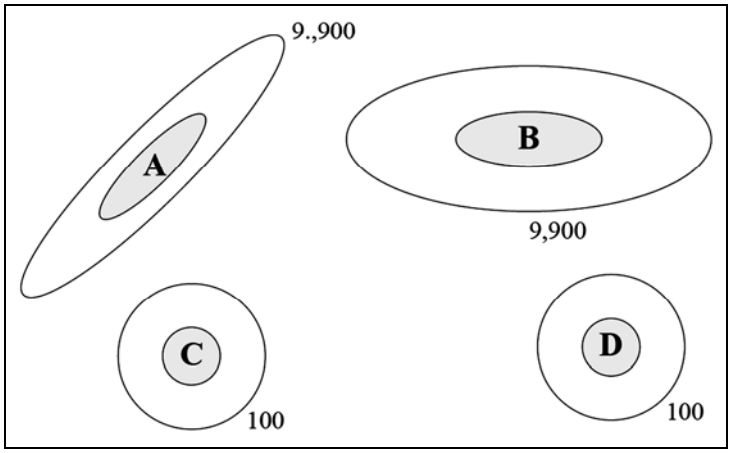
```
W := EXP(- LOG(RANDOM( )) / n);                          {Generate W}
term := t - n + 1;                                {term is always equal to t - n + 1}
while not eof do
  begin
    loop
    {Generate U and X }
    U := RANDOM( );
    X := t * (W - 1.0);
    P := TRUNC(X);                                    {P is tentatively set to ⌊X⌋}
    {Test if U ≤ h(P) / cg(X)}
    lhs := EXP(LOG(((U (((t + l)/term) T^2)) * (term + P))/(t + X)) / n);
    rhs := (((t + X)/(term + P)) * term)/t;
    if lhs ≤ rhs then
    begin W := rhs/lhs;  break loop end;
    {Test if U ≤ f(P) / cg(X)}
    y := (((U * (t + l))/term) * (t + P + l))/(t + X);
    if  n < P then begin   denom := t;    numer_lim := term + 50   end
    else begin   denom := t - n + P;     numer_lim := t + 1    end;
    for numer := t + P downto numer_lim do
      begin   y := (y * numer)/denom;   denom := denom - 1   end;
    W := EXP(- LOG(RANDOM( ))/n);                     {Generate W in advance}
     if EXP(LOG(y)/n) ≤ (t + X)/t then break loop
  end loop;
  SKIP_RECORDS(P);                                    {Skip over the next P records}
  if not eof then
    begin                {Make the next record a candidate, replacing one at random}
    M := TRUNC (n * RANDOM( ));     { M is uniform in the range 0 ≤ M ≤ n – 1}
    READ_NEXT_RECORD(C[M])
    end;
  t := t + P + 1;
  term := term + P + 1
  end;
```

แม้ว่า reservoir algorithm จะสามารถสร้างชุดข้อมูลสุ่มที่สมบูรณ์ได้ภายในการอ่านข้อมูลเพียงหนึ่งรอบเท่านั้น แต่อย่างไรก็ตามการใช้ reservoir algorithm กับชุดข้อมูลที่มีขนาดใหญ่มากกว่าขนาดของ reservoir มากๆ อาจทำให้ข้อมูลสุ่มที่ได้มาจากข้อมูลส่วนท้ายๆ มากกว่าข้อมูลส่วนต้นก็เป็นได้ เพราะเมื่ออัลกอริธึมตัดสินใจจะรับข้อมูลใหม่เข้ามาเก็บใน reservoir อัลกอริธึมจะทำการสุ่มข้อมูลเก่าทิ้งไปโดยไม่มีการพิจารณาเปรียบเทียบความสำคัญหรือความน่าจะเป็นที่ข้อมูลเก่าหรือข้อมูลใหม่จะถูกเลือกมากกว่ากัน ด้วยเหตุนี้ข้อมูลเก่าจึงอาจถูกแทนที่ด้วยข้อมูลใหม่เสมอ ซึ่งหากคลัสเตอร์ที่มีขนาดเล็กเรียงตัวอยู่ในข้อมูลส่วนต้นๆ อาจทำให้คลัสเตอร์ดังกล่าวไม่ปรากฏ

ในชุดข้อมูลสุ่มได้

2.4.2 เทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่น (Density Biased Sampling: DBS)

สืบเนื่องมาจากงานทางด้านการทำเหมืองข้อมูลนั้นเป็นงานที่ต้องกระทำกับข้อมูลขนาดใหญ่ซึ่งมักต้องใช้เทคนิคการลดขนาดข้อมูลเข้ามาเกี่ยวข้องในขั้นตอนการเตรียมข้อมูลเสมอ เพื่อให้กระบวนการสังเคราะห์ข้อมูลดำเนินไปอย่างมีประสิทธิภาพ เทคนิคการลดขนาดข้อมูลโดยการสุ่มแบบสม่ำเสมอเป็นเทคนิคที่นิยมนำมาใช้ แต่อย่างไรก็ตามสำหรับงานทางด้านการจัดกลุ่มข้อมูลซึ่งต้องการค้นหาลักษณะร่วมกันของข้อมูลในทุกกลุ่ม การใช้เทคนิคดังกล่าวยังมีข้อจำกัดสำหรับข้อมูลที่การกระจายตัวแบบไม่ปกติหรือ zipf's distribution (ข้อมูลที่ประกอบด้วยคลัส-เตอร์ที่มีขนาดและความหนาแน่นแตกต่างกัน) ซึ่งเป็นลักษณะของข้อมูลทั่วไปตามธรรมชาติ การใช้เทคนิคการสุ่มแบบสม่ำเสมอ (uniform sampling) อาจทำให้คลัสเตอร์ที่มีขนาดเล็กมักถูกละเลย เป็นผลให้กระบวนการจัดกลุ่มข้อมูลไม่สามารถค้นพบคลัสเตอร์เหล่านั้นจากชุดข้อมูลสุ่มได้ ตัวอย่างเช่น ข้อมูลตัวอย่างซึ่งประกอบด้วยข้อมูล 4 คลัสเตอร์ คือ คลัสเตอร์ A, B, C, และ D ดังรูปที่ 2.19



รูปที่ 2.19 ข้อมูลตัวอย่างซึ่งประกอบด้วย 4 คลัสเตอร์ (Palmer and Faloutsos, 2000)

โดยในแต่ละคลัสเตอร์มีข้อมูล 9900, 9900, 100, และ 100 เรคคอร์ด ตามลำดับ การสุ่มแบบสม่ำเสมอขนาด 1% สามารถคาดได้ว่าข้อมูลประมาณ 99 เรคคอร์ดจะมาจากแต่ละคลัสเตอร์ A และ B และอีก 1 เรคคอร์ดจะมาจากแต่ละคลัสเตอร์ C และ D ซึ่งข้อมูลที่มีจำนวนน้อยมากๆ สำหรับคลัสเตอร์ C และ D นั้นจะถูกจัดให้เป็นข้อมูลรบกวนโดยอัลกอริธึมจัดกลุ่มข้อมูล อันเป็นผลให้คลัสเตอร์ C และ D ถูกละเลยไปได้

Palmer and Faloutsos (2000) ได้เสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นของข้อมูลเพื่อให้คลัสเตอร์ขนาดเล็กยังคงสามารถปรากฏบนชุดข้อมูลสุ่ม หลักการพื้นฐานคือการสุ่มด้วยค่าความน่าจะเป็นที่ข้อมูลใดๆ จะถูกเลือกเข้ามาในชุดข้อมูลสุ่มซึ่งแปรผกผันกับความหนาแน่นของข้อมูลบริเวณนั้นๆ หรืออีกนัยหนึ่งคือการเพิ่มอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่เบาบางและลดอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่หนาแน่น โดยที่ความหนาแน่นของข้อมูลจะมีค่าแปรผันตรงกับจำนวนข้อมูลในบริเวณนั้น

เทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นของข้อมูล เป็นเทคนิคที่ใช้เลือกข้อมูลขนาด M ตัว จากข้อมูลทั้งหมด N ตัวซึ่งประกอบด้วยข้อมูลย่อย $x_1, x_2, x_3 \ldots x_N$ โดยเริ่มต้นจากการแบ่งข้อมูลทั้งหมดออกเป็นกลุ่มย่อยๆ g กลุ่มที่มีพื้นที่เท่าๆ กัน โดยที่ข้อมูลย่อย x_i จะถูกจัดให้อยู่ในกลุ่มย่อย โดยที่ n_{x_i} คือขนาดของข้อมูลในกลุ่มนั้น จากนั้นจึงทำการสุ่มแบบสม่ำเสมอด้วยค่าความน่าจะเป็น (P) เท่าๆ กันสำหรับข้อมูลทุกตัวภายในกลุ่มย่อยเดียวกัน คือ

$$P = \frac{M}{n_x^e \sum_{i=1}^{g} n_g^{1-e}} \quad (e \text{ คือค่าคงที่ใดๆ})$$

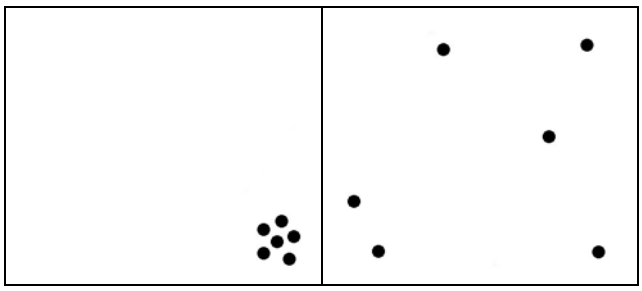
เพื่อให้กระบวนการสุ่มดำเนินไปอย่างสมบูรณ์ด้วยการอ่านข้อมูลเพียงหนึ่งรอบ ขั้นตอนการแบ่งกลุ่มข้อมูลและขั้นตอนการสุ่มข้อมูลจะต้องดำเนินไปพร้อมๆกันโดยการนำขั้นตอนดังกล่าวมาประยุกต์ร่วมกับ reservoir algorithm และการใช้ฟังก์ชัน hash ในการจัดกลุ่มย่อยให้กับข้อมูลใดๆ ดังอัลกอริธึมที่แสดงไว้ดังรูปที่ 2.20

อัลกอริธึมเริ่มต้นด้วยการอ่านข้อมูลเข้ามาและพิจารณาว่าจะจัดข้อมูลนั้นให้อยู่ในกลุ่มย่อยไหนโดยใช้ฟังก์ชัน hash พร้อมทั้งคำนวณค่าความน่าจะเป็นที่ข้อมูลนั้นจะถูกเลือกเข้ามาไว้ใน reservoir ซึ่งในที่นี้จะเรียกว่า output buffer และเมื่อกระบวนการดำเนินไปจน output buffer ไม่สามารถรองรับข้อมูลใหม่ได้ อัลกอริธึมจะทำการลดขนาดข้อมูลที่เก็บไว้ลงโดยการคำนวณค่าความน่าจะเป็นที่ข้อมูลที่เก็บไว้เดิมนั้นควรที่จะคงไว้หรือไม่ เพราะค่าความน่าจะเป็นที่ข้อมูลใดๆ จะถูกเลือกเข้ามาไว้ใน reservoir จะเปลี่ยนไปเรื่อยๆ เมื่อมีการอ่านข้อมูลใหม่เข้ามา และเมื่อการอ่านข้อมูลเสร็จสิ้น ข้อมูลที่อยู่ใน output buffer จะเป็นชุดข้อมูลสุ่มสุดท้ายของอัลกอริธึม ส่วนค่าพารามิเตอร์ e เป็นพารามิเตอร์ที่ใช้ปรับอัตราการเบี่ยงเบนของการสุ่มข้อมูล ซึ่งสามารถกำหนดได้ตามความต้องการโดยถ้าให้ $e = 0$ จะทำให้ได้ชุดข้อมูลสุ่มแบบสม่ำเสมอ (ไม่มีการเบี่ยงเบนตามความหนาแน่น) $e = 1$ จะเป็นการสุ่มโดยคาดหวังให้ข้อมูลในแต่ละกลุ่มย่อยถูกเลือกขึ้นมาในจำนวนที่เท่ากัน จากการทดลองพบว่าการให้ค่า $e = 0.5$ จะทำให้ได้ผลลัพธ์ที่ดีที่สุด

$\alpha_D = 0$
FOR each input point x DO
  IF $n[h(x)] \neq 0$ THEN $\alpha_D = \alpha_D - n[h(x)]^{1-e}$ (*)
  $n[h(x)] = n[h(x)] + 1$
  $\alpha_D = \alpha_D + n[h(x)]^{1-e}$ (*)
  WITH prob. $P = min\{M/(\alpha_D * n[h(x)]^e), 1\}$ DO
    IF the output buffer is full THEN *reduce*()
    add $< P, x >$ to the output buffer
reduce()
FOR each output buffer entry $< P_i, x_i >$ DO output $< 1/P_i, x_i >$

reduce() is
  FOR each output buffer entry $< P_i, x_i >$ DO
    Let $P_i' = min\{M/(\alpha_D * n[h(x)]^e), 1\}$
    WITH prob. P_i'/P_i replace this entry with $< P_i', x_i >$
    OTHERWISE remove this entry

รูปที่ 2.20 อัลกอริธึมสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่น (Palmer and Faloutsos, 2000)



(a) (b)

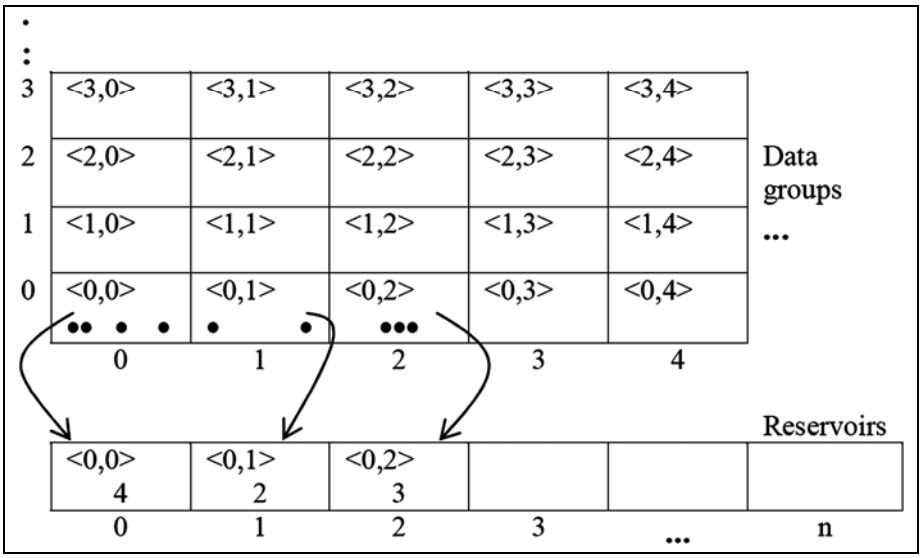
รูปที่ 2.21 แสดงตัวอย่างกลุ่มข้อมูลย่อยสองกลุ่มที่มีจำนวนข้อมูลเท่ากัน

แม้ว่า DBS จะถูกออกแบบมาเพื่อให้สามารถเพิ่มอัตราการสุ่มข้อมูลในบริเวณที่ข้อมูลมีความหนาแน่นเบาบางเพื่อให้คลัสเตอร์ขนาดเล็กยังคงปรากฏในชุดข้อมูลสุ่มได้ แต่จากการพิจารณาเฉพาะจำนวนข้อมูลโดยมิได้ให้ความสำคัญกับความใกล้ชิดกันของข้อมูลร่วมด้วย ซึ่งข้อมูลส่วนนั้นอาจไม่ใช่กลุ่มข้อมูลขนาดเล็กที่ต้องการก็เป็นได้ และนั่นยังหมายความรวมถึงการเพิ่มอัตราการถูกเลือกเข้ามาของข้อมูลรบกวนซึ่งเป็นตัวสร้างผลกระทบต่อความแม่นตรงของกระบวนการค้นหากลุ่มข้อมูลอีกด้วย ดังรูปที่ 2.21 แสดงกลุ่มข้อมูลย่อยสองกลุ่มที่มีจำนวนข้อมูลเท่ากัน โดยจากกรณีนี้ DBS จะให้ค่าความน่าจะเป็นที่ข้อมูลจะถูกเลือกเท่ากันทั้งสองกลุ่มในขณะที่ข้อมูลในกลุ่มย่อย (a) มีความใกล้ชิดกันมากกว่าข้อมูลในกลุ่มย่อย (b)

2.4.3 เทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นแบบสะสม (Density Biased Reservoir Sampling: DBRVS)

K. Kerdprasop et al. (2005) ได้เสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นแบบสะสม ซึ่งเป็นเทคนิคที่มีความพยายามที่จะแก้ไขปัญหาที่เกิดขึ้นจากข้อมูลรบกวนบนชุดข้อมูล อีกทั้งยังเพิ่มประสิทธิภาพในการใช้งานพื้นที่ในหน่วยความจำให้สามารถรองรับกับข้อมูลขนาดใหญ่มากๆ ได้

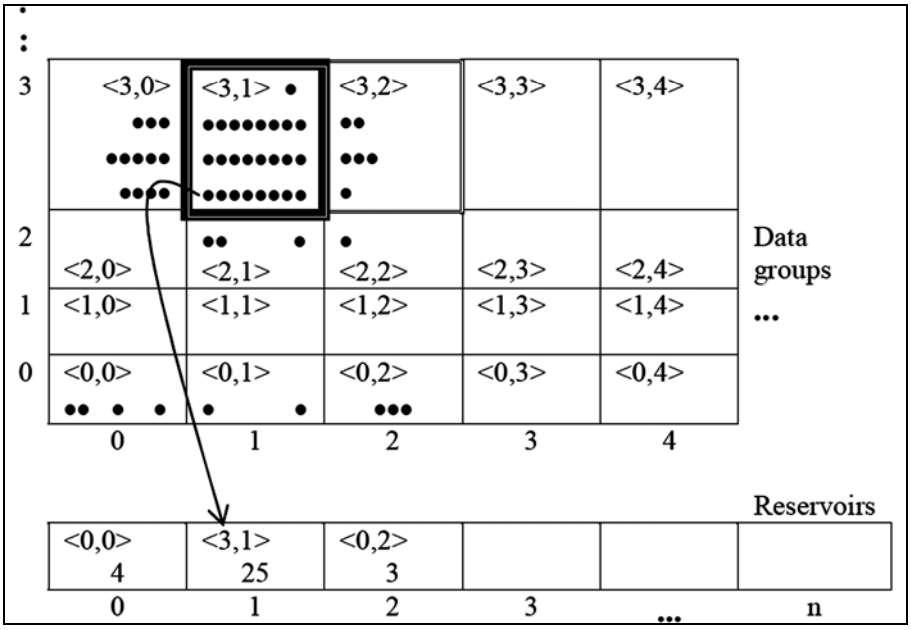
กระบวนการสุ่มข้อมูลด้วยเทคนิคนี้ได้ยึดเอาหลักการของ reservoir algorithm เป็นหลัก เริ่มต้นจากการแบ่งข้อมูลทั้งหมดออกเป็นกลุ่มย่อยโดยใช้วิธีการแบ่งกลุ่มเช่นเดียวกันกับเทคนิคของ Palmer and Faloutsos (2000) ซึ่งจะเก็บรายละเอียดของกลุ่มข้อมูลตามลำดับการเรียงตัวของข้อมูลไว้ในหน่วยความจำ จากนั้นจึงดำเนินการสุ่มตามแบบ reservoir algorithm โดยการกระทำกับแถวลำดับของกลุ่มข้อมูลในหน่วยความจำนั้น ในขั้นแรกอัลกอริธึมจะทำการเลือกกลุ่มข้อมูล n กลุ่มแรกใส่ไว้ใน reservoir ซึ่งจะเก็บเฉพาะข้อมูลของกลุ่มที่ประกอบด้วยจำนวนข้อมูลและรหัสของกลุ่มนั้นๆ ดังรูปที่ 2.22



รูปที่ 2.22 แสดงการให้ค่าเริ่มต้นกับ reservoir ของ DBRVS (Kerdprasop et al., 2005)

หลังจากการให้ค่าเริ่มต้นกับ reservoir แล้ว อัลกอริธึมจะทำการอ่านกลุ่มข้อมูลถัดไปตามลำดับ โดยจะทำการสุ่มค่าจำนวนกลุ่มข้อมูลที่จะกระโดดข้ามไปเพื่อเพิ่มความเร็วสำหรับการอ่านกลุ่มข้อมูลในแต่ละครั้ง ความแตกต่างของเทคนิคนี้อยู่ที่วิธีการพิจารณาเลือกกลุ่มใดๆ เข้ามาเก็บไว้ใน reservoir โดยจะใช้วิธีการพิจารณากลุ่มข้อมูลสองกลุ่มที่อยู่ใกล้กัน กล่าวคือถ้าความแตกต่างของความหนาแน่นของสองกลุ่มนั้นมีค่าเกินกว่าค่า δ (เพื่อการค้นหาขอบเขตของคลัส-

เตอร์) หรือผลรวมของความหนาแน่นของสองกลุ่มนั้นมีค่าเกินกว่าค่า ε (เพื่อลดปัญหาการเลือกข้อมูลรบกวนเข้ามา) กลุ่มข้อมูลที่มีค่าความหนาแน่นมากกว่าจะถูกเลือกให้เข้าไปเก็บไว้ใน reservoir โดยการสุ่มเลือกกลุ่มข้อมูลเดิมใน reservoir และแทนที่กลุ่มข้อมูลเดิมนั้นด้วยกลุ่มข้อมูลใหม่ (ดังรูปที่ 2.23)



รูปที่ 2.23 การปรับปรุง reservoir เมื่อกลุ่มข้อมูลใหม่ถูกเลือก (Kerdprasop et al., 2005)

เมื่อการอ่านข้อมูลเสร็จสิ้น อัลกอริธึมจะทำการแปลงกลุ่มข้อมูลที่ถูกเลือกใน reservoir จากรายละเอียดของกลุ่มที่เก็บไว้กลับไปเป็นข้อมูลย่อยๆ ดังเดิม ซึ่งข้อมูลที่ได้จะเป็นชุดข้อมูลสุ่มสุดท้ายของเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นแบบสะสม ซึ่งอัลกอริธึมสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นแบบสะสมแสดงได้ดังรูปที่ 2.24

ด้วยการเลือกเอาเฉพาะกลุ่มข้อมูลที่มีความหนาแน่นมากทำให้ DBRVS สามารถกรองข้อมูลรบกวนไม่ให้ถูกเลือกเข้าไปในชุดข้อมูลสุ่มได้ และถือได้ว่าเทคนิคการสุ่มข้อมูลดังกล่าวเป็นเทคนิคที่ทนทานต่อข้อมูลรบกวน แต่อย่างไรก็ตามเนื่องจากการสุ่มข้อมูลในแต่ละครั้งเป็นการสุ่มเลือกทั้งกลุ่มข้อมูลย่อยจึงอาจทำให้ชุดข้อมูลสุ่มที่ได้ปรากฏคลัสเตอร์ได้ไม่ครบถ้วนถ้าขนาดของชุดข้อมูลสุ่มที่ต้องการมีขนาดไม่ใหญ่พอ อีกทั้งยังอาจพบปัญหาได้เช่นเดียวกับ RVS ซึ่งข้อมูลส่วนต้นจะถูกแทนที่ด้วยข้อมูลส่วนท้ายๆ

Algorithm: Density-biased reservoir sampling
Input: a data set of N objects
Output: a density-biased sample of size n ($n \le N$)
Steps:

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

รูปที่ 2.24 อัลกอริธึมสุ่มแบบเบี่ยงเบนตามความหนาแน่นแบบสะสม (Kerdprasop et al., 2005)

จากที่ได้กล่าวมาทั้งหมดนั้น จะเห็นได้ว่างานการจัดกลุ่มข้อมูลเป็นงานที่ต้องใช้เทคนิคที่ซับซ้อนและมักต้องการข้อมูลที่มีจำนวนมากพอเพื่อความถูกต้องของผลลัพธ์ แต่เนื่องจากการที่ต้องวิเคราะห์ข้อมูลทุกตัวทำให้ต้องสิ้นเปลืองทั้งทรัพยากรและใช้เวลามาก โดยที่ข้อมูลแต่ละตัวนั้นมีความสำคัญมากน้อยไม่เท่ากันต่องานการจัดกลุ่มข้อมูล ดังนั้นการเลือกใช้เฉพาะข้อมูลที่มีความสำคัญหรือข้อมูลที่สามารถเป็นตัวแทนของข้อมูลทั้งหมดได้จึงน่าจะเป็นแนวทางที่ดี การลดขนาดข้อมูลด้วยเทคนิคที่แม่นยำและมีประสิทธิภาพเป็นอีกแนวทางหนึ่งเพื่อจัดการกับปัญหาดังกล่าว เทคนิคหนึ่งที่นิยมใช้สำหรับการลดขนาดข้อมูลคือการสุ่มข้อมูล ซึ่งการสุ่มข้อมูลมีอยู่ด้วยกันหลายวิธี เช่น การสุ่มอย่างง่ายแบบใส่กลับ การสุ่มอย่างง่ายแบบไม่ใส่กลับ เป็นต้น ซึ่งแต่ละวิธีก็มีข้อจำกัดและให้ผลลัพธ์ซึ่งเหมาะกับงานที่แตกต่างกัน สำหรับงานการจัดกลุ่มข้อมูลซึ่งมีวัตถุประสงค์หลักเพื่อค้นหาคลัสเตอร์หรือการรวมกลุ่มกันของข้อมูลจึงต้องการเทคนิคที่เฉพาะเพื่อให้สามารถเลือกชุดข้อมูลสุ่มที่เหมาะสมกับงานที่สุด โดยที่ชุดข้อมูลสุ่มที่ดีจะต้องสามารถคงความเป็นกลุ่มข้อมูลของทุกคลัสเตอร์ของชุดข้อมูลเดิมได้ และเทคนิคการสุ่มที่ดียังจะต้องสามารถตรวจพบและขจัดเอ้าท์ไลเออร์ออกไปได้อีกด้วย ดังนั้นจึงจำเป็นจะต้องนำเทคนิคต่างๆ ที่มีอยู่แล้วมาทำ

การทดสอบเพื่อหาจุดเด่นของแต่ละเทคนิค และเพื่อเป็นการเริ่มต้นที่ดีสำหรับการพัฒนาเทคนิคการสุ่มข้อมูลที่ดีขึ้นต่อไป โดยขั้นตอนและวิธีดำเนินการวิจัยจะได้กล่าวถึงในบทต่อไป

บทที่ 3
วิธีดำเนินการวิจัย

งานวิจัยนี้มีจุดประสงค์เพื่อศึกษาและเปรียบเทียบเทคนิคการสุ่มข้อมูล ตลอดจนพัฒนาเทคนิคการสุ่มข้อมูลที่มีประสิทธิภาพสูงขึ้นเพื่อใช้ในงานการจัดกลุ่มข้อมูลขนาดใหญ่ สำหรับเนื้อหาในบทนี้จะได้นำเสนอขั้นตอนและวิธีดำเนินการวิจัย โดยเริ่มจากระเบียบวิธีวิจัยดังแสดงในหัวข้อที่ 3.1 และหัวข้อที่ 3.2 จะนำเสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นและความเป็นปึกแผ่นของข้อมูล (DBSPACE) ซึ่งเป็นเทคนิคที่ผู้วิจัยได้พัฒนาขึ้นสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่โดยเฉพาะ ในหัวข้อที่ 3.3 จะเป็นรายละเอียดของชุดข้อมูลที่ใช้ในงานวิจัยนี้ และในหัวข้อที่ 3.4 จะเป็นขั้นตอนการเปรียบเทียบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล

3.1 ระเบียบวิธีวิจัย

3.1.1 ศึกษาค้นคว้าและรวบรวมงานวิจัยที่เกี่ยวข้อง

3.1.2 ศึกษาการใช้งานระบบ WEKA เพื่อใช้ในการทดสอบการจัดกลุ่มข้อมูล

3.1.3 รวบรวมและพัฒนาอัลกอริธึมสุ่มข้อมูลที่มีคุณสมบัติเหมาะสมสำหรับงานจัดกลุ่มข้อมูลที่มีนักวิจัยท่านอื่นเคยเสนอไว้แล้ว โดยอัลกอริธึมที่เลือกใช้ในการวิจัยนี้ประกอบด้วย

1) RVS ซึ่งเสนอโดย Vitter (1985)
2) DBS ซึ่งเสนอโดย Palmer and Faloutsos (2000)
3) DBRVS ซึ่งเสนอโดย K. Kerdprasop et al. (2005)

3.1.4 ศึกษาการใช้งานอัลกอริธึมสร้างคลัสเตอร์สังเคราะห์ เพื่อใช้ในการสร้างชุดข้อมูลสำหรับทดสอบอัลกอริธึมสุ่มข้อมูล ซึ่งพัฒนาขึ้นที่ The University of Manchester โดย Handl and Knowles (n.d.)

3.1.5 พัฒนาเครื่องมือที่จำเป็นเพื่อช่วยในการทดลอง

1) CLSCTR เพื่อใช้ในการหาเซนทรอยด์ของคลัสเตอร์จากชุดข้อมูลตั้งต้น
2) Weka2*NC* เพื่อใช้ในการแปลงผลลัพธ์ของคลัสเตอร์ที่ได้จาก WEKA (*.txt) ให้เป็นเซนทรอยด์ เพื่อใช้เปรียบเทียบกับคลัสเตอร์แท้ที่ได้จากชุดข้อมูลตั้งต้น

3) *NC*metric เพื่อใช้ในการเปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับคลัสเตอร์แท้ โดยให้ผลลัพธ์เป็นจำนวนของคลัสเตอร์แท้ที่พบ (*NC*) จากชุดข้อมูลสุ่ม

4) NOISEGEN เพื่อใช้ในการสร้างข้อมูลรบกวนขนาดต่างๆบนชุดข้อมูลที่ต้องการ

3.1.6 ทดสอบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS เพื่อวิเคราะห์หาลักษณะเด่นของแต่ละอัลกอริธึม เพื่อใช้ประกอบการออกแบบอัลกอริธึมใหม่

3.1.7 ออกแบบและพัฒนาอัลกอริธึมสุ่มข้อมูลแบบใหม่ที่มีประสิทธิภาพสูงขึ้น

3.1.8 ทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลแบบใหม่ที่พัฒนาขึ้น

3.1.9 วิเคราะห์และสรุปผลการวิจัย

3.2 เทคนิคการสุ่มแบบเบี่ยงเบนตามความหนาแน่นและความเป็นปึกแผ่นของข้อมูล

ในส่วนนี้จะเป็นการนำเสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นและความเป็นปึกแผ่นของข้อมูล (DBSPACE: A Density-Biased Sampling using Partial Approximate Compactness Estimator) ซึ่งเป็นเทคนิคที่พัฒนาขึ้นสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่โดยเฉพาะ ลักษณะที่สำคัญของอัลกอริธึมมีดังต่อไปนี้

1) สามารถสร้างชุดข้อมูลสุ่มได้ด้วยการอ่านข้อมูลเพียงแค่รอบเดียวเท่านั้น
2) สามารถรองรับข้อมูลที่มีขนาดใหญ่มากและไม่สามารถทราบจำนวนข้อมูลทั้งหมดมาก่อนล่วงหน้าการสุ่มข้อมูล
3) สามารถสร้างชุดข้อมูลสุ่มที่ดีสำหรับข้อมูลที่มีการกระจายแบบไม่ปกติ (zipf)
4) สามารถตรวจสอบและกรองข้อมูลรบกวนไม่ให้ถูกเลือกเข้ามาในชุดข้อมูลสุ่ม โดยใช้ค่าประมาณความเป็นปึกแผ่นของข้อมูลประกอบการสุ่มข้อมูลแต่ละครั้ง

สำหรับเทคนิคการสุ่มข้อมูลแบบใหม่ที่พัฒนาขึ้นนั้น (DBSPACE) เป็นเทคนิคที่ใช้โครงสร้างหลักของอัลกอริธึม DBS คือการเพิ่มอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่เบาบางและลดอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่หนาแน่น และปรับปรุงอัลกอริธึมโดยการเพิ่มเติมคุณสมบัติที่ขาดหายไป ได้แก่คุณสมบัติการทนทานต่อข้อมูลรบกวนโดยการพิจารณาความเป็นปึกแผ่นของข้อมูลข้างเคียงประกอบการสุ่มข้อมูลในแต่ละครั้งด้วย ข้อมูลที่มีความเป็นปึกแผ่น (compactness) หรือมีการรวมกลุ่มกันสูงกว่าจะมีความน่าจะเป็นที่จะถูกเลือกสูงกว่าบริเวณที่ข้อมูลอยู่อย่างกระจัดกระจาย ทำให้เทคนิคนี้สามารถลดอัตราการถูกเลือกเข้าไปของข้อมูลรบกวนในชุดข้อมูลที่ต้องการได้

3.2.1 Compactness and Discordancy Estimator

ในหัวข้อนี้จะกล่าวถึงการคำนวณค่าต่างๆ ที่จำเป็นสำหรับอัลกอริธึม DBSPACE ซึ่งจะเริ่มจากการให้คำจำกัดความของ centroid, diameter, compactness, weighted discordancy, และ non-weighted discordancy โดยให้ $\{\vec{x}_i\}$ เป็นเซ็ตข้อมูลขนาด d มิติจำนวน N อ็อบเจกต์ของคลัส-เตอร์ใดๆ บน data space โดยที่ $i = 1, 2, \ldots, N$

Centroid ($\vec{\mu}$) หรือจุดศูนย์กลางมวลของคลัสเตอร์ เป็นจุดที่ใช้เป็นตัวแทนของทุกอ็อบเจกต์ในคลัสเตอร์นั้น ซึ่งหาได้จาก

$$\vec{\mu} = \frac{\sum_{i=1}^{N} \vec{x}_i}{N}$$

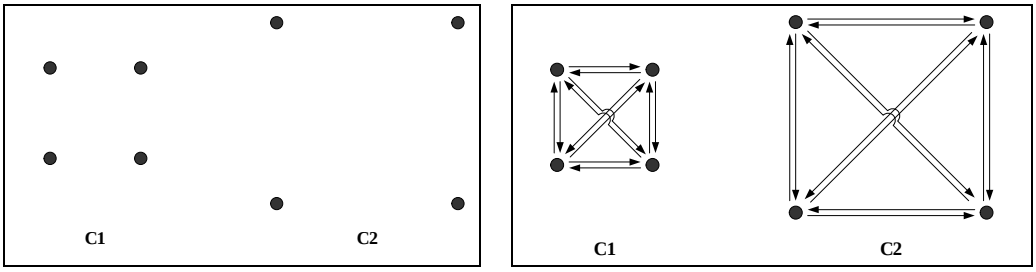
Diameter (D) หรือเส้นผ่านศูนย์กลางของคลัสเตอร์ เป็นค่าระยะทางเฉลี่ยของทุกคู่อ็อบเจกต์ภายในคลัสเตอร์เดียวกัน ให้ $d(\vec{x}_i, \vec{x}_j)$ เป็นระยะทางตามแบบยูคลิด (euclidean distance) ระหว่างอ็อบเจกต์ $\vec{x}_i$ กับ $\vec{x}_j$ โดยที่ $x_{i1}, x_{i2}, ..., x_{id}$ และ $x_{j1}, x_{j2}, ..., x_{jd}$ คือค่าของอ็อบเจกต์ $\vec{x}_i$ และ $\vec{x}_j$ ของแอททริบิวต์ที่ 1, 2, …, d ตามลำดับ diameter ของคลัสเตอร์สามารถหาได้จาก

$$D = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N} d(\vec{x}_i, \vec{x}_j)^2}{N(N-1)}}$$

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \cdots + \left(x_{id} - x_{jd}\right)^2}$$

จะเห็นได้ว่าค่า diameter ของคลัสเตอร์สามารถแสดงถึงความใกล้ชิดกันระหว่างอ็อบเจกต์ภายในคลัสเตอร์ได้ โดย D จะมีค่าน้อยๆ เมื่ออ็อบเจกต์ภายในคลัสเตอร์มีความเป็นปึกแผ่นหรือมีความหนาแน่นมาก และ D จะมีค่ามากเมื่ออ็อบเจกต์ภายในคลัสเตอร์อยู่กันอย่างกระจัดกระจาย ดังรูปที่ 3.1 แสดงตัวอย่างข้อมูลที่ประกอบด้วย 2 คลัสเตอร์คือ C1 และ C2 ที่มีจำนวนอ็อบเจกต์ภายในเท่ากันแต่มีความหนาแน่นต่างกัน โดยรูปที่ 3.1a จะเห็นได้ว่าคลัสเตอร์ C1 มีความหนาแน่นและความเป็นปึกแผ่นของข้อมูลสูงกว่าคลัสเตอร์ C2 และเมื่อลากเส้นเชื่อมระหว่างอ็อบเจกต์ใดๆ ไปยังทุกอ็อบเจกต์ภายในคลัสเตอร์ (ดังรูปที่ 3.1b) และนำความยาวกำลังสองของเส้นเชื่อมเหล่านั้นมารวมกันแล้วหารด้วยจำนวนเส้นเชื่อมทั้งหมด (12 เท่ากันทั้งสองคลัส-เตอร์) และนำมาถอดรากที่สองจะพบว่าค่าที่ได้จากคลัสเตอร์ C1 น้อยกว่าค่าที่ได้จากคลัสเตอร์ C2 ซึ่งแสดงให้เห็น

ว่าอ็อบเจกต์ของคลัสเตอร์ C1 มีความเป็นปึกแผ่นสูงกว่าอ็อบเจกต์ของคลัสเตอร์ C2 และยังเป็นเครื่องบอกได้ว่าอ็อบเจกต์ของคลัสเตอร์ C1 น่าจะมีความสำคัญในกระบวนการจัดกลุ่มข้อมูลสูงกว่าอ็อบเจกต์ของคลัสเตอร์ C2 (พิจารณาจากความเป็นปึกแผ่นของข้อมูลซึ่งอธิบายได้จากค่า diameter)



(a) (b)

รูปที่ 3.1 แสดงตัวอย่างของสองคลัสเตอร์ที่มีความหนาแน่นต่างกัน

แต่อย่างไรก็ตามแม้ว่าค่า diameter จะสามารถบอกถึงระดับความเป็นปึกแผ่นของข้อมูลได้ แต่การหาค่าดังกล่าวจำเป็นต้องอาศัยการคำนวณค่อนข้างมากอีกทั้งยังต้องอ่านข้อมูลหลายรอบเพื่อหาความแตกต่างของทุกคู่อ็อบเจกต์ ซึ่งต้องใช้เวลาเป็น $O(n^2)$ เพราะในแต่ละคู่ของอ็อบเจกต์จะมีการคำนวณระยะทางถึงสองรอบ (ทั้งไปและกลับ) เช่น การคำนวณ diameter ซึ่งประกอบด้วยอ็อบเจกต์ a และอ็อบเจกต์ b จะต้องคำนวณระยะทางระหว่างอ็อบเจกต์ทั้งสองทั้งสิ้นสองครั้ง คือคำนวณระยะทางจากอ็อบเจกต์ a ไปอ็อบเจกต์ b และระยะทางจากอ็อบเจกต์ b ไป อ็อบเจกต์ a ซึ่งเป็นการคำนวณที่ซ้ำซ้อนและไม่เหมาะสำหรับการสุ่มข้อมูลแบบต่อเนื่องซึ่งไม่ทราบจำนวนข้อมูลทั้งหมดล่วงหน้าก่อนการสุ่ม การสุ่มข้อมูลแบบต่อเนื่องโดยไม่ทราบจำนวนข้อมูลทั้งหมดล่วงหน้าจำเป็นจะต้องสามารถสร้างชุดข้อมูลสุ่มได้สมบูรณ์ภายในการอ่านข้อมูลเพียงรอบเดียวเท่านั้น ดังนั้นงานวิจัยนี้จึงได้เสนอเครื่องมือที่ใช้วัดค่าความเป็นปึกแผ่นของข้อมูลที่เรียกว่า compactness

Compactness (Θ) หรือค่าความเป็นปึกแผ่นของข้อมูลแบบประมาณโดยใช้การอ่านข้อมูลเพียงรอบเดียวเท่านั้น โดยค่า compactness เป็นส่วนกลับของ discordancy (Γ) หรือความไม่ประสานกันภายในกลุ่มข้อมูล โดยค่า compactness สามารถคำนวณได้จาก

$$\Theta = \frac{1}{\Gamma} \tag{3-1}$$

สำหรับค่า discordancy สามารถคำนวณได้จากความแตกต่างของทุกอ็อบเจกต์ภายในคลัสเตอร์ ซึ่งคล้ายกับการคำนวณค่า diameter แต่จะเป็นการคำนวณโดยประมาณจากการอ่านข้อมูลเพียงรอบเดียว โดยที่ discordancy ต่างจาก diameter ตรงที่การหาค่าความแตกต่างของ อ็อบเจกต์จะไม่หาจากทุกคู่อ็อบเจกต์โดยตรง แต่จะคำนวณจากความแตกต่างระหว่างอ็อบเจกต์ที่กำลังอ่านกับตัวแทนของอ็อบเจกต์อื่นๆ ตัวอย่างเช่น การหาค่าความแตกต่างของอ็อบเจกต์ *d* กับ อ็อบเจกต์ *a*,*b*, และ *c* (ซึ่งมี *o* เป็นตัวแทนของอ็อบเจกต์ *a*, *b*, และ *c*) สามารถคำนวณได้จาก *d*(*o*,*d*) แทน *d*(*a*,*d*), *d*(*b*,*d*) และ *d*(*c*,*d*) ให้ $\{\vec{x}_i\}$ เป็นเซ็ตข้อมูลขนาด *d* มิติจำนวน *N* อ็อบเจกต์ของคลัสเตอร์ใดๆ บน data space โดยที่ $i = 1, 2, \ldots, N$ และ n_i' คือจำนวนอ็อบเจกต์ทั้งหมดของคลัสเตอร์นั้นขณะอ่านค่า $\vec{x}_i$ และ $\vec{\mu}_{i-1}$ คือตัวแทนของอ็อบเจกต์ $\vec{x}_1, \vec{x}_2, ..., \vec{x}_{i-1}$ โดยค่า discordancy สามารถคำนวณได้สองแบบคือ weighted discordancy และ non-weighted discordancy

Weighted discordancy ($\overline{\Gamma}$) คือค่าความไม่ประสานกันภายในกลุ่มข้อมูลแบบให้น้ำหนัก สำหรับการคำนวณค่า $\overline{\Gamma}$ จะมีการให้น้ำหนักของความแตกต่างระหว่างอ็อบเจกต์ที่กำลังอ่านกับตัวแทนของกลุ่มอ็อบเจกต์ด้วยโดยการคูณระยะห่างระหว่างอ็อบเจกต์นั้นๆ กับจำนวนอ็อบเจกต์อ้างอิงของตัวแทน เช่น เมื่อ *i* มีค่าเท่ากับ 4 อัลกอริธึมจะทำการคำนวนระยะห่างระหว่างอ็อบเจกต์ $\vec{x}_4$ กับ $\vec{\mu}_3$ ซึ่งเป็นตัวแทนของอ็อบเจกต์ 3 ตัวคือ $\vec{x}_1, \vec{x}_2,$ และ $\vec{x}_3$ แทนการคำนวณระยะห่างระหว่างอ็อบเจกต์ $\vec{x}_4$ กับอ็อบเจกต์ $\vec{x}_1, \vec{x}_2,$ และ $\vec{x}_3$ โดยตรง จากนั้นจะทำการคำนวณความแตกต่างระหว่างอ็อบเจกต์โดยจะให้ค่าน้ำหนักของระยะห่างนั้นเท่ากับ 3 เป็นต้น โดยค่า $\overline{\Gamma}$ สามารถคำนวณได้จาก

$$\overline{\Gamma} = \frac{\sum_{i=1}^{N}\left[(n_i' - 1)\cdot d(\vec{\mu}_{i-1}, \vec{x}_i)\right]}{\sum_{i=1}^{N}(n_i' - 1)} \qquad (3\text{-}2)$$

(โดยให้ $\vec{\mu}_0 = \vec{x}_1$ และ $n_1' = 2$)

Non-weighted discordancy ($\tilde{\Gamma}$) คือค่าความไม่ประสานกันภายในกลุ่มข้อมูลแบบไม่ให้น้ำหนัก ซึ่งแสดงถึงค่าความแตกต่างเฉลี่ยระหว่างอ็อบเจกต์ที่กำลังอ่านกับตัวแทนของกลุ่มอ็อบเจกต์ โดยไม่มีการคิดค่าน้ำหนักตามจำนวนอ็อบเจกต์อ้างอิงของตัวแทนในขณะนั้นร่วมด้วย โดยค่า $\tilde{\Gamma}$ สามารถคำนวณได้จาก

$$\tilde{\Gamma} = \frac{\sum_{i=1}^{N} d(\mu_{i-1}, x_i)}{(N-1)} \qquad (3\text{-}3)$$

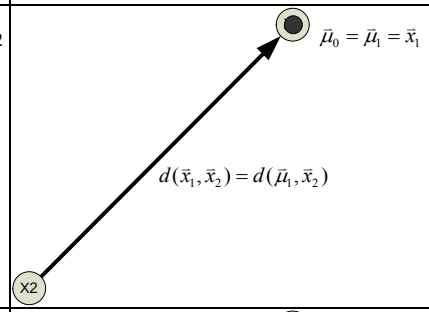
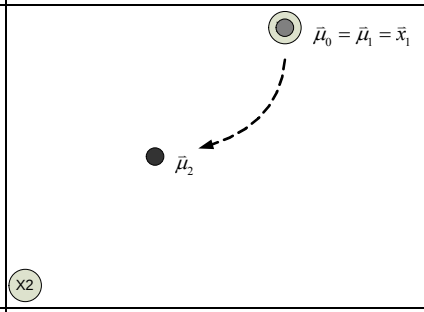
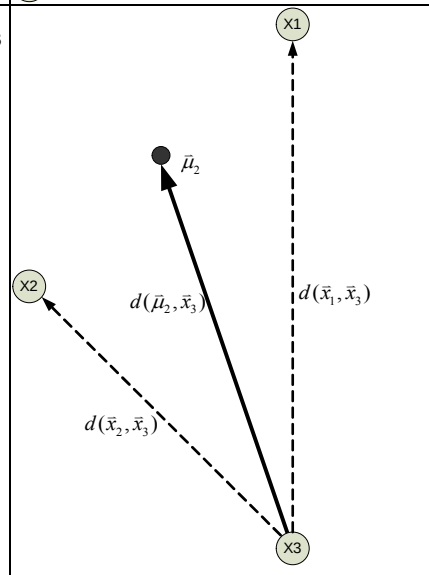
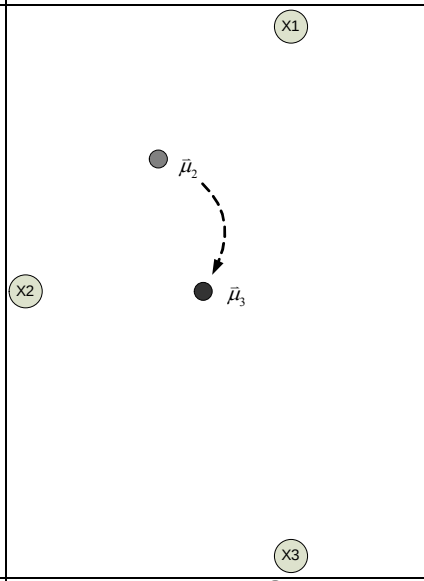
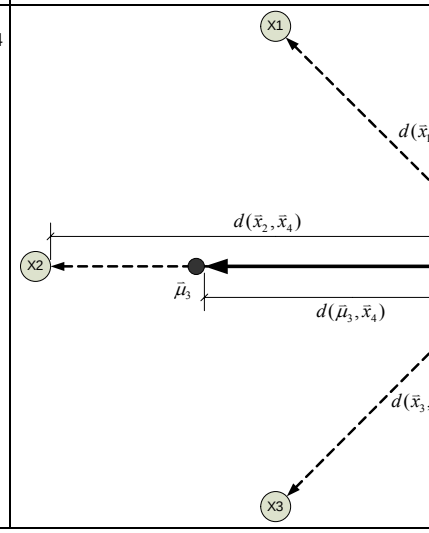
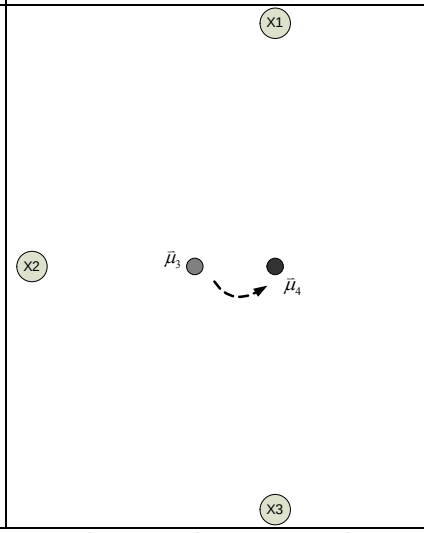
(โดยให้ $\vec{\mu}_0 = \vec{x}_1$ และ $\tilde{\Gamma}_1 = 0$ เมื่อ $N = 1$)

ที่มาและวิธีการคำนวณค่า weighted discordancy ($\bar{\Gamma}$), non-weighted discordancy ($\tilde{\Gamma}$) และ centroid ($\vec{\mu}$) เมื่อมีการอ่านอ็อบเจกต์ $\vec{x}_1, \vec{x}_2, \vec{x}_3$ และ $\vec{x}_4$ ตามลำดับ สามารถพิจารณาได้ดังตารางที่ 3.1 และเพื่อความเข้าใจมากยิ่งขึ้น สามารถพิจารณาตารางที่ 3.2 ซึ่งแสดงการคำนวณระยะทางและการเปลี่ยนตำแหน่งของตัวแทนของอ็อบเจกต์เมื่อมีการอ่านอ็อบเจกต์ $\vec{x}_1, \vec{x}_2, \vec{x}_3$ และ $\vec{x}_4$ ตามลำดับ โดยให้อ็อบเจกต์ตัวแรกแทนตัวแทนของอ็อบเจกต์เริ่มต้น ($\vec{\mu}_0 = \vec{x}_1$)

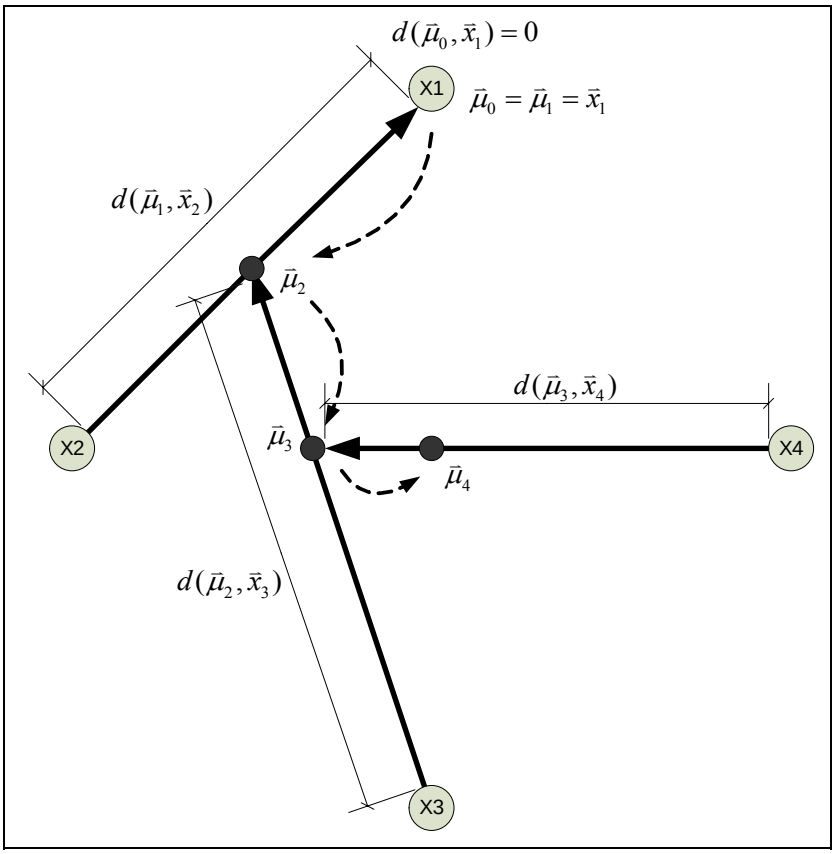
ตารางที่ 3.1 แสดงวิธีการคำนวณค่า $\bar{\Gamma}, \tilde{\Gamma}$ และ $\vec{\mu}$ ของอ็อบเจกต์ $\vec{x}_i$

i	$\bar{\Gamma}_i$	$\tilde{\Gamma}_i$	$\vec{\mu}_i$
1	$0 \cdot d(\vec{\mu}_0, \vec{x}_1)$	$d(\vec{\mu}_0, \vec{x}_1)$	$\frac{\vec{x}_1}{1}$
2	$\frac{0 \cdot d(\vec{\mu}_0, \vec{x}_1) + 1 \cdot d(\vec{\mu}_1, \vec{x}_2)}{1}$	$\frac{d(\vec{\mu}_0, \vec{x}_1) + d(\vec{\mu}_1, \vec{x}_2)}{1}$	$\frac{\vec{x}_1 + \vec{x}_2}{2}$
3	$\frac{\left\{\begin{matrix} 0 \cdot d(\vec{\mu}_0, \vec{x}_1) + 1 \cdot d(\vec{\mu}_1, \vec{x}_2) \\ +2 \cdot d(\vec{\mu}_2, \vec{x}_3) \end{matrix}\right\}}{1+2}$	$\frac{\left\{\begin{matrix} d(\vec{\mu}_0, \vec{x}_1) + d(\vec{\mu}_1, \vec{x}_2) \\ +d(\vec{\mu}_2, \vec{x}_3) \end{matrix}\right\}}{2}$	$\frac{\left\{\begin{matrix} \vec{x}_1 + \vec{x}_2 \\ +\vec{x}_3 \end{matrix}\right\}}{3}$
4	$\frac{\left\{\begin{matrix} 0 \cdot d(\vec{\mu}_0, \vec{x}_1) + 1 \cdot d(\vec{\mu}_1, \vec{x}_2) \\ +2 \cdot d(\vec{\mu}_2, \vec{x}_3) + 3 \cdot d(\vec{\mu}_3, \vec{x}_4) \end{matrix}\right\}}{1+2+3}$	$\frac{\left\{\begin{matrix} d(\vec{\mu}_0, \vec{x}_1) + d(\vec{\mu}_1, \vec{x}_2) \\ +d(\vec{\mu}_2, \vec{x}_3) + d(\vec{\mu}_3, \vec{x}_4) \end{matrix}\right\}}{3}$	$\frac{\left\{\begin{matrix} \vec{x}_1 + \vec{x}_2 \\ +\vec{x}_3 + \vec{x}_4 \end{matrix}\right\}}{4}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
N	$\bar{\Gamma}_N = \frac{\sum_{i=1}^{N} \left[(n_i' - 1) \cdot d(\vec{\mu}_{i-1}, \vec{x}_i) \right]}{\sum_{i=1}^{N} (n_i' - 1)}$	$\tilde{\Gamma}_N = \frac{\sum_{i=1}^{N} d(\mu_{i-1}, x_i)}{(N-1)}$	$\vec{\mu}_N = \frac{\sum_{i=1}^{N} \vec{x}_i}{N}$

ตารางที่ 3.2 แสดงการคำนวณระยะทางและการเปลี่ยนตำแหน่งของตัวแทนอ็อบเจกต์

$\vec{x}_i$	การคำนวณระยะทาง	การเปลี่ยนตำแหน่งของตัวแทนอ็อบเจกต์
$\vec{x}_1$		
$\vec{x}_2$		
$\vec{x}_3$		
$\vec{x}_4$		

<u>หมายเหตุ</u> เส้นประแสดงระยะทางระหว่างทุกคู่อ็อบเจกต์ที่ถูกแทนที่ด้วยเส้นทึบซึ่งแสดงระยะทางระหว่างอ็อบเจกต์กับตัวแทนกลุ่มอ็อบเจกต์



รูปที่ 3.2 สรุปวิธีการคำนวณระยะทางและการเปลี่ยนตำแหน่งของตัวแทนอ็อบเจกต์

เนื่องจากค่า discordancy (Γ) เป็นค่าที่บอกถึงความไม่ประสานกันภายในกลุ่มข้อมูล ดังนั้นค่า discordancy จึงสามารถมีส่วนช่วยในการค้นหาว่าข้อมูลในกลุ่มใดมีความน่าจะเป็นที่จะเป็นเอ้าท์ไลเออร์ได้อีกวิธีหนึ่ง โดยกลุ่มข้อมูลที่มีความไม่ประสานกันภายในกลุ่มสูง จะมีความน่าจะเป็นที่จะเป็นเอ้าท์ไลเออร์สูงกว่าข้อมูลในกลุ่มที่มีค่าความไม่ประสานกันภายในกลุ่มต่ำกว่า

3.2.2 DBSPACE: A Density-Biased Sampling using Partial Approximate Compactness Estimator

สำหรับเนื้อหาในส่วนนี้จะเป็นการนำเสนอเทคนิคการสุ่มข้อมูลแบบเบี่ยงเบนตามความหนาแน่นและความเป็นปึกแผ่นของข้อมูล หรือ DBSPACE ซึ่งเป็นเทคนิคที่ใช้โครงสร้างหลักของอัลกอริธึม DBS คือการเพิ่มอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่เบาบางและลดอัตราการสุ่มในบริเวณที่มีข้อมูลรวมกันอยู่หนาแน่น โดยจะพิจารณาความเป็นปึกแผ่นของข้อมูลข้างเคียงประกอบการสุ่มข้อมูลในแต่ละครั้งด้วย ข้อมูลที่มีความเป็นปึกแผ่น (compactness) มากกว่าจะมีความน่าจะเป็นที่จะถูกเลือกสูงกว่าบริเวณที่ข้อมูลอยู่อย่างกระจัดกระจาย

สมมุติให้ข้อมูลขนาด N อ็อบเจกต์ $x_1, x_2, ...x_N$ ซึ่งถูกแบ่งออกเป็นคลัสเตอร์หรือกลุ่มย่อยๆ g กลุ่ม โดยแต่ละกลุ่มประกอบด้วยอ็อบเจกต์จำนวน $n_1, n_2, ..., n_g$ (จำนวนของอ็อบ-เจกต์แสดงถึงความหนาแน่นของข้อมูลภายในกลุ่ม) และความเป็นปึกแผ่นของข้อมูลภายในกลุ่ม (compactness) มีค่าเท่ากับ $\Theta_1, \Theta_2, ..., \Theta_g$ เมื่อต้องการสร้างชุดข้อมูลสุ่มขนาด M อ็อบเจกต์ โดยที่ความน่าจะเป็นที่อ็อบเจกต์ x_j ใดๆ ภายในกลุ่มที่ i จะถูกเลือกจะต้อง

(i) แปรผกผันกับค่าความหนาแน่นของข้อมูลภายในกลุ่ม

$$P(x_j) \propto \frac{1}{n_i}$$

(ii) แปรผันตรงกับค่าความเป็นปึกแผ่นของข้อมูลภายในกลุ่ม

$$P(x_j) \propto \Theta_g \text{ หรือ } P(x_j) \propto \frac{1}{\Gamma_i}$$

จาก (i) จะทำให้กลุ่มที่มีความหนาแน่นสูงกว่าถูกสุ่มเลือกในอัตราที่น้อยกว่ากลุ่มที่มีความหนาแน่นต่ำกว่าเพื่อให้คลัสเตอร์ขนาดเล็กมีโอกาสปรากฏในชุดข้อมูลสุ่มมากขึ้น และในขณะเดียวกันจาก (ii) จะเป็นการรับประกันว่าข้อมูลในกลุ่มที่มีความหนาแน่นต่ำนั้นจะมีความน่าจะเป็นที่จะไม่เป็นเอ้าท์ไลเออร์ เนื่องจากกลุ่มข้อมูลนั้นจะต้องมีความเป็นปึกแผ่นของข้อมูลภายในกลุ่มด้วย อีกทั้งยังเป็นการลดความน่าจะเป็นที่อ็อบเจกต์ที่มีความน่าจะเป็นที่จะเป็นเอ้าท์ไล-เออร์จะถูกเลือกเข้ามาในชุดข้อมูลสุ่มได้อีกประการหนึ่ง ดังนั้นเพื่อให้สอดคล้องกับ (i) และ (ii) จึงให้ความน่าจะเป็นที่อ็อบเจกต์ x_j ใดๆ ภายในกลุ่มที่ i จะถูกเลือกมีค่าเท่ากับ

$$P(x_j) = \frac{\alpha}{n_i^e \cdot \Gamma_i^\delta} \tag{3-4}$$

โดยให้ α, e, และ δ เป็นค่าคงที่ใดๆ เมื่อกำหนด $e = 0$ จะทำให้ค่าความน่าจะเป็นในการสุ่มไม่ขึ้นกับจำนวนข้อมูลภายในกลุ่ม และเมื่อกำหนด $\delta = 0$ จะทำให้ค่าความน่าจะเป็นในการสุ่มไม่ขึ้นกับความเป็นปึกแผ่นของข้อมูลภายในกลุ่ม ซึ่งจะทำให้ได้ชุดข้อมูลสุ่มเช่นเดียวกับการใช้เทคนิค DBS และถ้ากำหนดให้ $e = 0$ และ $\delta = 0$ จะทำให้ทุกกลุ่มข้อมูลถูกสุ่มในอัตรา α เท่ากันทั้งหมด (uniform sampling) ซึ่งค่า α ที่เหมาะสมสำหรับการสุ่มที่ต้องการชุดข้อมูลสุ่มขนาด M อ็อบเจกต์สามารถพิจารณาได้ดังนี้

$$M = \sum_{j=1}^{N} P(x_j)$$

$$= \sum_{i=1}^{g} n_i \cdot P(x_j)$$

$$= \sum_{i=1}^{g} \left[n_i \cdot \frac{\alpha}{n_i^e \cdot \Gamma_i^{\delta}} \right]$$

$$= \alpha \cdot \sum_{i=1}^{g} \left[n_i^{1-e} \cdot \Gamma_i^{-\delta} \right]$$

จะได้ว่า

$$\alpha = \frac{M}{\sum_{i=1}^{g} \left[n_i^{1-e} \cdot \Gamma_i^{-\delta} \right]} \tag{3-5}$$

และเมื่อแทนค่า α ในสมการที่ 3-4 จะได้ความน่าจะเป็นที่อ็อบเจกต์ x_j ใดๆ ภายในกลุ่มที่ i จะถูกเลือกสำหรับชุดข้อมูลสุ่มขนาด M มีค่าเท่ากับ

$$P(x_j) = \frac{M}{n_i^e \cdot \Gamma_i^{\delta} \cdot \sum_{i=1}^{g} \left[n_i^{1-e} \cdot \Gamma_i^{-\delta} \right]} \tag{3-6}$$

จากสมการข้างต้นจะเห็นได้ว่าในบางกรณีส่วนหารสามารถมีค่าเป็นศูนย์ได้ เช่น n_i มีค่าเท่ากับ 1 ซึ่งทำให้ค่า Γ_1 มีค่าเป็นศูนย์ ดังนั้นเพื่อเป็นการเลี่ยงปัญหาการหารด้วยค่าศูนย์ (devide by zero) จึงปรับปรุงฟังก์ชัน $d(\vec{x}_i, \vec{x}_j)$ ที่ใช้สำหรับอัลกอริธึม DBSPACE โดยการบวก 1 เพิ่มเข้าไปกับค่าของระยะห่างที่ได้จากการคำนวณปกติ ซึ่งจะมีผลทำให้ค่า Γ ที่ได้มีค่าเพิ่มขึ้นจากเดิมประมาณ 1 มีผลทำให้ Γ^{δ} เป็นฟังก์ชันเพิ่มเสมอ

$$d(\vec{x}_i, \vec{x}_j) = 1 + \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \cdots + \left(x_{id} - x_{jd}\right)^2}$$

เนื่องจากข้อมูลถูกสุ่มด้วยค่าความน่าจะเป็นต่างกัน นั่นหมายความว่าข้อมูลแต่ละอ็อบเจกต์ที่ถูกเลือกมีความสำคัญหรือน้ำหนักไม่เท่ากัน จึงเป็นการไม่ถูกต้องหากให้ทุกอ็อบเจกต์ที่ถูกเลือกขึ้นมามีน้ำหนักเท่ากัน โดยค่าน้ำหนักนั้นจะแสดงถึงสัดส่วนที่อ็อบเจกต์นั้นสามารถเป็นตัวแทนของอ็อบเจกต์อื่นๆ นั่นคือ ถ้าให้กลุ่มย่อยที่ i ประกอบด้วยอ็อบเจกต์ $\left\{\vec{x}_1, \vec{x}_1, ..., \vec{x}_{n_i}\right\}$ $\vec{x}_j$ จะ

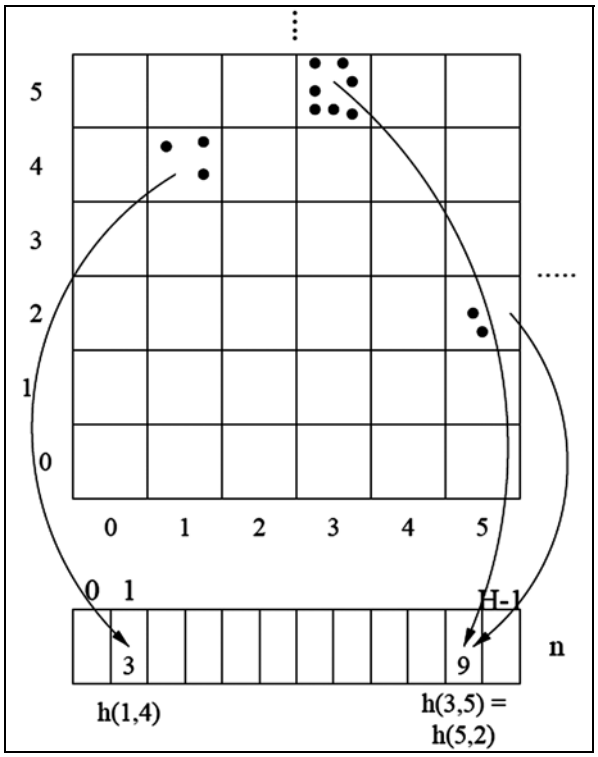
ถูกเลือกเป็นข้อมูลสุ่ม (ด้วยน้ำหนัก w_j) ด้วยความน่าจะเป็นที่จะถูกเลือก $P(\vec{x}_j)$

$$\sum_{j=1}^{n_i} P(\vec{x}_j) \cdot w_j = \sum_{j=1}^{n_i} P(\vec{x}_j) \cdot \frac{1}{P(\vec{x}_j)} = n_i$$

สำหรับการสุ่มข้อมูลอย่างง่ายทุกอ็อบเจกต์จะถูกสุ่มด้วยความน่าจะเป็นเท่ากัน $P(\vec{x}_j) = M / N$ ด้วยน้ำหนักเท่ากันทั้งหมดคือ $1/P(\vec{x}_j)$ หรือ N/M

สำหรับเทคนิคการสุ่มแบบ DBSPACE นั้นหัวใจหลักอยู่ที่การเบี่ยงเบนค่าความน่าจะเป็นในการสุ่มอ็อบเจกต์ใดๆ ซึ่งถูกแบ่งออกเป็นกลุ่มหรือคลัสเตอร์ย่อยๆ โดยค่าความน่าจะเป็นที่ได้จะขึ้นอยู่กับตัวแปรหรือคุณสมบัติจำเพาะของแต่ละคลัสเตอร์ย่อยๆ ดังที่ได้กล่าวไปแล้วข้างต้น ซึ่งคุณภาพของการแบ่งข้อมูลออกเป็นคลัสเตอร์ย่อยๆ นั้นจะมีผลโดยตรงต่อการคำนวณค่าคุณสมบัติจำเพาะของแต่ละคลัสเตอร์ย่อยๆ นั้นด้วย เทคนิคอย่างง่ายที่ใช้ในการแบ่งกลุ่มข้อมูล (data partitioning) โดยที่ไม่ทราบลักษณะการกระจายของข้อมูลมาก่อนทำได้โดยการแบ่งข้อมูลที่เป็นตัวเลขออกเป็น G ช่วง(บิน: bin) และสำหรับข้อมูลที่เป็นเชิงอักขระจะถูกแบ่งออกเป็นแต่ละบิน เมื่อข้อมูลเป็นข้อมูลเชิงตัวเลขขนาด d มิติ ข้อมูลจะถูกจัดให้อยู่ในกริด (grid) ขนาด d มิติโดยแต่ละมิติแบ่งออกเป็น B บิน โดยโครงสร้างกริดนี้จะประกอบด้วย *cell* จำนวน $d \times B$ *cell* ซึ่งต้องใช้หน่วยความจำขนาด $O(d \times B)$ ไบต์ แต่เนื่องจากบางบินอาจเป็นบินที่ว่างๆ เพราะไม่มีข้อมูลอยู่ในช่วงนั้น ดังนั้นการใช้เทคนิคนี้ในการแบ่งกลุ่มข้อมูลอาจเป็นการสิ้นเปลืองหน่วยความจำที่ต้องจองไว้สำหรับทุกบินได้

สำหรับ DBSPACE จะใช้เทคนิคการแบ่งกลุ่มข้อมูลเหมือนกันกับเทคนิค DBS ซึ่งเป็นเทคนิคแบบประมาณโดยการใช้ฟังก์ชัน hash เทคนิคนี้จะทำการจองหน่วยความจำสำหรับแถวลำดับของกลุ่มข้อมูลโดยใช้ชื่อว่า *hash_tab* ซึ่งประกอบด้วย H ลำดับ (เริ่มต้นจากลำดับที่ 0 จนถึง H-1) สามารถพิจารณาจากรูปที่ 3.3 แสดงตัวอย่างของข้อมูลซึ่งถูกแบ่งออกเป็น *cell* ที่มีขนาดเท่าๆ กันบนโครงสร้างแบบกริดโดยแต่ละ *cell* แทนกลุ่มข้อมูลหรือคลัสเตอร์ย่อย (partial cluster) และการจัดข้อมูลที่อยู่ในแต่ละบินลงไปในแถวลำดับที่สร้างขึ้นจะใช้ฟังก์ชัน hash ($hash(\vec{x}_j)$ ดังรูปที่ 3.4) ซึ่ง $hash(\vec{x}_j)$ อาจพิจารณาคล้ายกับการเชื่อมโยงค่าของอ็อบเจกต์ $\vec{x}_j$ ใดๆ ไปยังหมายเลขลำดับบน *hash_tab*



รูปที่ 3.3 การแบ่งกลุ่มข้อมูลแบบประมาณด้วยฟังก์ชัน hash (Palmer and Faloutsos, 2000)

hash($< v_1, ..., v_d >$) =
  FOR i = 1 TO d DO $h = h$ * 65599+ v_i
  RETURN h MOD H

รูปที่ 3.4 ฟังก์ชัน hash (Aho, Sethi and Ulman, quoted in Palmer and Faloutsos, 2000)

ข้อมูลในแถวลำดับแต่ละอันดับจะเก็บข้อมูลที่ใช้ประกอบการคำนวณค่าความน่าจะเป็นดังต่อไปนี้

1) $n[i]$ ซึ่งแสดงถึงจำนวนข้อมูลภายในกลุ่มที่ i ดังที่เคยกล่าวไปแล้ว (ใน DBS จะเก็บเฉพาะ n_i เท่านั้น)
2) $ls\vec{x}[i]$ ซึ่งเป็นผลรวมเชิงเวกเตอร์ของทุกอ็อบเจกต์ภายในกลุ่มที่ i ($ls\vec{x} = \sum_{j=1}^{n_i} \vec{x}_j$)
3) $lsd[i]$ คือผลรวมของระยะห่างระหว่างอ็อบเจกต์ภายในกลุ่มที่ i ซึ่งเป็นตัวตั้งของค่า discordancy (Γ) จากสมการที่ 3-2 และ 3-3

a. $lsd = \sum_{j=1}^{n_i} \left[(n'_j - 1) \cdot d(\bar{\mu}_{j-1}, \bar{x}_j) \right]$ เมื่อ $\Gamma = \overline{\Gamma}$, และ

b. $lsd = \sum_{j=1}^{n_i} d(\bar{\mu}_{j-1}, \bar{x}_j)$ เมื่อ $\Gamma = \tilde{\Gamma}$

อัลกอริธึม DBSPACE จะทำการแบ่งกลุ่มข้อมูลและสุ่มเลือกข้อมูลไปพร้อมๆ กันแบบคู่ขนานภายในการอ่านข้อมูลเพียงหนึ่งรอบ และข้อมูลที่ถูกเลือกจะถูกเก็บไว้ในบัฟเฟอร์ (output buffer) ที่ถูกจองไว้บนหน่วยความจำหลัก โดยในบัฟเฟอร์ประกอบด้วย $\{< P_i, \bar{x}_i >\}$ ซึ่งแสดงถึงการที่ $\bar{x}_i$ ถูกเลือกด้วยค่าความน่าจะเป็นเท่ากับ P_i และในเวลาต่อมาเมื่อมีการรับข้อมูลเข้ามาเรื่อยๆ $\bar{x}_i$ จะมีค่าความน่าจะเป็นที่จะถูกเลือกเท่ากับ P'_i ดังนั้นจึงต้องมีการปรับปรุงค่าในบัฟเฟอร์อยู่เสมอเพื่อให้ชุดข้อมูลสุ่มที่ได้เป็นปัจจุบัน โดยการเก็บ $< P'_i, \bar{x}_i >$ แทน $< P_i, \bar{x}_i >$ ด้วยความน่าจะเป็น P'_i / P_i (หรือทำการลบอ็อบเจกต์นั้นออกจากบัฟเฟอร์) ซึ่งแสดงถึงการเก็บ $\bar{x}_i$ ไว้ในบัฟเฟอร์ด้วยค่าความน่าจะเป็นเท่ากับ P'_i ด้วยน้ำหนักเท่ากับ $1/P'_i$ รายละเอียดของอัลกอริธึม DBSPACE แสดงดังรูปที่ 3.5

ฟังก์ชัน reduce() เป็นฟังก์ชันที่ใช้ในการลดจำนวนอ็อบเจกต์ในบัฟเฟอร์ลงเพื่อเปิดช่องว่างให้อ็อบเจกต์ที่มีความน่าจะเป็นที่จะเป็นตัวเลือกที่ดีกว่าเข้ามาแทนที่ อีกทั้งยังเป็นการปรับปรุงข้อมูลในบัฟเฟอร์ให้เป็นปัจจุบันด้วย โดยฟังก์ชัน reduce() จะทำการลดจำนวนอ็อบเจกต์ในบัฟเฟอร์อย่างน้อย 1 อ็อบเจกต์เสมอ และจากรูปที่ 3.5 ค่า α_D คือค่าของส่วนหารของ α (จากสมการที่ 3-5) โดยที่บรรทัด (*) เป็นการลบค่าเก่าของส่วนหารนั้นออกก่อนที่จะทำการบวกค่าส่วนหารใหม่ที่มีการปรับปรุงค่าให้เป็นปัจจุบันแล้วในบรรทัด (**) ส่วนฟังก์ชัน discordancy() เป็นฟังก์ชันที่ใช้ในการคำนวณค่า discordancy ซึ่งรองรับการคำนวณทั้งค่า weighted discordancy และ non-weighted discordancy

จากอัลกอริธึมดังรูปที่ 3.5 อัลกอริธึม DBSPACE จะทำการอ่านข้อมูลทั้งหมด N รอบ และในแต่ละรอบฟังก์ชัน reduce() จะถูกเรียกใช้งานด้วยความน่าจะเป็น $P = \min\left\{ M / \left[\alpha_D \cdot n[i]^e \cdot \Gamma^\delta \right], 1 \right\}$ เมื่อหน่วยความจำไม่เพียงพอ โดยจะสแกนข้อมูลจำนวน M อ็อบเจกต์ที่ถูกเก็บไว้ในบัฟเฟอร์เพื่อคำนวณความน่าจะเป็นใหม่อีกครั้งและลดจำนวนอ็อบเจกต์ในบัฟเฟอร์อย่างน้อยสุด 1 อ็อบเจกต์ โดยความซับซ้อน (time complexity) ของอัลกอริธึม DBSPACE มีค่าเท่ากับ $O(NM)$ ซึ่งในความเป็นจริงฟังก์ชัน reduce() จะถูกเรียกใช้งานจำนวนครั้งน้อยมากเมื่อเทียบกับจำนวนข้อมูลทั้งหมด (N) และเมื่อ M มีค่าน้อยกว่า N มากๆ ($M \ll N$) ความซับซ้อนของอัลกอริธึมสามารถลดลงได้เป็น $O(N)$

$\alpha_D = 0$
FOR each input point *x* DO
    i = hash($\vec{x}$)
    IF $n[i]$ <> 0 THEN $\alpha_D = \alpha_D - \left(n[i]^{1-e} \cdot discordancy(lsd[i], n[i])^{-\delta}\right)$ (*)
    $n[i] = n[i] + 1$
    IF $n[i] = 1$ THEN $\vec{\mu} = \vec{x}$
    IF $\Gamma = \overline{\Gamma}$ THEN $lsd[i] = lsd[i] + (n[i] - 1) \cdot d(\vec{\mu}, \vec{x})$
    ELSE $lsd[i] = lsd[i] + d(\vec{\mu}, \vec{x})$
    $ls\vec{x} = ls\vec{x} + \vec{x}$
    $\vec{\mu} = ls\vec{x} / n[i]$
    $\Gamma = discordancy(lsd[i], n[i])$
    $\alpha_D = \alpha_D + \left(n[i]^{1-e} \cdot \Gamma^{-\delta}\right)$ (**)
    WITH prob. $P = \min\left\{M \Big/ \left[\alpha_D \cdot n[i]^e \cdot \Gamma^{\delta}\right], 1\right\}$ DO
        IF the output buffer is full THEN *reduce*()
        // start drawing when group contains more than 1 object
        IF $n[i] > 1$ THEN add $< P, \vec{x} >$ to the output buffer
reduce()
FOR each output buffer entry $< P_i, \vec{x}_i >$ DO output $< 1/P_i, \vec{x}_i >$

reduce() IS
    FOR each output buffer entry $< P_i, \vec{x}_i >$ DO
    Let $P_i' = \min\left\{M \Big/ \left[\alpha_D \cdot n[i]^e \cdot \Gamma^{\delta}\right], 1\right\}$
    WITH prob. P_i' / P_i replace this entry with $< P_i', \vec{x}_i >$
    OTHERWISE remove this entry

discordancy(*lsd*, *n*) IS
    IF *n*<= 1 THEN *n*=2
    IF $\Gamma = \overline{\Gamma}$ THEN
        disc = *lsd*/((*n**(*n*-1))/2)
    ELSE
        disc = *lsd*/(*n*-1)
    RETURN *disc*

รูปที่ 3.5 อัลกอริธึม DBSPACE

อย่างไรก็ตาม เนื่องจากการคำนวณค่า Γ เป็นการหาค่าโดยประมาณจากตัวแทนในขณะนั้น (แทนข้อมูลทุกตัวที่ผ่านมา) ซึ่งตำแหน่งของตัวแทนจะปรับเปลี่ยนไปตามข้อมูลที่รับเข้ามาใหม่อยู่ตลอดเวลา ดังนั้นค่า Γ ที่ได้ก็จะเปลี่ยนไปตามลักษณะการเรียงลำดับของข้อมูลด้วยเช่น

กัน และอาจมีผลต่อการเลือกหรือไม่เลือกข้อมูลใดๆ เข้ามาในชุดข้อมูลสุ่มด้วย แต่เนื่องจากอัลกอริธึม DBSPACE มีกลไกสำหรับการปรับปรุงบัฟเฟอร์ของชุดข้อมูลสุ่มอยู่ตลอดเวลา โดยการคัดเอาข้อมูลที่เคยถูกเลือกซึ่งในเวลาต่อมามีความน่าจะเป็นที่จะถูกเลือกต่ำออกไปจากชุดข้อมูลผล- ลัพธ์เพื่อเปิดช่องว่างสำหรับข้อมูลที่ดีกว่า ซึ่งคล้ายกับกลไกในการเรียนรู้และปรับตัวของมนุษย์ซึ่งจะใช้ฐานความรู้ขณะปัจจุบันประกอบการตัดสินใจดีที่สุด ณ ขณะนั้น โดยฐานความรู้จะมีการปรับตัวให้เกิดความสมดุลตามข้อมูลที่ได้รับ เช่นเดียวกับ DBSPACE ซึ่งใช้ฐานความรู้ขณะปัจจุบันประกอบการสร้างชุดข้อมูลผลลัพธ์ที่เหมาะสมที่สุด ณ ขณะปัจจุบัน

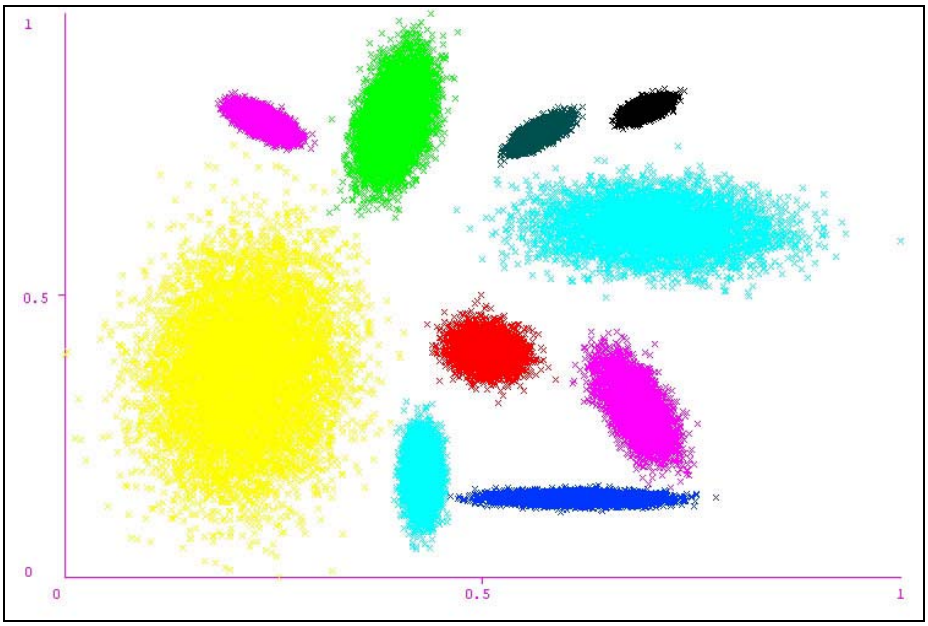
3.3 ข้อมูลที่ใช้ในการวิจัย

สำหรับข้อมูลที่ใช้ในงานวิจัยนี้จะใช้ชุดข้อมูลที่ได้จากการสังเคราะห์ ซึ่งมีลักษณะการกระจายของข้อมูลเป็นแบบ Gaussian distribution โดยที่แต่ละคลัสเตอร์มีความแตกต่างกันทั้งขนาดและความหนาแน่น อัลกอริธึมที่ใช้สำหรับการสร้างชุดข้อมูลสังเคราะห์คือ Gaussian Cluster Generator พัฒนาขึ้นที่ University of Manchester โดย Handl and Knowles (n.d.) ซึ่งจะสร้างชุดข้อมูลที่แต่ละเรคคอร์ดระบุหมายเลขคลัสเตอร์ที่ถูกต้องกำกับไว้แล้ว พารามิเตอร์ของอัลกอริธึมแสดงไว้ดังตารางที่ 3.3

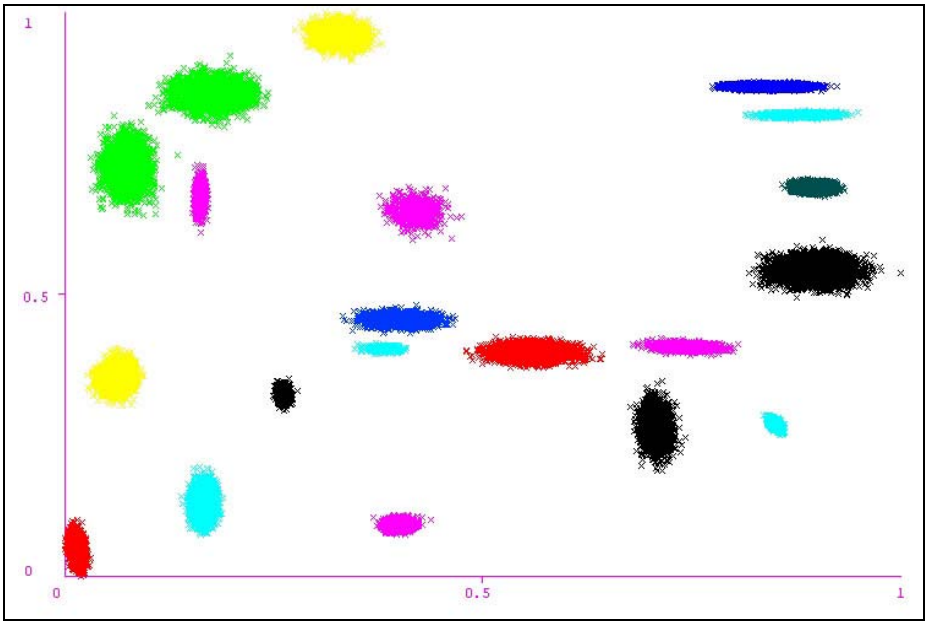
ตารางที่ 3.3 แสดงพารามิเตอร์สำหรับอัลกอริธึมสังเคราะห์ข้อมูล

พารามิเตอร์	ค่า	รายละเอียด
dim	number	จำนวนมิติหรือแอททริบิวต์ของข้อมูล
k	number	จำนวนคลัสเตอร์ที่ต้องการสร้าง
MINMU	number	ค่าเฉลี่ยต่ำสุดของแต่ละแอททริบิวต์
MAXMU	number	ค่าเฉลี่ยสูงสุดของแต่ละแอททริบิวต์
MINSIGMA	0	ความแปรปรวน(σ) ต่ำสุดของแต่ละแอททริบิวต์
MAXSIGMA	$2 \cdot \sqrt{\dim}$	ความแปรปรวน(σ) สูงสุดของแต่ละแอททริบิวต์
MINSIZE	number	ขนาดหรือจำนวนอ็อบเจกต์ต่ำสุดสำหรับคลัสเตอร์
MAXSIZE	number	ขนาดหรือจำนวนอ็อบเจกต์สูงสุดสำหรับคลัสเตอร์

ตัวอย่างชุดข้อมูลสังเคราะห์ที่ใช้ในการวิจัยนี้ได้แก่ชุดข้อมูล DS1 และ DS2 ซึ่งเป็นข้อมูลขนาด 2 มิติ จะแสดงไว้ดังรูปที่ 3.6 และ 3.7 ตามลำดับ



รูปที่ 3.6 ชุดข้อมูลสังเคราะห์ DS1 (dim=2, k=10, size= [5000, 10000], 72703 points)



รูปที่ 3.7 ชุดข้อมูลสังเคราะห์ DS2 (dim=2, k=20, size= [500, 5000], 58822 points)

3.4 การเปรียบเทียบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล

ในส่วนนี้จะกล่าวถึงการเปรียบเทียบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลในแบบต่างๆ โดยเริ่มจากการนำอัลกอริธึมสุ่มข้อมูลที่มีอยู่มาทดสอบเพื่อหาคุณสมบัติเด่นของแต่ละอัลกอริธึม เพื่อนำไปสู่การพัฒนาอัลกอริธึมสุ่มข้อมูลที่มีประสิทธิภาพสูงขึ้น จากนั้นจึงนำอัลกอริธึมสุ่มข้อมูลที่ผู้วิจัยได้เสนอขึ้นนั้นมาทดสอบเปรียบเทียบประสิทธิภาพกับอัลกอริธึมอื่นๆ ต่อไป โดยจะทำการทดสอบด้วยชุดข้อมูลสังเคราะห์จำนวนหนึ่ง ซึ่งมีการกระจายของข้อมูลแบบไม่ปกติ โดยเกณฑ์ในการพิจารณาความเหมาะสมของแต่ละเทคนิคประกอบด้วย เวลาที่ใช้ในการสุ่มข้อมูล (time to sample) จำนวนหน่วยความจำที่ใช้ (memory usage) เวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลจากชุดข้อมูลสุ่ม (time to cluster) ตลอดจนความแม่นตรงของคลัสเตอร์ที่พบ (*number of cluster found*: *NC*) หรือจำนวนคลัสเตอร์ที่ถูกต้องและสามารถยอมรับได้จากชุดข้อมูลสุ่ม อีกทั้งยังจะศึกษาถึงความทนทานต่อข้อมูลรบกวนของแต่ละเทคนิคโดยการทดสอบกับชุดข้อมูลที่ถูกสร้างข้อมูลรบกวนเข้าไปในปริมาณต่างๆ กัน โดยเครื่องมือที่ใช้ในการทดลองและรายละเอียดของขั้นตอนการทดลองมีดังต่อไปนี้

3.4.1 เครื่องมือที่เกี่ยวข้อง

WEKA เป็นระบบเปิดเผยซอร์สโค้ดที่ใช้สำหรับทำเหมืองข้อมูล ในการทดลองจะใช้ *k*-means ซึ่งเป็นอัลกอริธึมจัดกลุ่มข้อมูลที่เป็นที่นิยมในการค้นหาคลัสเตอร์บนชุดข้อมูลที่มีรูปร่างที่ไม่ซับซ้อน โดยอัลกอริธึมที่ใช้พัฒนาขึ้นบนระบบ WEKA และจะแสดงผลการค้นหาคลัสเตอร์เป็นเซนทรอยด์หรือตัวแทนของแต่ละคลัสเตอร์ และรายละเอียดอื่นๆ ที่เกี่ยวข้อง โดยระบบ WEKA จะรับข้อมูลที่อยู่ในรูปแบบ ARFF (Attribute-Relation File Format) ซึ่งประกอบด้วยข้อมูลสองส่วนคือส่วนอธิบายข้อมูลและส่วนที่เป็นข้อมูล ดังตัวอย่างในรูปที่ 3.8 (สำหรับการวิจัยนี้จะใช้ WEKA 3-4-7 ในการทดลอง)

```
@relation DS1
@attribute ATTR1 numeric
@attribute ATTR2 numeric
@attribute CLSID {0,1,2,3,4,5,6,7,8,9}
@data
0.205456,    0.286055,    6
0.209253,    0.313145,    5
0.274268,    0.567863,    9
```

รูปที่ 3.8 ตัวอย่างข้อมูลในรูปแบบ ARFF

NOISEGEN เป็นอัลกอริธึมที่ผู้วิจัยพัฒนาขึ้นเพื่อใช้สร้างข้อมูลรบกวนบนชุดข้อมูลตั้งต้น ซึ่งข้อมูลรบกวนในที่นี้จะหมายถึงข้อมูลใดๆ ซึ่งมีค่าไม่เท่ากับข้อมูลเดิม โดยอัลกอริธึมจะสุ่มเลือกเรคคอร์ดและแอททริบิวต์ด้วยฟังก์ชัน random จากนั้นจะทำการสุ่มค่าอื่นเพื่อใช้แทนที่ค่าเดิม พารามิเตอร์ที่สำคัญคือจำนวนข้อมูลรบกวนคิดเป็นร้อยละของจำนวนเรคคอร์ดทั้งหมด (%noise) และจำนวนแอททริบิวต์ที่มีความน่าจะเป็นที่จะเป็นข้อมูลรบกวนเมื่อเรคคอร์ดนั้นถูกเลือกเป็นข้อมูลรบกวน (np: [0, dim])

usage: inputFile.dat dim %noise np outputFile.arff

CLSCTR เป็นอัลกอริธึมที่ผู้วิจัยพัฒนาขึ้นเพื่อใช้หาจุดเซนทรอยด์จากชุดข้อมูลสังเคราะห์ซึ่งแต่ละเรคคอร์ดระบุหมายเลขคลัสเตอร์ไว้แล้ว โดยอัลกอริธึมจะสร้างผลลัพธ์ที่เป็นเซ็ตของจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ และจัดเก็บไว้ในไฟล์ที่มีนามสกุล *.nc

usage: inFile.dat dim k Ncfile.nc

Weka2*NC* เป็นอัลกอริธึมที่ผู้วิจัยพัฒนาขึ้นเพื่อใช้ดึงข้อมูลผลการค้นหาคลัสเตอร์ของระบบ WEKA โดยจะเลือกเฉพาะส่วนที่รายงานค่าของจุดเซนทรอยด์เท่านั้น และจัดเก็บไว้ในไฟล์ที่มีนามสกุล *.nc

usage: WekaFile.txt

***NC*metric** เป็นอัลกอริธึมที่ผู้วิจัยพัฒนาขึ้นเพื่อใช้เปรียบเทียบจุดเซนทรอยด์ของคลัสเตอร์ที่พบกับจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ และนับจำนวนคลัสเตอร์ที่ตรงกับคลัส-เตอร์ต้นแบบ ให้ $\{c_1, c_2, ..., c_k\}$ เป็นเซ็ตของจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ และ $\{\hat{c}_1, \hat{c}_2, ..., \hat{c}_{k'}\}$ เป็นเซ็ตของจุดเซนทรอยด์ของคลัสเตอร์ที่ได้จาก *k*-means จะถือว่า c_i ถูกพบก็ต่อเมื่อมีบาง $\hat{c}_j$ ที่ทำให้ $d(c_i, \hat{c}_j) < \xi$ โดยอัลกอริธึม *NC*metric จะทำการเปรียบเทียบจุดเซน-ทรอยด์จากไฟล์ที่มีนามสกุล *.nc และจะให้ผลลัพธ์เป็นค่า *NC* (*number of cluster found*) ซึ่งหมายถึงจำนวนของคลัสเตอร์ต้นแบบที่พบ (Palmer and Faloutsos, 2000)

usage: oNCfile.nc nNCfile.nc dim k xi

3.4.2 ขั้นตอนการทดลอง

สำหรับการทดลองในการวิจัยนี้จะแบ่งออกเป็นสองส่วนคือ การทดสอบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS เพื่อวิเคราะห์หาลักษณะเด่นของแต่ละอัลกอริธึม เพื่อใช้ประกอบการออกแบบอัลกอริธึมใหม่ และการทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลแบบใหม่ที่พัฒนาขึ้น (DBSPACE) และการเปรียบเทียบผลลัพธ์กับอัลกอริธึมอื่นๆ และในการทดลองแต่ละส่วนจะทำการศึกษาคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึมนั้นๆ ด้วย

การทดสอบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS มีขั้นตอนดังต่อไปนี้

1) แปลงข้อมูลต้นฉบับที่ได้จากอัลกอริธึมสร้างข้อมูลสังเคราะห์ (untitled.dat) ให้อยู่ในรูปแบบ ARFF (DS*_ori.arff) สำหรับใช้ในระบบ WEKA ซึ่งทำได้โดยการเพิ่มส่วนที่ใช้ในการอธิบายข้อมูลลงไปในส่วนบนสุดของข้อมูล จากนั้นจึงบันทึกไฟล์เป็น DS*_ori.arff

```
@relation DS1
@attribute ATTR1 numeric
@attribute ATTR2 numeric
@attribute CLSID {0,1,2,3,4,5,6,7,8,9}
@data
0.205456,    0.286055,    6
0.209253,    0.313145,    5
0.274268,    0.567863,    9
0.19724,     0.414573,    1
```

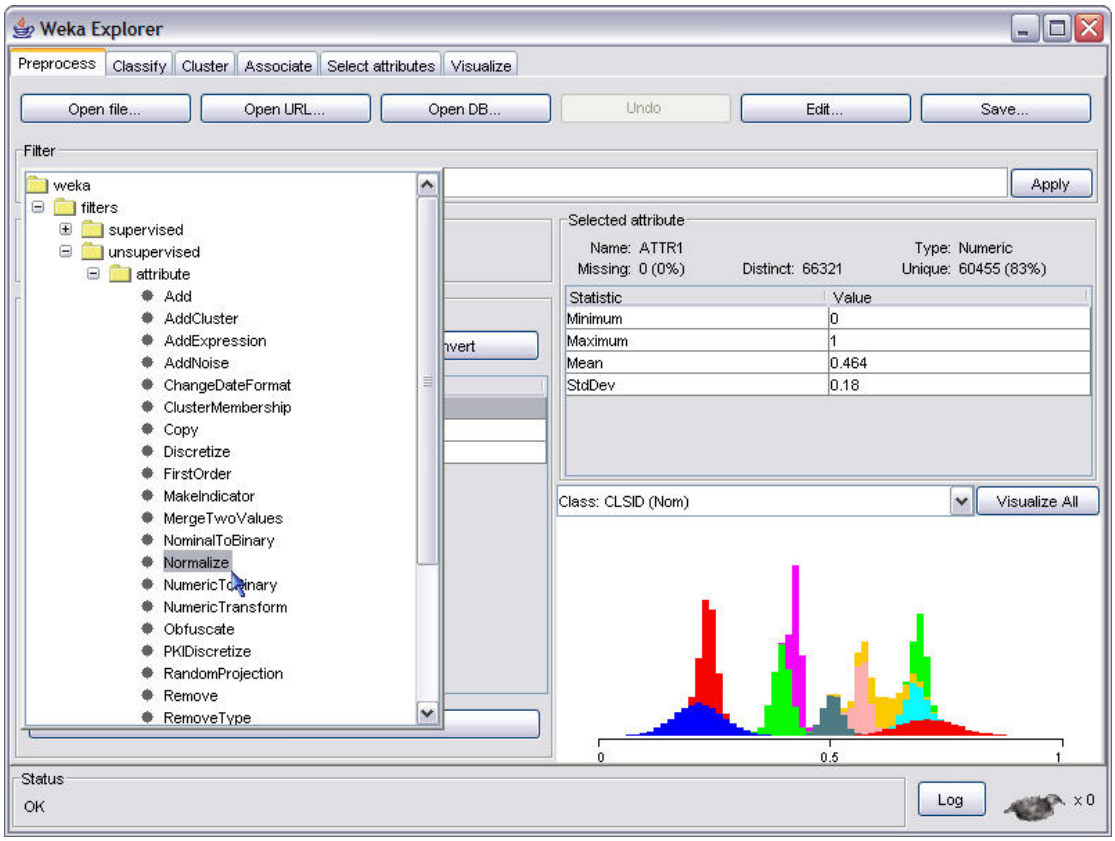
← ARFF Header

รูปที่ 3.9 แสดงการแปลงข้อมูลให้อยู่ในรูปแบบ ARFF

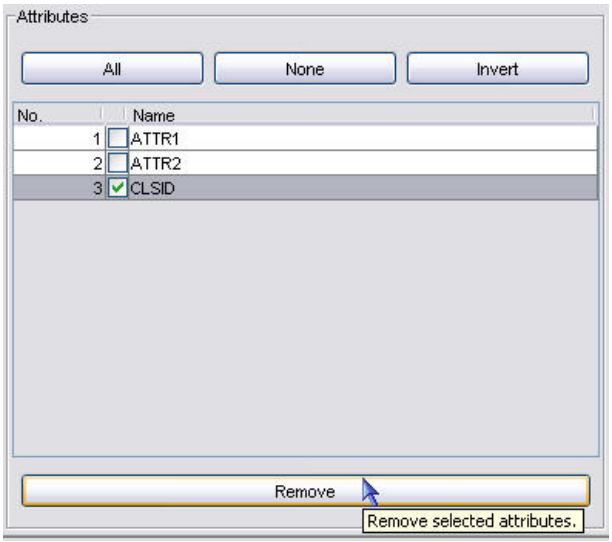
2) แปลงข้อมูลให้มีค่าที่เหมาะสมสำหรับอัลกอริธึมสุ่มข้อมูล โดยใช้โปรแกรม WEKA เปิดข้อมูลที่ได้จากข้อ 1 แล้วเลือก filter “Normalize” โดยคลิ๊กที่ปุ่ม Choose แล้วเลือก weka\filters\unsupervised\attribute\Normalize และกดปุ่ม Apply เพื่อทำการปรับค่าของข้อมูลให้อยู่ในช่วง [0.0,1.0] (ดูรูปที่ 3.10)

(1) กดปุ่ม Save เพื่อบันทึกปรับปรุงข้อมูลลงในไฟล์เดิม (DS*_ori.arff)

(2) ตัดแอททริบิวต์ที่ระบุหมายเลขคลัสเตอร์ทิ้งไป (CLSID) แล้วกดปุ่ม Save เพื่อบันทึกข้อมูลลงในไฟล์ใหม่ชื่อว่า DS*.arff (ดูรูปที่ 3.11)



รูปที่ 3.10 แสดงวิธีการ Normalize ข้อมูลด้วยโปรแกรม WEKA



รูปที่ 3.11 แสดงการตัดแอททริบิวต์ CLSID ที่ระบุหมายเลขคลัสเตอร์ด้วยโปรแกรม WEKA

3) แปลงข้อมูลกลับไปเป็นรูปแบบ DAT เหมือนเดิม เนื่องจากอัลกอริธึมสุ่มข้อมูลที่ใช้ยังไม่รองรับข้อมูลในรูปแบบ ARFF โดยการ

(1) ตัด arff header ในชุดข้อมูล DS*_ori.arff ทิ้งไปและแทนที่ comma(,) ที่ใช้ขั้นระหว่างแอททริบิวต์ด้วยแท็ป(\t) จากนั้นบันทึกข้อมูลเป็น DS*_ori.dat

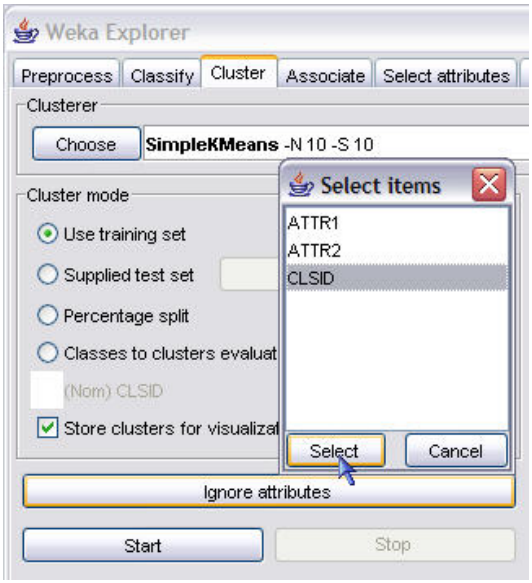
(2) ตัด arff header ในชุดข้อมูล DS*.arff ทิ้งไปและแทนที่ comma(,) ด้วยแท็ป(\t) จากนั้นบันทึกข้อมูลเป็น DS*.dat

4) ทำการสร้างชุดข้อมูลสุ่มในรูปแบบ ARFF ด้วยอัลกอริธึมสุ่มข้อมูลทั้งสามด้วยอัตราการสุ่มต่างๆ พร้อมทั้งบันทึกเวลาและหน่วยความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.4

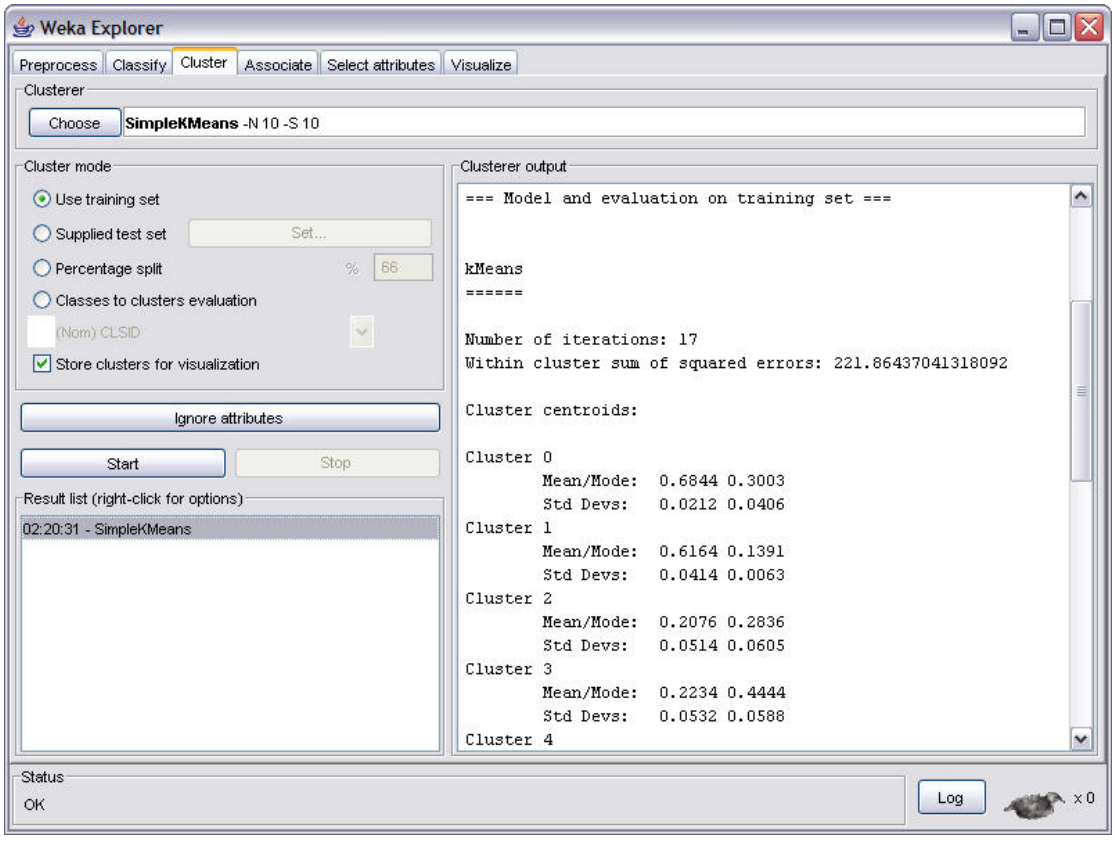
ตารางที่ 3.4 แสดงการสร้างชุดข้อมูลสุ่มด้วยอัลกอริธึมและอัตราการสุ่มขนาดต่างๆ

ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS1.dat DS2.dat	RVS DBS DBRVS	[1, 2, 3, …, 10] (5 *i*terations)	DS*_[sampler][%s][i].arff
รวม: 2	3	10 x 5	300 ชุดข้อมูล

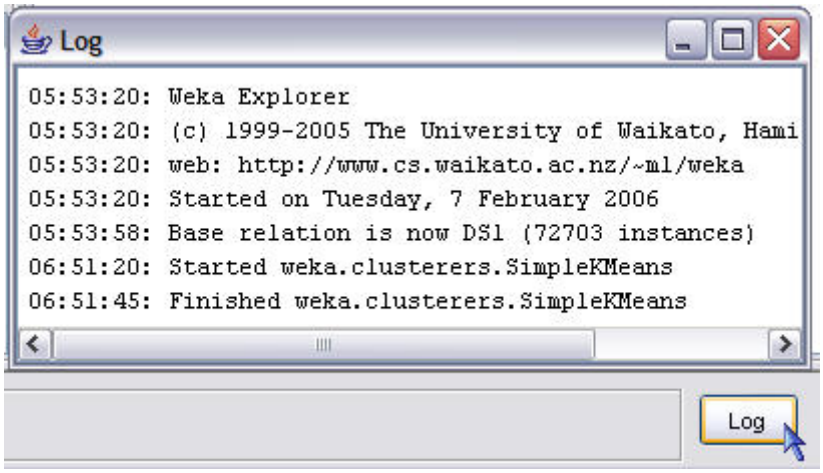
5) ทำการค้นหาคลัสเตอร์จากชุดข้อมูลตั้งต้นและชุดข้อมูลสุ่มที่สร้างขึ้นทั้งหมดจากข้อ 4 โดยใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และกำหนดค่า k (จำนวนคลัสเตอร์ที่ต้องการค้นหา) เท่ากับจำนวนคลัสเตอร์ที่มีอยู่จริงของชุดข้อมูลตั้งต้น และต้องไม่นำแอททริบิวต์ที่ระบุหมายเลขคลัสเตอร์มาเกี่ยวข้องในการค้นหาคลัสเตอร์ด้วย (ดังรูปที่ 3.12) จากนั้นจึงกดปุ่ม Start เพื่อเริ่มการค้นหาคลัสเตอร์และรอจนกระบวนการเสร็จสิ้น (ดูรูปที่ 3.13) แล้วบันทึกผลลัพธ์ (สำหรับชุดข้อมูล DS*_[sampler][%s][i].arff ให้บันทึกผลเป็นไฟล์ชื่อว่า DS*_[sampler][%s][i].txt) และบันทึกเวลาที่ใช้ในการค้นหาคลัสเตอร์โดยดูจาก log ของโปรแกรม WEKA (ดังรูปที่ 3.14)



รูปที่ 3.12 แสดงการละเว้นแอททริบิวต์ที่ระบุหมายเลขคลัสเตอร์ในระหว่างการค้นหาคลัสเตอร์



รูปที่ 3.13 แสดงผลการค้นหาคลัสเตอร์



รูปที่ 3.14 log ของโปรแกรม WEKA

6) จากผลลัพธ์ที่ได้จากการค้นหาคลัสเตอร์ด้วย SimpleKMeans ใน WEKA แต่ละคลัสเตอร์จะประกอบด้วยค่าเฉลี่ยของแต่ละแอททริบิวต์ของอ็อบเจกต์ภายในคลัสเตอร์นั้น ซึ่งแสดงถึงพิกัดของจุดเซนทรอยด์ที่ใช้เป็นตัวแทนของคลัสเตอร์ โดยในขั้นตอนนี้จะทำการดึงเฉพาะข้อมูลจุดเซนทรอยด์ซึ่งเป็นตัวแทนของแต่ละคลัสเตอร์มาสร้างเป็นไฟล์จุดเซนทรอยด์ (ตัวอย่างดังรูปที่ 3.15) เพื่อเป็นการเตรียมพร้อมสำหรับการเปรียบเทียบคลัสเตอร์ที่พบกับคลัสเตอร์ต้นแบบต่อไป

(1) ใช้ CLSCTR เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบจากไฟล์ DS*_ori.dat แล้วบันทึกผลที่ได้ลงบนไฟล์ชื่อ DS*_ori.nc

(2) ใช้ Weka2NC เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS*_[sampler][%s][i].txt) แล้วบันทึกผลที่ได้ลงบนไฟล์ชื่อ DS*_[sampler][%s][i].nc

0.6844 0.3003 0.6164 0.1391 0.2076 0.2536 0.2876 0.2846 0.3071 0.1836

รูปที่ 3.15 แสดงตัวอย่างข้อมูลจุดเซนทรอยด์ขนาดสองมิติ

7) เปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับเซนทรอยด์ของคลัสเตอร์ต้นแบบ โดยใช้ *NC*metric ซึ่งจะทำการเปรียบเทียบเซนทรอยด์จากไฟล์จุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ (DS*_ori.nc) กับไฟล์จุดเซนทรอยด์ของคลัสเตอร์ที่พบ DS*_[sampler][%s][i].nc และกำหนดค่า ξ เท่ากับ 0.01 แล้วบันทึกจำนวนจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบที่สามารถพบได้จากไฟล์ข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่ได้จาก WEKA (*NC*)

การทดสอบคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึม RVS, DBS, และ DBRVS

1) สร้างข้อมูลรบกวนลงบนชุดข้อมูลตั้งต้นด้วย NOISEGEN ในอัตรา 1%, 2%, 3%, …, 10%, 15%, 20%, 25%, และ 30% ตามลำดับ (สำหรับชุดข้อมูลสุ่ม DS*_ori.dat ให้บันทึกไฟล์ในชื่อ DS*_n[%2n].dat) ซึ่งสามารถสรุปได้ดังตารางที่ 3.5

ตารางที่ 3.5 แสดงการสร้างข้อมูลรบกวนลงบนชุดข้อมูลตั้งต้นด้วยขนาดต่างๆ

ชุดข้อมูลทดสอบ	ข้อมูลรบกวน (%)	ชุดข้อมูลรบกวนผลลัพธ์
DS*_ori.dat	[1,2,3,…,10,15,20,25,30]	DS*_n[%2n].dat
รวม: 1	14	14 ชุดข้อมูล

2) ทำการสุ่มข้อมูลด้วยอัลกอริธึม RVS, DBS, และ DBRVS ในอัตราการสุ่มเท่ากันที่สามารถให้ผลลัพธ์ของการค้นหาคลัสเตอร์ได้ดีที่สุด พร้อมทั้งบันทึกเวลาและหน่วย ความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.6

ตารางที่ 3.6 แสดงการสร้างชุดข้อมูลสุ่มจากชุดข้อมูลรบกวนด้วยขนาดต่างๆ

ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS*_n[%2n].dat	RVS DBS DBRVS	*c* (5 *i*terations)	DS*_n[%2n][sampler][%s][i].arff
รวม: 14	3	1 x 5	210 ชุดข้อมูล

3) ทำการค้นหาคลัสเตอร์จากชุดข้อมูลที่สร้างขึ้นทั้งหมดจากข้อ 2 โดยใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และกำหนดค่า k (จำนวนคลัสเตอร์ที่ต้องการค้นหา) เท่ากับจำนวนคลัสเตอร์ที่มีอยู่จริงของชุดข้อมูลตั้งต้น จากนั้นบันทึกผลลัพธ์ลงในไฟล์ *.txt และบันทึกเวลาที่ใช้ในการค้นหาคลัสเตอร์โดยดูจาก log ของโปรแกรม WEKA

4) ทำการดึงเฉพาะข้อมูลจุดเซนทรอยด์ซึ่งเป็นตัวแทนของแต่ละคลัสเตอร์มาสร้างเป็นไฟล์จุดเซนทรอยด์เพื่อเป็นการเตรียมพร้อมสำหรับการเปรียบเทียบคลัสเตอร์ที่พบกับ คลัสเตอร์ต้นแบบ โดยใช้ Weka2NC เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS*_n[%2n][sampler][%s][i].txt) แล้วบันทึกผลเป็น DS*_n[%2n] [sampler][%s][i].nc

5) เปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับเซนทรอยด์ของคลัสเตอร์ต้นแบบ โดยใช้ *NC*metric ซึ่งจะทำการเปรียบเทียบเซนทรอยด์จากไฟล์จุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ (DS*_ori.nc) กับไฟล์จุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS*_n[%2n][sampler][%s][i].nc) และกำหนดค่า ξ เท่ากับ 0.01 แล้วบันทึกจำนวนจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบที่สามารถพบได้จากไฟล์ข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่ได้จาก WEKA (*NC*)

การทดสอบประสิทธิภาพของอัลกอริธึม DBSPACE

สำหรับเนื้อหาในส่วนนี้จะเป็นการนำเสนอวิธีการทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล DBSPACE ซึ่งพัฒนาขึ้นโดยผู้วิจัย การทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลแบ่งออกเป็นสองส่วนคือ การทดสอบประสิทธิภาพของอัลกอริธึมสำหรับชุดข้อมูลที่ไม่มีข้อมูลรบกวน และการทดสอบประสิทธิภาพของอัลกอริธึมสำหรับชุดข้อมูลที่มีข้อมูลรบกวนจำนวนหนึ่ง ในการทดลองจะทำการทดสอบคุณภาพของชุดข้อมูลสุ่มโดยการทดลองปรับค่าพารามิเตอร์ของอัลกอริธึม (δ: delta) จาก 0, 1, 2, …, 8 โดยใช้อัตราการสุ่มคงที่ และทำการศึกษาคุณสมบัติของทั้ง weighted discordancy ($\overline{\Gamma}$: สมการที่ 3-2) และ non-weighted discordancy ($\tilde{\Gamma}$: สมการที่ 3-3) ไปพร้อมๆ กัน การทดลองมีขั้นตอนดังต่อไปนี้

1) ทำการสร้างชุดข้อมูลสุ่มในรูปแบบ ARFF ด้วยอัลกอริธึมสุ่มข้อมูล DBSPACE ด้วยอัตราการสุ่มคงที่ โดยทดลองปรับค่าพารามิเตอร์ของอัลกอริธึม (δ: delta) จาก 0, 1, 2, …, 8 พร้อมทั้งบันทึกเวลาและหน่วยความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.7

2) ทำการสร้างชุดข้อมูลสุ่มในรูปแบบ ARFF ด้วยอัลกอริธึมสุ่มข้อมูล DBSPACE ด้วยอัตราการสุ่มคงที่บนชุดข้อมูลที่มีข้อมูลรบกวนจำนวนหนึ่ง โดยทดลองปรับค่าพารามิเตอร์ของอัลกอริธึม (δ: delta) จาก 0, 1, 2, …, 8 พร้อมทั้งบันทึกเวลาและหน่วยความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.8

ตารางที่ 3.7 แสดงการสร้างชุดข้อมูลสุ่มเพื่อทดสอบอัลกอริธึม DBSPACE

ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS2.dat	DBSPACE ($\bar{\Gamma}$) DBSPACE ($\tilde{\Gamma}$)	[1,2,3,…,8] (5 iterations)	DS*_DP[Γ][δ][%s][i].arff
รวม: 1	2	8 x 5	80 ชุดข้อมูล

ตารางที่ 3.8 แสดงการสร้างชุดข้อมูลสุ่มจากชุดข้อมูลรบกวนเพื่อทดสอบอัลกอริธึม DBSPACE

ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS2_n[%2n].dat	DBSPACE ($\bar{\Gamma}$) DBSPACE ($\tilde{\Gamma}$)	[1,2,3,…,8] (5 iterations)	DS*_DP[Γ][δ][%s][i] _n[%2n].arff
รวม: 1	2	8 x 5	80 ชุดข้อมูล

3) ทำการค้นหาคลัสเตอร์จากชุดข้อมูลที่สร้างขึ้นทั้งหมดจากข้อ 1 และ 2 โดยใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และกำหนดค่า k (จำนวนคลัสเตอร์ที่ต้องการค้นหา) เท่ากับจำนวนคลัสเตอร์ที่มีอยู่จริงของชุดข้อมูลตั้งต้น จากนั้นบันทึกผลลัพธ์ลงในไฟล์ *.txt และบันทึกเวลาที่ใช้ในการค้นหาคลัสเตอร์โดยดูจาก log ของโปรแกรม WEKA

4) ทำการดึงเฉพาะข้อมูลจุดเซนทรอยด์ซึ่งเป็นตัวแทนของแต่ละคลัสเตอร์มาสร้างเป็นไฟล์จุดเซนทรอยด์เพื่อเป็นการเตรียมพร้อมสำหรับการเปรียบเทียบคลัสเตอร์ที่พบกับ คลัสเตอร์ต้นแบบ โดยใช้ Weka2*NC* เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS***.txt) แล้วบันทึกผลที่ได้ลงบนไฟล์ชื่อ DS***.nc

5) เปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับเซนทรอยด์ของคลัสเตอร์ต้นแบบ โดยใช้ *NC*metric ซึ่งจะทำการเปรียบเทียบเซนทรอยด์จากไฟล์จุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ (DS*_ori.nc) กับไฟล์จุดเซนทรอยด์ของคลัสเตอร์ที่พบ DS***.nc และกำหนดค่า ξ เท่ากับ 0.01 แล้วบันทึกจำนวนจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบที่สามารถพบได้จากไฟล์ข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบโดย WEKA (*NC*)

การเปรียบเทียบประสิทธิภาพของอัลกอริธึม DBSPACE กับอัลกอริธึมอื่นๆ

เพื่อความกระชับของการทดลองจึงเลือกที่จะทดสอบอัลกอริธึม DBSPACE กับเฉพาะอัลกอริธึมที่ให้ผลลัพธ์ดีที่สุดเพียงอัลกอริธึมเดียว โดยในการทดลองจะกำหนดค่าพารามิเตอร์ delta ของอัลกอริธึม DBSPACE เป็นค่าคงที่ค่าหนึ่ง และสร้างชุดข้อมูลสุ่มของ DBSPACE จากทั้ง weighted discordancy และ non-weighted discordancy การทดลองมีขั้นตอนดังต่อไปนี้

1) ทำการสร้างชุดข้อมูลสุ่มในรูปแบบ ARFF ด้วยอัลกอริธึมสุ่มข้อมูลด้วยอัตราการสุ่มต่างๆ พร้อมทั้งบันทึกเวลาและหน่วยความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.9

ตารางที่ 3.9 แสดงการสร้างชุดข้อมูลสุ่มสำหรับการเปรียบเทียบอัลกอริธึม DBSPACE

ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS2.dat	BEST DBSPACE ($\bar{\Gamma}$) DBSPACE ($\tilde{\Gamma}$)	[1,2,3,…,10] (5 *i*terations)	DS*_ BEST[%s][i].arff DS*_DP[1][δ][%s][i].arff DS*_DP[2][δ][%s][i].arff
รวม: 1	3	10 x 5	150 ชุดข้อมูล

2) ทำการค้นหาคลัสเตอร์จากชุดข้อมูลที่สร้างขึ้นทั้งหมดจากข้อ 1 โดยใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และกำหนดค่า k (จำนวนคลัสเตอร์ที่ต้องการค้นหา) เท่ากับจำนวนคลัสเตอร์ที่มีอยู่จริงของชุดข้อมูลตั้งต้น จากนั้นบันทึกผลลัพธ์ลงในไฟล์ *.txt และบันทึกเวลาที่ใช้ในการค้นหาคลัสเตอร์โดยดูจาก log ของโปรแกรม WEKA

3) ทำการดึงเฉพาะข้อมูลจุดเซนทรอยด์ซึ่งเป็นตัวแทนของแต่ละคลัสเตอร์มาสร้างเป็นไฟล์จุดเซนทรอยด์เพื่อเป็นการเตรียมพร้อมสำหรับการเปรียบเทียบคลัสเตอร์ที่พบกับ คลัสเตอร์ต้นแบบ โดยใช้ Weka2*NC* เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS***.txt) แล้วบันทึกผลที่ได้ลงบนไฟล์ชื่อ DS***.nc

4) เปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับเซนทรอยด์ของคลัสเตอร์ต้นแบบ โดยใช้ *NC*metric ซึ่งจะทำการเปรียบเทียบเซนทรอยด์จากไฟล์จุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ (DS*_ori.nc) กับไฟล์จุดเซนทรอยด์ของคลัสเตอร์ที่พบ DS***.nc และกำหนดค่า ξ เท่ากับ 0.01 แล้วบันทึกจำนวนจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบที่สามารถพบได้จากไฟล์ข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบโดย WEKA (*NC*)

การเปรียบเทียบคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึม DBSPACE

1) ทำการสุ่มข้อมูลจากชุดข้อมูลรบกวนที่ได้จากตารางที่ 3.5 ด้วยอัลกอริธึมที่ต้องการทดสอบในอัตราการสุ่มเท่ากันที่สามารถให้ผลลัพธ์ของการค้นหาคลัสเตอร์ได้ดี พร้อมทั้งบันทึกเวลาและหน่วยความจำที่ใช้ในกระบวนการสุ่มข้อมูล ซึ่งสามารถสรุปได้ดังตารางที่ 3.10

ตารางที่ 3.10 แสดงการสร้างชุดข้อมูลสุ่มจากชุดข้อมูลรบกวนเพื่อเปรียบเทียบอัลกอริธึม DBSPACE

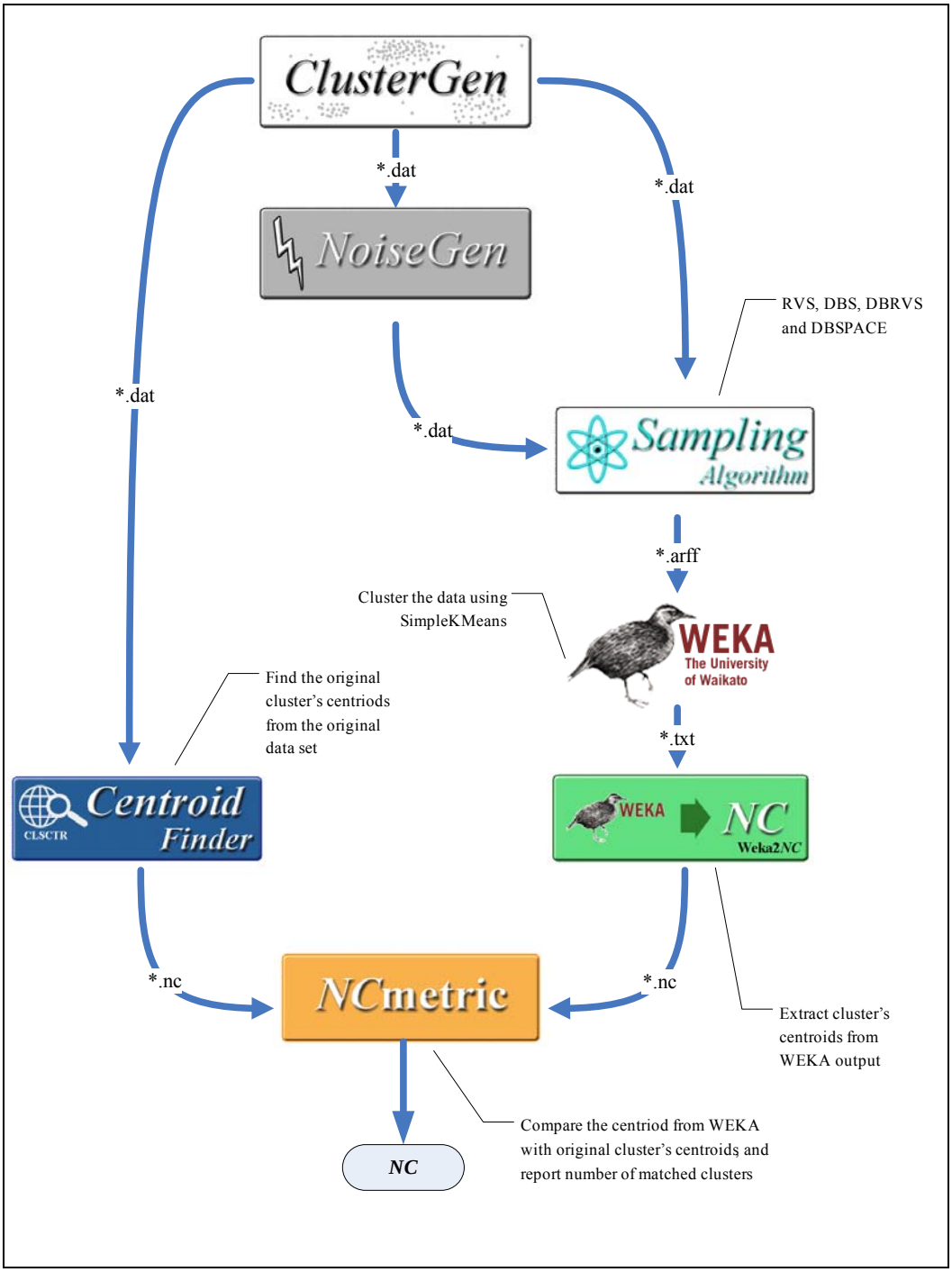
ชุดข้อมูลทดสอบ	อัลกอริธึม	อัตราสุ่ม (%)	ชุดข้อมูลสุ่มผลลัพธ์
DS*_n[%2n].dat	BEST DBSPACE ($\bar{\Gamma}$) DBSPACE ($\tilde{\Gamma}$)	*c* (5 *i*terations)	DS*_BEST[%s][i] _n[%2n].arff DS*_DP[1][δ][%s][i] _n[%2n].arff DS*_DP[2][δ][%s][i] _n[%2n].arff
รวม: 14	3	1 x 5	210 ชุดข้อมูล

2) ทำการค้นหาคลัสเตอร์จากชุดข้อมูลที่สร้างขึ้นทั้งหมดจากข้อ 1 โดยใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และกำหนดค่า k (จำนวนคลัสเตอร์ที่ต้องการค้นหา) เท่ากับจำนวนคลัสเตอร์ที่มีอยู่จริงของชุดข้อมูลตั้งต้น จากนั้นบันทึกผลลัพธ์ลงในไฟล์ *.txt และบันทึกเวลาที่ใช้ในการค้นหาคลัสเตอร์โดยดูจาก log ของโปรแกรม WEKA

3) ทำการดึงเฉพาะข้อมูลจุดเซนทรอยด์ซึ่งเป็นตัวแทนของแต่ละคลัสเตอร์มาสร้างเป็นไฟล์จุดเซนทรอยด์เพื่อเป็นการเตรียมพร้อมสำหรับการเปรียบเทียบคลัสเตอร์ที่พบกับ คลัสเตอร์ต้นแบบ โดยใช้ Weka2*NC* เพื่อทำการดึงข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบจาก WEKA (DS***.txt) แล้วบันทึกผลที่ได้ลงบนไฟล์ชื่อ DS***.nc

4) เปรียบเทียบเซนทรอยด์ของคลัสเตอร์ที่พบกับเซนทรอยด์ของคลัสเตอร์ต้นแบบ โดยใช้ *NC*metric ซึ่งจะทำการเปรียบเทียบเซนทรอยด์จากไฟล์จุดเซนทรอยด์ของคลัสเตอร์ต้นแบบ (DS*_ori.nc) กับไฟล์จุดเซนทรอยด์ของคลัสเตอร์ที่พบโดย WEKA (DS***.nc) และกำหนดค่า ξ เท่ากับ 0.01 แล้วบันทึกจำนวนจุดเซนทรอยด์ของคลัสเตอร์ต้นแบบที่สามารถพบได้จากไฟล์ข้อมูลจุดเซนทรอยด์ของคลัสเตอร์ที่พบโดย WEKA (*NC*)

ขั้นตอนการทดลองทั้งหมดสามารถแสดงเป็นแผนภาพโดยสรุปได้ดังรูปที่ 3.16



รูปที่ 3.16 แผนภาพสรุปวิธีการทดสอบอัลกอริธึม

บทที่ 4
ผลการวิเคราะห์ข้อมูลและการอภิปรายผล

การทดลองในการวิจัยครั้งนี้จะแบ่งออกเป็นสองส่วนคือ การทดสอบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS เพื่อวิเคราะห์หาลักษณะเด่นของแต่ละอัลกอริธึม และการทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลแบบใหม่ที่พัฒนาขึ้น (DBSPACE) โดยในการทดลองแต่ละส่วนจะทำการศึกษาคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึมนั้นๆ ด้วยเกณฑ์ในการพิจารณาความเหมาะสมของแต่ละเทคนิคประกอบด้วย เวลาที่ใช้ในการสุ่มข้อมูล (time to sample) จำนวนหน่วยความจำที่ใช้ (memory usage) เวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลจากชุดข้อมูลสุ่ม (time to cluster) ตลอดจนความแม่นตรงของคลัสเตอร์ที่พบ (*number of cluster found*: *NC*) และความทนทานต่อข้อมูลรบกวนของแต่ละอัลกอริธึม ซึ่งจะทำการทดสอบกับชุดข้อมูลที่ถูกสร้างข้อมูลรบกวนเข้าไปในปริมาณต่างๆ ในการทดลองนี้จะใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA และทำการทดลองบนเครื่องคอมพิวเตอร์ Pentium4 1.7GHz หน่วยความจำหลัก 640MB บนระบบปฏิบัติการ Windows XP SP2

สำหรับเนื้อหาของบทนี้จะเป็นการนำเสนอผลการทดลองจากการทดสอบประสิทธิภาพการสุ่มข้อมูลของแต่ละอัลกอริธึม โดยจะนำเสนอตามลำดับดังต่อไปนี้

4.1 ผลการเปรียบเทียบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS

4.2 ผลการเปรียบเทียบความทนทานต่อข้อมูลรบกวนของอัลกอริธึม RVS, DBS, และ DBRVS

4.3 ผลการทดสอบประสิทธิภาพของอัลกอริธึม DBSPACE

4.4 ผลการเปรียบเทียบประสิทธิภาพของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS

4.5 ผลการเปรียบเทียบความทนทานต่อข้อมูลรบกวนของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS

4.1 ผลการเปรียบเทียบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS

ตารางที่ 4.1 ถึง 4.3 ต่อไปนี้แสดงผลการทดสอบประสิทธิภาพบนข้อมูล 2 มิติจำนวน 72703 เรคคอร์ดซึ่งมีขนาด 10 คลัสเตอร์ ของอัลกอริธึม RVS, DBS, และ DBRVS ตามลำดับ โดยในตารางจะแสดงผลการทดลองบนชุดข้อมูลสุ่มขนาดต่างๆ

ตารางที่ 4.1 แสดงผลการทดสอบอัลกอริธึม RVS บนชุดข้อมูล DS1

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (10)
1	0.440	17448	0.0	4.0
2	0.440	34896	0.0	4.0
3	0.440	52344	1.0	3.0
4	0.430	69792	2.0	3.0
5	0.430	87240	3.0	0.0
6	0.480	104688	2.0	0.0
7	0.500	122136	4.0	0.0
8	0.480	139584	4.0	0.0
9	0.490	157032	4.0	0.0
10	0.500	174480	5.0	0.0
ข้อมูลทั้งหมด	-	-	17.0	7.0

ตารางที่ 4.2 แสดงผลการทดสอบอัลกอริธึม DBS บนชุดข้อมูล DS1

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (10)
1	0.540	92344	1.0	6.0
2	0.570	144688	0.0	7.0
3	0.620	197032	1.0	6.0
4	0.640	249376	0.0	7.0
5	0.650	301720	1.0	7.0
6	0.660	354064	1.0	6.0
7	0.680	406408	1.0	6.0
8	0.690	458752	1.0	7.0
9	0.690	511096	1.0	7.0
10	0.721	563440	3.0	7.0
ข้อมูลทั้งหมด	-	-	17.0	7.0

ตารางที่ 4.3 แสดงผลการทดสอบอัลกอริธึม DBRVS บนชุดข้อมูล DS1

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (10)
1	0.460	80216	0.0	0.0
2	0.450	80400	0.0	1.0
3	0.500	80400	0.0	1.0
4	0.510	80480	0.0	2.0
5	0.530	80520	0.0	1.0
6	0.530	80640	1.0	1.0
7	0.570	80720	2.0	3.0
8	0.540	80800	0.0	3.0
9	0.540	80880	1.0	3.0
10	0.510	81000	0.0	4.0
ข้อมูลทั้งหมด	-	-	17.0	7.0

ตารางที่ 4.4 ถึง 4.6 ต่อไปนี้แสดงผลการทดสอบประสิทธิภาพบนข้อมูล 2 มิติจำนวน 58822 เรคคอร์ดซึ่งมีขนาด 20 คลัสเตอร์ ของอัลกอริธึม RVS, DBS, และ DBRVS ตามลำดับ โดยในตารางจะแสดงผลการทดลองบนชุดข้อมูลสุ่มขนาดต่างๆ กัน

ตารางที่ 4.4 แสดงผลการทดสอบอัลกอริธึม RVS บนชุดข้อมูล DS2

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.350	40160	0.0	2.0
2	0.360	40192	1.0	2.0
3	0.430	40320	1.0	3.0
4	0.360	40360	1.0	4.0
5	0.350	40400	0.0	6.0
6	0.390	40480	1.0	4.0
7	0.390	40560	2.0	4.0
8	0.390	40640	1.0	3.0
9	0.390	40720	1.0	5.0
10	0.390	40800	3.0	3.0
ข้อมูลทั้งหมด	-	-	36.0	11.0

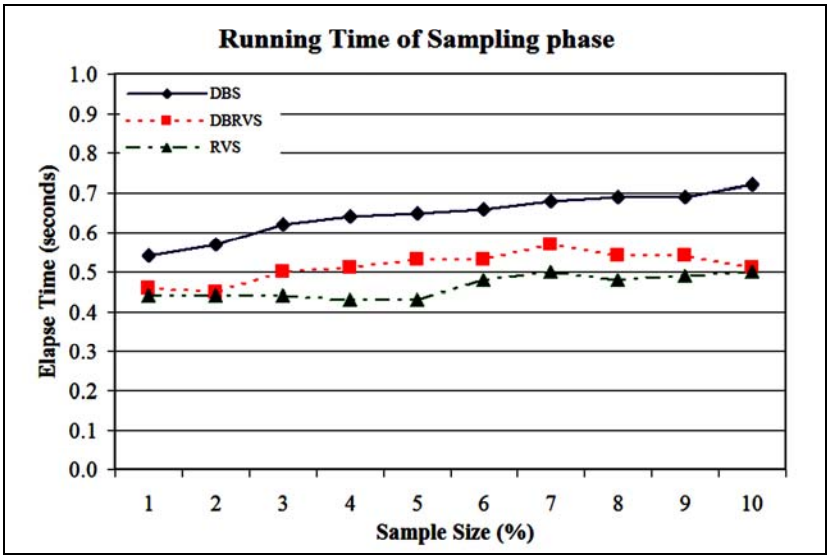
ตารางที่ 4.5 แสดงผลการทดสอบอัลกอริธึม DBS บนชุดข้อมูล DS2

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.440	82336	0.0	8.0
2	0.460	124672	0.0	11.0
3	0.500	167080	1.0	12.0
4	0.520	209416	1.0	13.0
5	0.530	251752	1.0	12.0
6	0.540	294088	0.0	11.0
7	0.550	336496	2.0	13.0
8	0.560	378832	1.0	12.0
9	0.560	421168	3.0	11.0
10	0.580	463504	2.0	14.0
ข้อมูลทั้งหมด	-	-	36.0	11.0

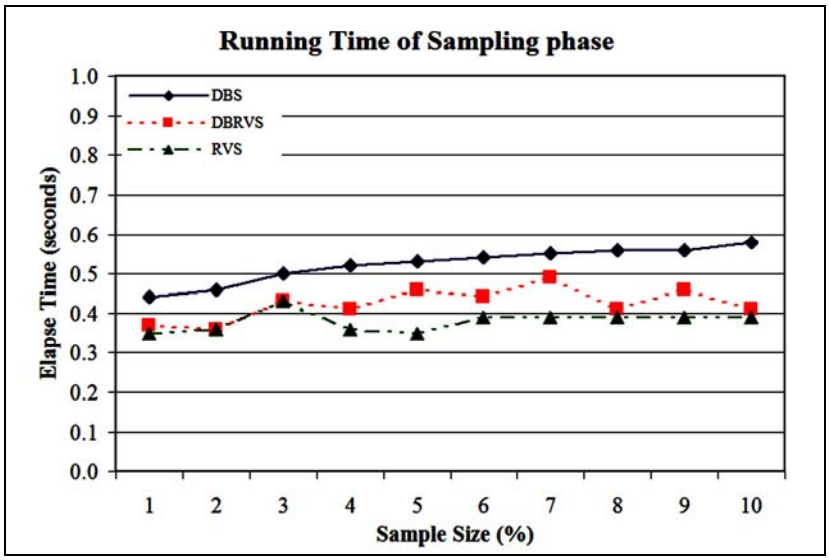
ตารางที่ 4.6 แสดงผลการทดสอบอัลกอริธึม DBRVS บนชุดข้อมูล DS2

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.370	588	0.0	5.0
2	0.360	1176	2.0	6.0
3	0.430	1765	1.0	5.0
4	0.410	2353	2.0	3.0
5	0.460	2941	4.0	1.0
6	0.440	3529	5.0	1.0
7	0.490	4118	8.0	1.0
8	0.410	4706	15.0	2.0
9	0.460	5294	9.0	2.0
10	0.410	5882	6.0	2.0
ข้อมูลทั้งหมด	-	-	36.0	11.0

จากข้อมูลในตารางที่ 4.1 ถึง 4.6 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับเวลาที่ใช้ในการสุ่มเปรียบเทียบกันระหว่างแต่ละอัลกอริธึม สำหรับการสุ่มข้อมูลบนชุดข้อมูล DS1 และ DS2 จะได้ผลดังรูปที่ 4.1 และ 4.2 ตามลำดับ ดังนี้

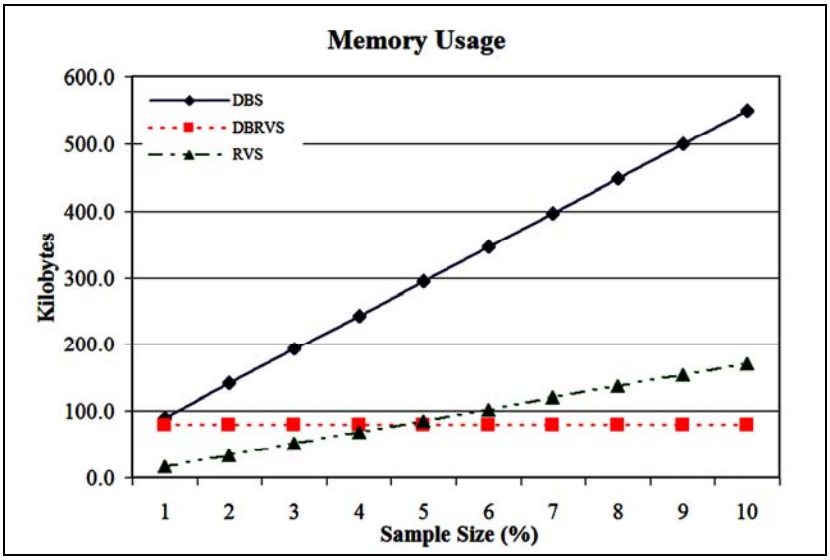


รูปที่ 4.1 เวลาที่ใช้ในการสุ่มข้อมูลของแต่ละอัลกอริธึมบนชุดข้อมูล DS1

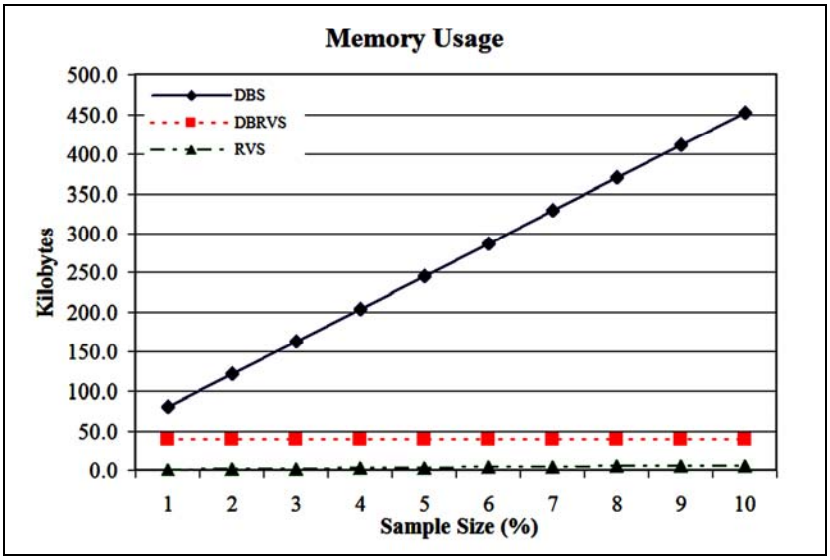


รูปที่ 4.2 เวลาที่ใช้ในการสุ่มข้อมูลของแต่ละอัลกอริธึมบนชุดข้อมูล DS2

จากข้อมูลในตารางที่ 4.1 ถึง 4.6 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับหน่วยความจำที่ใช้ในการสุ่มข้อมูลเปรียบเทียบกันระหว่างแต่ละอัลกอริธึม สำหรับการสุ่มข้อมูลบนชุดข้อมูล DS1 และ DS2 จะได้ผลดังรูปที่ 4.3 และ 4.4 ตามลำดับ ดังนี้

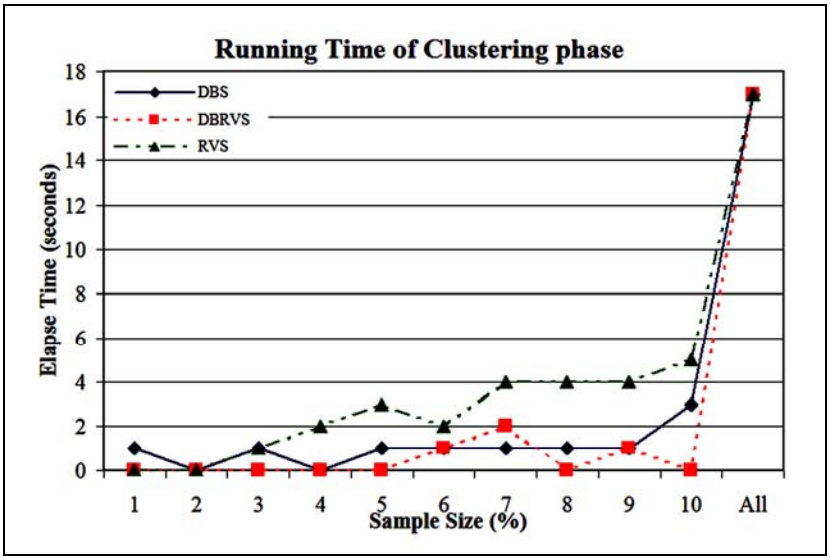


รูปที่ 4.3 หน่วยความจำที่ใช้ของแต่ละอัลกอริธึมบนชุดข้อมูล DS1

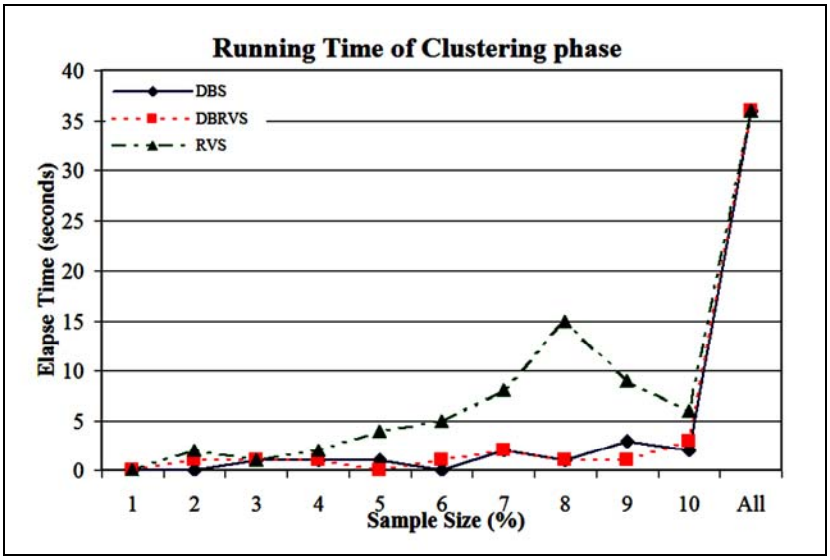


รูปที่ 4.4 หน่วยความจำที่ใช้ของแต่ละอัลกอริธึมบนชุดข้อมูล DS2

จากข้อมูลในตารางที่ 4.1 ถึง 4.6 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับเวลาที่ใช้ในการค้นหาคลัสเตอร์ด้วย *k*-means เปรียบเทียบกันระหว่างแต่ละอัลกอริธึม สำหรับการสุ่มข้อมูลบนชุดข้อมูล DS1 และ DS2 จะได้ผลดังรูปที่ 4.5 และ 4.6 ตามลำดับ ดังนี้

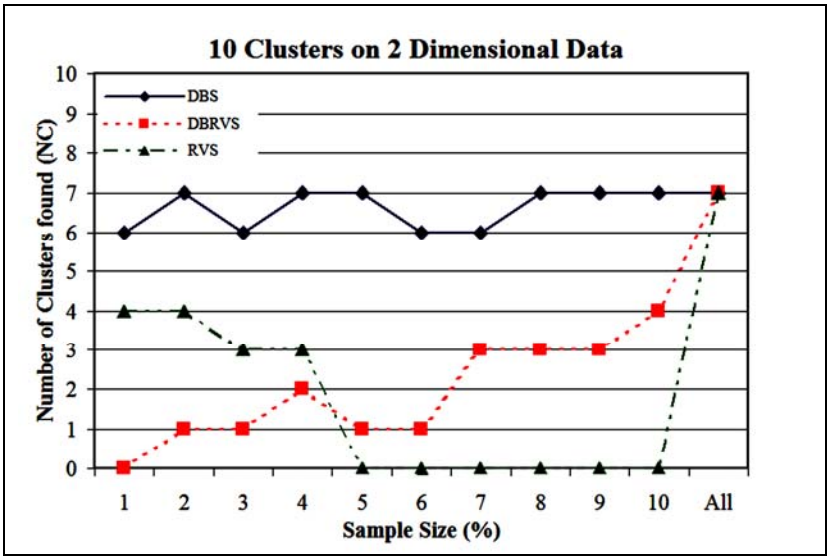


รูปที่ 4.5 เวลาที่ใช้ในการค้นหาคลัสเตอร์บนชุดข้อมูลสุ่มของแต่ละอัลกอริธึมจากชุดข้อมูล DS1

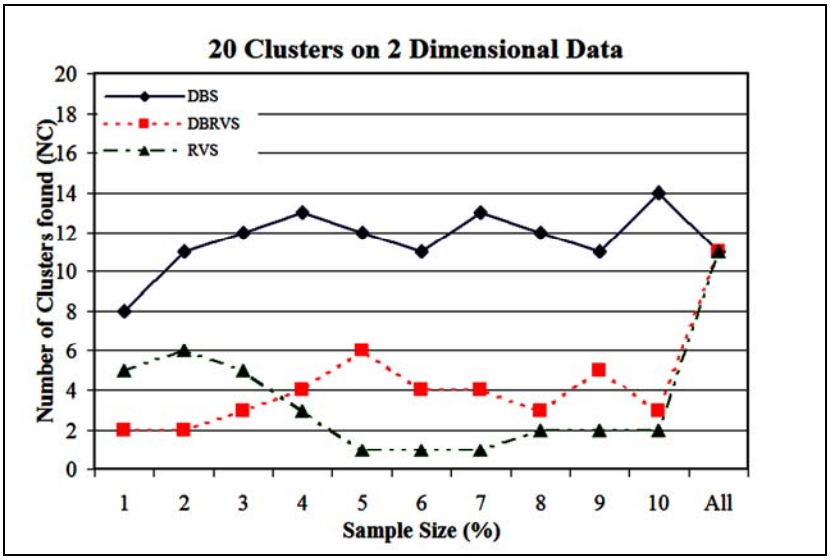


รูปที่ 4.6 เวลาที่ใช้ในการค้นหาคลัสเตอร์บนชุดข้อมูลสุ่มของแต่ละอัลกอริธึมจากชุดข้อมูล DS2

จากข้อมูลในตารางที่ 4.1 ถึง 4.6 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับจำนวนคลัสเตอร์แท้ที่สามารถพบได้จากการค้นหาด้วยอัลกอริธึม *k*-means เปรียบเทียบกันระหว่างแต่ละอัลกอริธึม สำหรับการสุ่มข้อมูลบนชุดข้อมูล DS1 และ DS2 จะได้ผลดังรูปที่ 4.7 และ 4.8 ตามลำดับ ดังนี้



รูปที่ 4.7 จำนวน *NC* ที่พบได้จากชุดข้อมูลสุ่มของแต่ละอัลกอริธึมจากชุดข้อมูล DS1



รูปที่ 4.8 จำนวน *NC* ที่พบได้จากชุดข้อมูลสุ่มของแต่ละอัลกอริธึมจากชุดข้อมูล DS2

จากกราฟดังปรากฏในรูปที่ 4.1 และ 4.2 แสดงให้เห็นว่าเมื่อเพิ่มอัตราการสุ่มข้อมูลมากขึ้น ทุกอัลกอริธึมจะใช้เวลาในการสุ่มเพิ่มมากขึ้น โดยที่อัลกอริธึม RVS จะใช้เวลาในการสุ่มข้อมูลน้อยที่สุดเนื่องจากอัลกอริธึมไม่ต้องทำการคำนวณค่าความหนาแน่นเพื่อใช้ในการสุ่มข้อมูล ในขณะที่อัลกอริธึม DBS จะใช้เวลาในการสุ่มข้อมูลมากกว่าทุกอัลกอริธึม และจากกราฟดังปรากฏในรูปที่ 4.3 และ 4.4 ยังแสดงให้เห็นว่าเมื่อเพิ่มอัตราการสุ่มข้อมูลมากขึ้น อัลกอริธึม RVS และ DBS จะใช้หน่วยความจำมากขึ้นเพราะจำเป็นจะต้องจองหน่วยความจำไว้ตามขนาดของชุดข้อมูลสุ่มที่ต้องการ ในขณะที่อัลกอริธึม DBRVS จะใช้หน่วยความจำเพิ่มขึ้นเพียงเล็กน้อยเพื่อใช้เก็บข้อมูลสรุปของกลุ่มข้อมูลที่ถูกสุ่มขึ้นมาเท่านั้น ซึ่งจากรูปที่ 4.3 จะเห็นได้ว่าเมื่อเพิ่มอัตราการสุ่มข้อมูลในอัตราสูงกว่า 4.5% อัลกอริธึม RVS ต้องการหน่วยความจำมากกว่าอัลกอริธึม DBRVS ในขณะที่อัลกอริธึม DBS ใช้หน่วยความจำมากกว่าทุกอัลกอริธึม

เมื่อนำชุดข้อมูลสุ่มของแต่ละอัลกอริธึมมาผ่านเข้าสู่กระบวนการจัดกลุ่มข้อมูลอัตโนมัติเพื่อค้นหาคลัสเตอร์แท้ที่ยังคงสามารถพบได้บนชุดข้อมูลสุ่มเหล่านั้น ผลที่ได้ดังรูปที่ 4.7 และ 4.8 แสดงให้เห็นว่า DBS สามารถให้ผลลัพธ์ของชุดข้อมูลสุ่มที่ยังคงข้อมูลที่สามารถแสดงถึง คลัส-เตอร์ที่ถูกต้องได้มากที่สุด โดยที่การสุ่มข้อมูลด้วยอัลกอริธึม DBS ขนาด 2% สามารถให้ผลลัพธ์ของจำนวน *NC* ได้เทียบเท่ากับการใช้ข้อมูลตั้งต้นทั้งหมด ส่วนชุดข้อมูลสุ่มของอัลกอริธึม DBRVS จะสามารถพบ *NC* ได้สูงขึ้นเมื่อเพิ่มอัตราการสุ่มให้มากขึ้น โดยในอัตราการสุ่มที่น้อยๆ ชุดข้อมูลสุ่มของอัลกอริธึม RVS จะสามารถพบ *NC* ได้สูงกว่าชุดข้อมูลสุ่มของอัลกอริธึม DBRVS แต่เมื่อเพิ่มอัตราการสุ่มจนถึงระดับหนึ่ง จำนวน *NC* ที่ได้จากชุดข้อมูลสุ่มของอัลกอริธึม RVS เริ่มลดลงเรื่อยๆ และจะสูงขึ้นอีกครั้งเมื่อเพิ่มอัตราการสุ่มให้มากขึ้นจนถึงระดับหนึ่ง เนื่องจากอัลกอริธึม RVS จะทำการสุ่มข้อมูลไปเรื่อยๆ โดยไม่ได้คำนึงถึงความสำคัญของข้อมูลแต่ละเรค-คอร์ด ข้อมูลที่ถูกเลือกจะถูกเก็บไว้ในหน่วยความจำ (reservoir buffer) และเมื่อหน่วยความจำเต็มอัลกอริธึมจะสุ่มเลือกข้อมูลที่เคยถูกเลือกออกเพื่อให้หน่วยความจำมีที่ว่างพอสำหรับข้อมูลใหม่ที่จะถูกเลือกต่อไป ซึ่งการสุ่มเลือกเอาข้อมูลใดๆ ออกจากหน่วยความจำเป็นเพียงการสุ่มโดยมิได้สนใจว่าข้อมูลที่เคยถูกเลือกไปแล้วนั้นมีความสำคัญมากน้อยเพียงใดเมื่อเทียบกับข้อมูลใหม่ที่กำลังถูกเลือกเข้ามา ด้วยเหตุนี้ข้อมูลที่เคยถูกเลือกไปในช่วงแรกจึงมีความน่าจะเป็นที่จะถูกแทนที่ด้วยข้อมูลที่จะถูกเลือกในเวลาต่อมาเสมอ ซึ่งอาจกล่าวได้ว่าอัลกอริธึม RVS จะให้ความสำคัญกับข้อมูลที่ถูกเลือกล่าสุดมากกว่าข้อมูลที่ถูกเลือกไปแล้ว และด้วยเหตุนี้เมื่อกำหนดขนาดของชุดข้อมูลสุ่มน้อยๆ (เมื่อเทียบกับข้อมูลทั้งหมด) ก็ยิ่งทำให้ข้อมูลส่วนต้นๆ ที่เคยถูกเลือกมีโอกาสที่จะถูกแทนที่สูงขึ้นไปด้วย โดยถ้าหากกลุ่มข้อมูลที่มีขนาดเล็กเรียงตัวกันอยู่ในส่วนต้นของชุดข้อมูล การสุ่มด้วยอัลกอริธึม RVS อาจทำให้กลุ่มข้อมูลดังกล่าวถูกละเลยได้ และจากการทดลองเมื่อเปรียบเทียบเวลาที่ใช้ในการค้นหาคลัสเตอร์ของแต่ละอัลกอริธึมพบว่า การค้นหาคลัสเตอร์บนชุดข้อมูลสุ่มของ

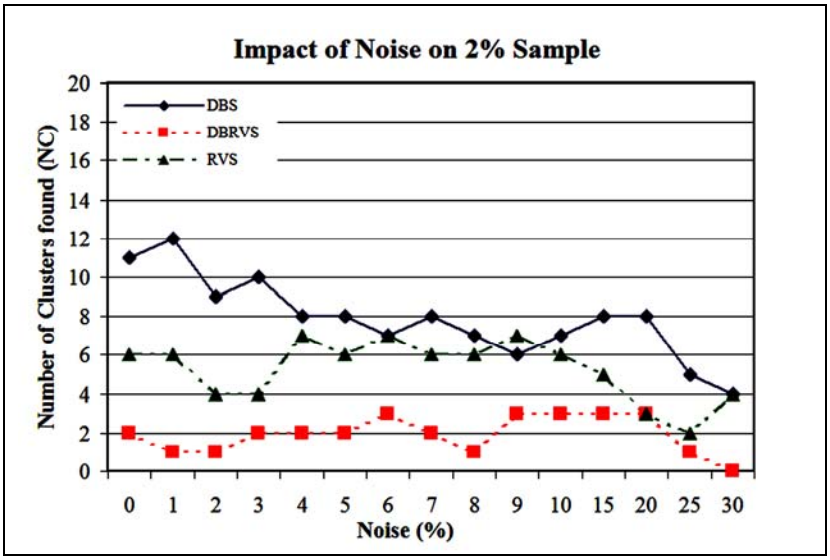
อัลกอริธึม RVS จะใช้เวลามากที่สุด ในขณะที่การค้นหาคลัสเตอร์บนชุดข้อมูลสุ่มของอัลกอริธึม DBS และ DBRVS จะใช้เวลาน้อยกว่าและใช้เวลาในการค้นหาใกล้เคียงกัน (ดังรูปที่ 4.5 และ 4.6)

4.2 ผลการเปรียบเทียบความทนทานต่อข้อมูลรบกวนของอัลกอริธึม RVS, DBS, และ DBRVS

สำหรับการเปรียบเทียบคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึม RVS, DBS, และ DBRVS สามารถทำได้โดยการสร้างชุดข้อมูลสุ่มด้วยอัลกอริธึมสุ่มข้อมูลทั้งสามกับข้อมูลที่มีข้อมูลรบกวนกระจายอยู่ในปริมาณต่างๆ จากนั้นจึงนำชุดข้อมูลสุ่มที่ได้มาผ่านกระบวนการจัดกลุ่มข้อมูลและวิเคราะห์หาจำนวน *NC* ที่พบในแต่ละชุดข้อมูล ซึ่งผลการทดลองที่ได้แสดงไว้ดังตารางที่ 4.7 และกราฟดังรูปที่ 4.9 โดยจะแสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 2% ของแต่ละอัลกอริธึมบนชุดข้อมูลตั้งต้น (DS2) ขนาด 2 มิติ 20 คลัสเตอร์ ที่ถูกสร้างข้อมูลรบกวนในปริมาณตั้งแต่ 1% จนถึง 30% ของจำนวนข้อมูลทั้งหมด

ตารางที่ 4.7 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 2% ของแต่ละอัลกอริธึมบนชุดข้อมูลตั้งต้น (DS2) ที่ถูกสร้างข้อมูลรบกวนในปริมาณตั้งแต่ 1% จนถึง 30%

อัลกอริธึมสุ่มข้อมูล / ปริมาณข้อมูลรบกวน (%)	RVS	DBS	DBRVS
0% (ข้อมูลปกติ)	6.0	11.0	2.0
1 %	6.0	12.0	1.0
2 %	4.0	9.0	1.0
3 %	4.0	10.0	2.0
4 %	7.0	8.0	2.0
5 %	6.0	8.0	2.0
6 %	7.0	7.0	3.0
7 %	6.0	8.0	2.0
8 %	6.0	7.0	1.0
9 %	7.0	6.0	3.0
10 %	6.0	7.0	3.0
15 %	5.0	8.0	3.0
20 %	3.0	8.0	3.0
25 %	2.0	5.0	1.0
30 %	4.0	4.0	0.0



รูปที่ 4.9 แสดงผลกระทบที่เกิดจากข้อมูลรบกวนบนชุดข้อมูลสุ่มขนาด 2% ของแต่ละอัลกอริธึม

จากผลการทดลองที่ได้แสดงไว้ดังตารางที่ 4.7 และกราฟดังรูปที่ 4.9 แสดงให้เห็นว่าเมื่อข้อมูลตั้งต้นมีข้อมูลรบกวนมากขึ้นจะส่งผลกระทบต่อคุณภาพของชุดข้อมูลสุ่มที่ได้จากทั้งสามอัลกอริธึม ดังแสดงได้จากการที่จำนวน *NC* ที่พบจะมีจำนวนลดลงเมื่อมีข้อมูลรบกวนมากขึ้น โดยจำนวน *NC* ที่ได้จากอัลกอริธึม DBS จะลดลงอย่างเห็นได้ชัดเมื่อข้อมูลเริ่มมีความไม่บริสุทธิ์ และรองลงมาคืออัลกอริธึม RVS ซึ่งจะยังคงรักษาจำนวน *NC* ไว้ได้จนเมื่อข้อมูลมีปริมาณข้อมูลรบกวนมากขึ้น (จนถึง 9% ดังรูปที่ 4.9) จำนวน *NC* ก็จะเริ่มลดลง ในขณะที่อัลกอริธึม DBRVS จะยังคงรักษาจำนวน *NC* ที่พบไว้ได้จนกระทั่งข้อมูลรบกวนมากขึ้นถึง 20% แม้ว่าจำนวน *NC* โดยรวมจะมีค่าน้อยกว่าทั้ง DBS และ RVS ก็ตาม ซึ่งทำให้ทราบได้ว่าการพิจารณาผลต่างของความหนาแน่นของสองกลุ่มข้อมูลที่อยู่ใกล้กันในอัลกอริธึม DBRVS สามารถช่วยกรองข้อมูลรบกวนไม่ให้เข้ามามีทบบาทในชุดข้อมูลสุ่มได้

4.3 ผลการทดสอบประสิทธิภาพของอัลกอริธึม DBSPACE

สำหรับเนื้อหาในส่วนนี้จะเป็นการนำเสนอผลการทดลองที่ได้จากการทดสอบและวิเคราะห์ประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล DBSPACE ซึ่งพัฒนาขึ้นโดยผู้วิจัย โดยได้ทำการทดสอบคุณภาพของชุดข้อมูลสุ่มจากการทดลองปรับค่าพารามิเตอร์ของอัลกอริธึม (δ: delta) จาก 0, 1, 2, …, 8 โดยใช้อัตราการสุ่มคงที่ที่ 5% ของจำนวนข้อมูลทั้งหมด และทำการศึกษาคุณสมบัติของทั้ง weighted discordancy (สมการที่ 3-2) และ non-weighted discordancy (สมการที่ 3-3) ไปพร้อมๆ กัน โดยผลการทดลองที่ได้แสดงได้ดังตารางที่ 4.8 และ 4.9

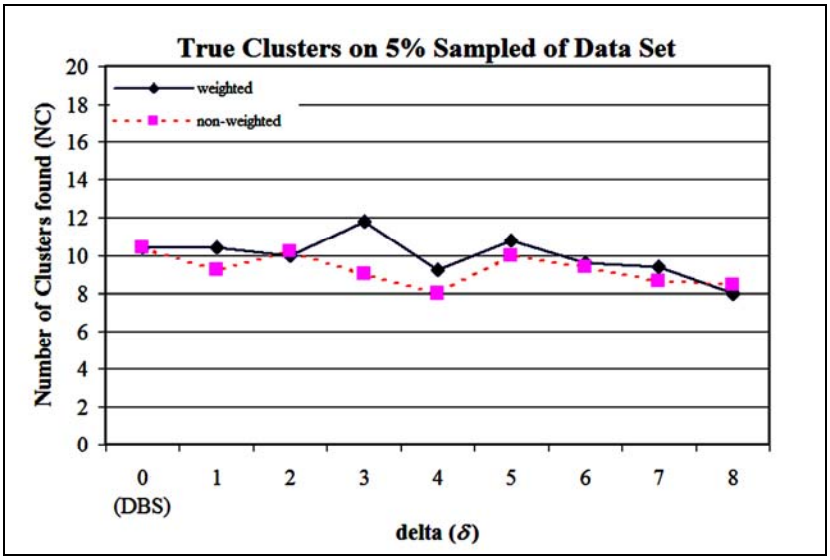
ตารางที่ 4.8 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBSPACE โดยใช้ weighted discordancy

ชุดข้อมูลที่ / delta (δ)	1	2	3	4	5	(*NC*) เฉลี่ย
0 (DBS)	10	11	11	10	10	10.4
1	9	13	7	10	13	10.4
2	10	13	9	11	7	10.0
3	9	10	14	15	11	11.8
4	11	7	10	7	11	9.2
5	12	11	12	11	8	10.8
6	6	9	10	11	12	9.6
7	9	12	6	7	13	9.4
8	10	11	6	3	10	8.0

ตารางที่ 4.9 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBSPACE โดยใช้ non-weighted discordancy

ชุดข้อมูลที่ / delta (δ)	1	2	3	4	5	(*NC*) เฉลี่ย
0 (DBS)	10	11	11	10	10	10.4
1	9	12	7	9	9	9.2
2	11	11	7	12	10	10.2
3	9	7	9	8	12	9.0
4	7	8	7	6	12	8.0
5	10	7	11	10	12	10.0
6	7	14	8	9	9	9.4
7	7	10	13	5	8	8.6
8	10	7	10	8	7	8.4

และเมื่อนำข้อมูลจากผลการทดลองดังตารางที่ 4.8 และ 4.9 มาสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างค่าพารามิเตอร์ delta กับจำนวน *NC* ที่พบจากการใช้สมการคำนวณค่า discordancy ทั้งสองจะได้กราฟดังแสดงได้ดังรูปที่ 4.10



รูปที่ 4.10 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBSPACE

จากรูปที่ 4.10 แสดงให้เห็นว่าเมื่อมีการปรับค่าพารามิเตอร์ delta เพิ่มขึ้น ก็ยังไม่สามารถทำให้พบคลัสเตอร์ได้จำนวนมากขึ้น สังเกตได้จากจำนวน *NC* ที่ได้มีจำนวนใกล้เคียงกันกับ DBS (delta = 0) ทั้ง weighted discordancy และ non-weighted discordancy จึงขยายการทดลองเพื่อศึกษาประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล DBSPACE ต่อไป โดยครั้งนี้จะใช้ชุดข้อมูลที่มีข้อมูลรบกวนกระจายอยู่ในปริมาณ 10% ของข้อมูลทั้งหมด ซึ่งผลการทดลองที่ได้แสดงไว้ดังตารางที่ 4.10 และ 4.11

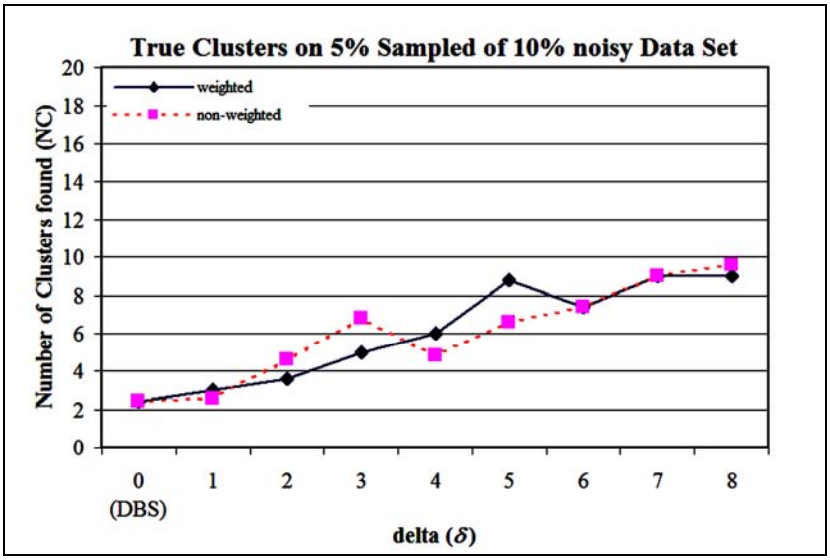
และเมื่อนำข้อมูลจากผลการทดลองดังตารางที่ 4.10 และ 4.11 มาสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างค่าพารามิเตอร์ delta กับจำนวน *NC* ที่พบโดยใช้สมการคำนวณค่า discordancy ทั้งสองจะได้กราฟดังแสดงได้ดังรูปที่ 4.11

ตารางที่ 4.10 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของชุดข้อมูลที่มีข้อมูลรบกวนปะปน 10% จากอัลกอริธึม DBSPACE โดยใช้ weighted discordancy

ชุดข้อมูลที่ / delta (δ)	1	2	3	4	5	(*NC*) เฉลี่ย
0 (DBS)	3	3	1	1	4	2.4
1	5	2	4	2	2	3.0
2	4	4	4	4	2	3.6
3	8	6	3	3	5	5.0
4	7	8	4	6	5	6.0
5	10	9	10	5	10	8.8
6	10	2	6	11	8	7.4
7	10	11	11	5	8	9.0
8	8	9	10	10	8	9.0

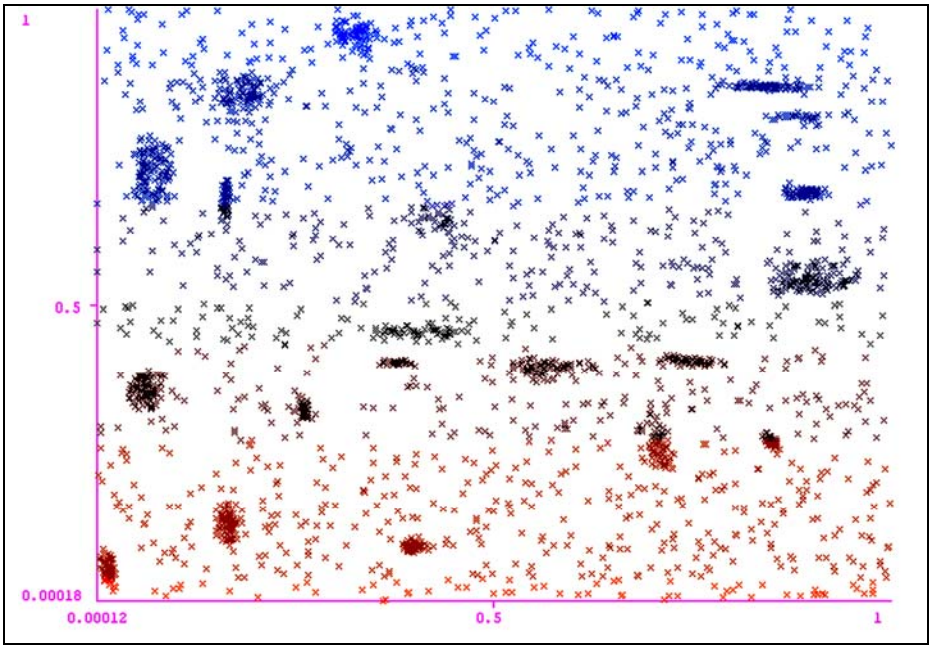
ตารางที่ 4.11 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของชุดข้อมูลที่มีข้อมูลรบกวนปะปน 10% จากอัลกอริธึม DBSPACE โดยใช้ non-weighted discordancy

ชุดข้อมูลที่ / delta (δ)	1	2	3	4	5	(*NC*) เฉลี่ย
0 (DBS)	3	3	1	1	4	2.4
1	3	2	1	3	4	2.6
2	2	6	2	6	7	4.6
3	9	4	5	7	9	6.8
4	6	9	3	3	3	4.8
5	6	8	7	8	4	6.6
6	6	7	8	7	9	7.4
7	6	9	11	9	10	9.0
8	12	11	7	10	8	9.6

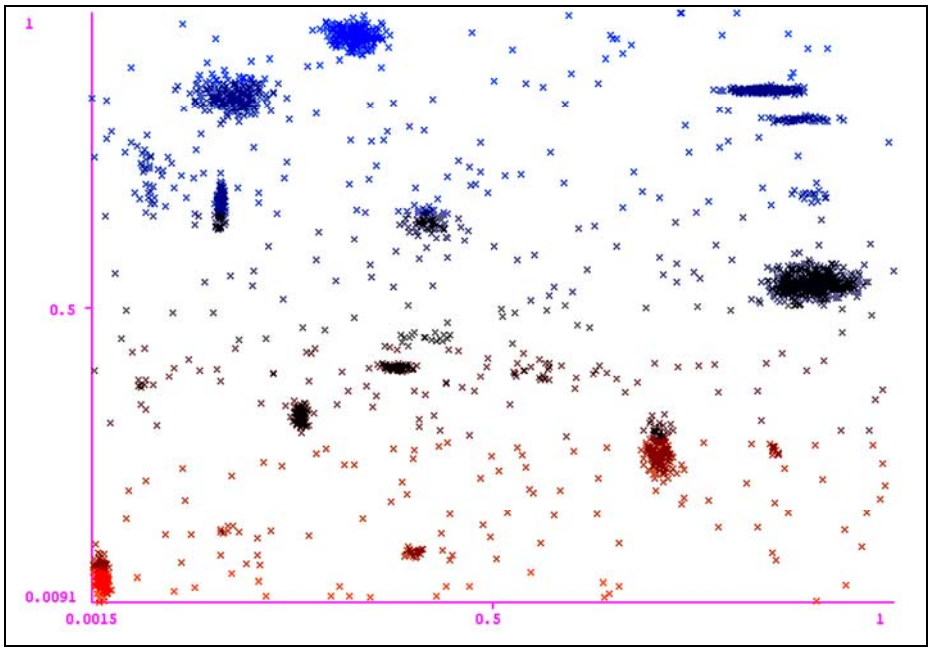


รูปที่ 4.11 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของชุดข้อมูลที่มีข้อมูลรบกวนปะปน 10% จากอัลกอริธึม DBSPACE

จากรูปที่ 4.11 แสดงให้เห็นว่าเมื่อทดสอบกับข้อมูลที่มีข้อมูลรบกวนปะปนและปรับค่าพารามิเตอร์ delta เพิ่มขึ้น จะทำให้ *k*-means สามารถพบจำนวน *NC* ได้มากขึ้น โดยที่ทั้ง weighted discordancy และ non-weighted discordancy สามารถให้ผลลัพธ์ค่อนข้างใกล้เคียงกัน ตัวอย่างของชุดข้อมูลสุ่มที่ได้จากการทดลองจากทั้ง DBS และ DPSPACE แสดงไว้ดังรูปที่ 4.12 และ 4.13 ตามลำดับ



รูปที่ 4.12 ชุดข้อมูลสุ่มขนาด 5% ของชุดข้อมูลที่มีข้อมูลรบกวนปะปน 10% ที่ได้จากอัลกอริธึม DBS



รูปที่ 4.13 ชุดข้อมูลสุ่มขนาด 5% ของชุดข้อมูลที่มีข้อมูลรบกวนปะปน 10% ที่ได้จากอัลกอริธึม DBSPACE (non-weighted discordancy, delta = 5)

จากรูปที่ 4.12 และ 4.13 แสดงให้เห็นอย่างชัดเจนว่าอัลกอริธึม DBSPACE สามารถสร้างชุดข้อมูลสุ่มที่มีประสิทธิผลในการกรองข้อมูลรบกวนได้ ดังจะเห็นได้จากการที่ข้อมูลสุ่มที่ได้จะเกาะกลุ่มกันอยู่อย่างชัดเจน โดยที่ข้อมูลรบกวนที่กระจายตัวต่างจากกลุ่มจะถูกเลือกเข้ามาในปริมาณที่น้อยกว่า DBS ทำให้ชุดข้อมูลที่ได้จากอัลกอริธึม DBSPACE มีความเป็นปึกแผ่นและมีความประสานกันภายในกลุ่มสูงกว่าชุดข้อมูลที่ได้จากอัลกอริธึม DBS และเมื่อปรับค่าพารามิเตอร์ delta ให้มีค่ามากขึ้นก็จะทำให้ได้ชุดข้อมูลสุ่มที่มีความเป็นปึกแผ่นสูงขึ้น และยังมีผลในการลดผลกระทบอันเกิดจากข้อมูลรบกวนที่ปะปนมากับชุดข้อมูลตั้งต้นได้อีกประการหนึ่ง

4.4 ผลการเปรียบเทียบประสิทธิภาพของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS

ในส่วนนี้จะเป็นการแสดงผลการทดลองการเปรียบเทียบประสิทธิภาพของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS โดยการเปรียบเทียบเวลาและหน่วยความจำที่ใช้ในการสุ่มข้อมูล และเวลาที่ใช้ในการค้นหาคลัสเตอร์ ตลอดจนจำนวน *NC* ที่สามารถพบได้ในชุดข้อมูลสุ่มของทั้งอัลกอริธึม DBSPACE และอัลกอริธึม DBS ในอัตราการสุ่มข้อมูลต่างๆ โดยการทดลองของ อัลกอริธึม DBSPACE กับอัลกอริธึม DBS ต่อไปนี้จะปรับความละเอียดในส่วนของการจัดข้อมูลให้กับพาร์ทิชันในปริมาณสูงขึ้นจากการทดลองที่ผ่านมาเพื่อให้ได้ผลที่ชัดเจนที่สุด (ขนาดของ *hash_tab* จาก 10000 เป็น 1000, จำนวนบินของแต่ละมิติ จาก 1 เป็น 4) ซึ่งผลการทดลองแสดงไว้ดังตารางที่ 4.12 ถึง 4.14

ตารางที่ 4.12 แสดงผลการทดสอบอัลกอริธึม DBS บนชุดข้อมูล DS2

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.449	22088	0.2	8.4
2	0.493	40204	0.6	7.8
3	0.547	58348	0.8	9.0
4	0.535	76464	0.6	9.2
5	0.545	94580	1.0	10.4
6	0.585	112668	1.2	10.8
7	0.589	130812	1.6	8.6
8	0.613	148928	2.0	8.2
9	0.639	167044	2.6	11.8
10	0.635	185160	2.4	9.0
ข้อมูลทั้งหมด	-	-	17.0	11.0

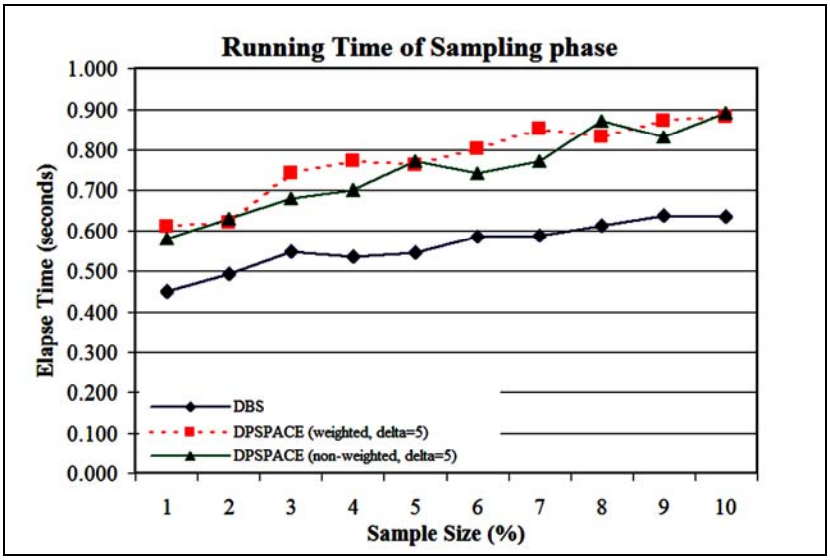
ตารางที่ 4.13 แสดงผลการทดสอบอัลกอริธึม DBSPACE $_{(weighted, delta=5)}$ บนชุดข้อมูล DS2

อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.610	43256	0.0	8.0
2	0.620	66548	0.0	9.0
3	0.741	89876	0.0	10.0
4	0.771	113168	0.0	7.0
5	0.761	136460	1.0	12.0
6	0.801	159716	1.0	13.0
7	0.851	183044	1.0	9.0
8	0.831	206336	2.0	9.0
9	0.871	229628	3.0	10.0
10	0.881	252920	2.0	10.0
ข้อมูลทั้งหมด	-	-	17.0	11.0

ตารางที่ 4.14 แสดงผลการทดสอบอัลกอริธึม DBSPACE $_{(non-weighted, delta=5)}$ บนชุดข้อมูล DS2

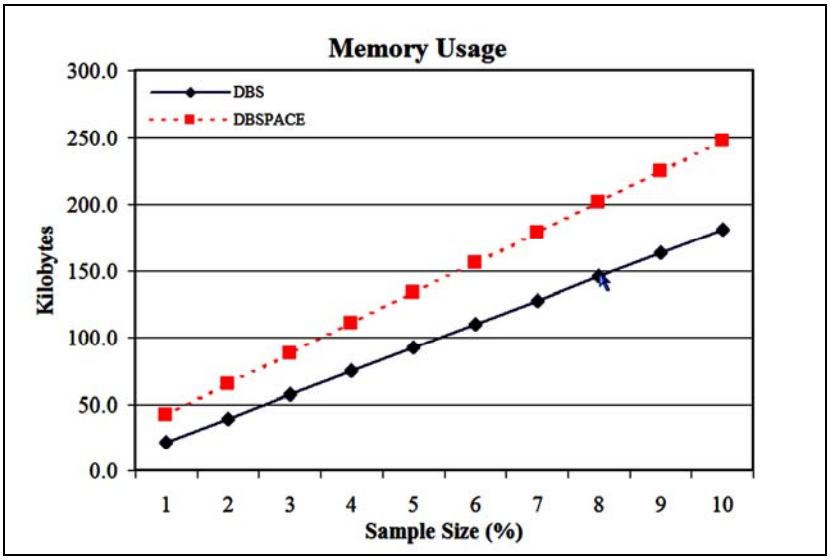
อัตราการสุ่ม (%)	เวลาที่ใช้ในการสุ่มข้อมูล (วินาที)	หน่วยความจำที่ใช้ในการสุ่มข้อมูล (ไบต์)	เวลาที่ใช้ในการค้นหาคลัสเตอร์ (วินาที)	*NC* (20)
1	0.580	43256	0.0	9.0
2	0.630	66548	0.0	7.0
3	0.680	89876	0.0	8.0
4	0.701	113168	0.0	10.0
5	0.771	136460	0.0	13.0
6	0.741	159716	1.0	9.0
7	0.771	183044	1.0	9.0
8	0.871	206336	2.0	8.0
9	0.831	229628	2.0	10.0
10	0.891	252920	3.0	8.0
ข้อมูลทั้งหมด	-	-	17.0	11.0

จากข้อมูลในตารางที่ 4.12 ถึง 4.14 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับเวลาที่ใช้ในการสุ่มเปรียบเทียบกันระหว่างแต่ละอัลกอริธึมจะได้ผลดังรูปที่ 4.14



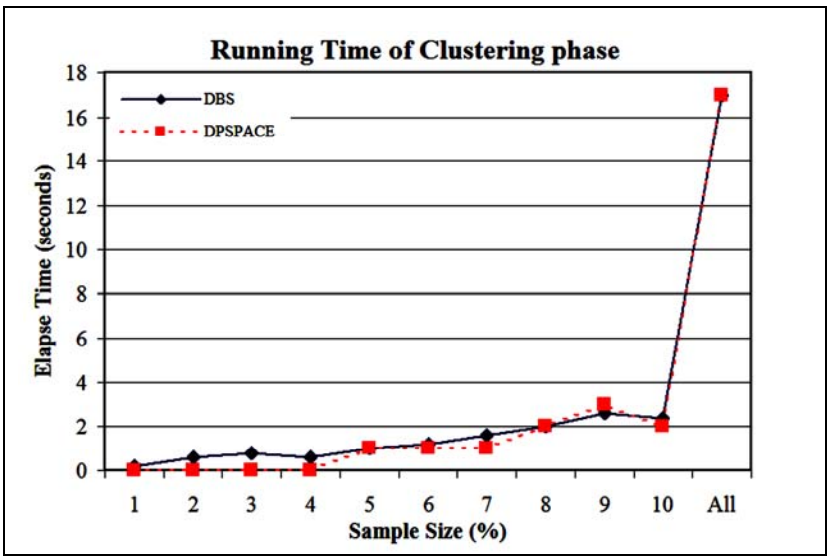
รูปที่ 4.14 เวลาที่ใช้ในการสุ่มข้อมูลของอัลกอริธึม DBS และ DBSPACE

จากข้อมูลในตารางที่ 4.12 ถึง 4.14 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับหน่วยความจำที่ใช้ในการสุ่มข้อมูลเปรียบเทียบกันระหว่างแต่ละอัลกอริธึมจะได้ผลดังรูปที่ 4.15



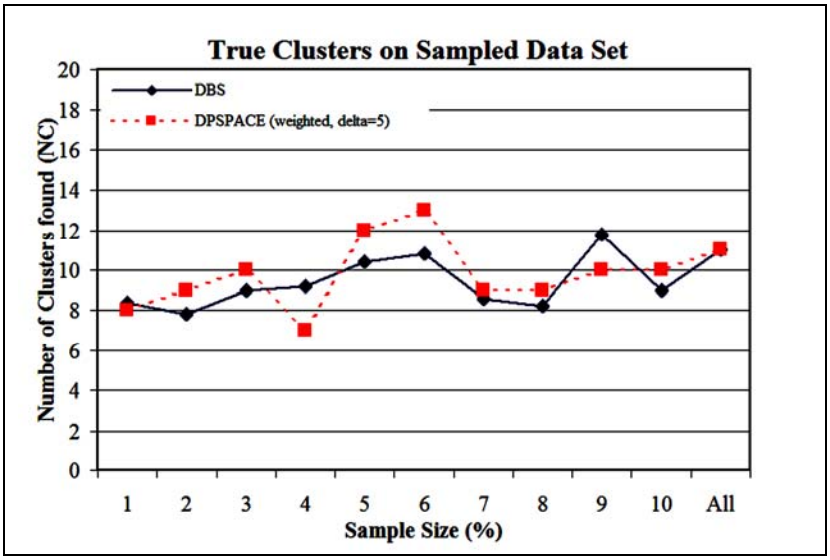
รูปที่ 4.15 หน่วยความจำที่ใช้ในการสุ่มข้อมูลของอัลกอริธึม DBS และ DBSPACE

จากข้อมูลในตารางที่ 4.12 ถึง 4.14 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับเวลาที่ใช้ในการค้นหาคลัสเตอร์ *k*-means เปรียบเทียบกันระหว่างแต่ละอัลกอริธึมจะได้ผลดังรูปที่ 4.16

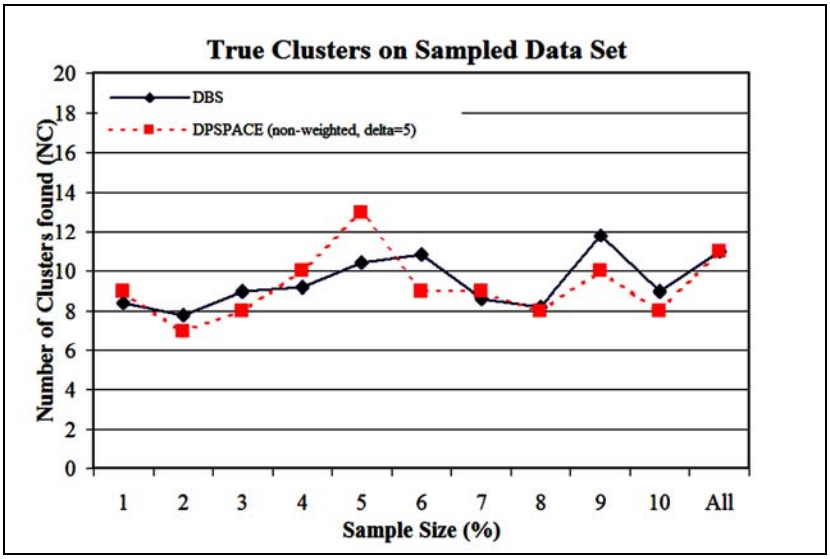


รูปที่ 4.16 เวลาที่ใช้ในการค้นหาคลัสเตอร์บนชุดข้อมูลสุ่มของอัลกอริธึม DBS และ DBSPACE

จากข้อมูลในตารางที่ 4.12 ถึง 4.14 เมื่อนำผลการทดลองที่ได้มาวิเคราะห์โดยการสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างอัตราการสุ่มกับจำนวนคลัสเตอร์แท้ที่สามารถพบได้จากการค้นหาด้วยอัลกอริธึม *k*-means เปรียบเทียบกันระหว่างแต่ละอัลกอริธึมจะได้ผลดังรูปที่ 4.17 และ 4.18



รูปที่ 4.17 จำนวน *NC* ที่พบได้จากชุดข้อมูลสุ่มของอัลกอริธึม DBS และ อัลกอริธึม DBSPACE $_{(\text{weighted, delta=5})}$



รูปที่ 4.18 จำนวน *NC* ที่พบได้จากชุดข้อมูลสุ่มของอัลกอริธึม DBS และ อัลกอริธึม DBSPACE $_{(\text{non-weighted, delta=5})}$

จากกราฟดังปรากฏในรูปที่ 4.14 แสดงให้เห็นว่าเมื่อเพิ่มอัตราการสุ่มข้อมูลมากขึ้น ทุกอัลกอริธึมจะใช้เวลาในการสุ่มเพิ่มมากขึ้น โดยที่อัลกอริธึม DBS จะใช้เวลาในการสุ่มข้อมูลน้อยที่สุด ในขณะที่อัลกอริธึม DBSPACE ทั้งแบบ weighted และ non-weighted ใช้เวลาในการสุ่มข้อมูล

มากกว่า DBS โดยที่อัลกอริธึม DBSPACE แบบ non-weighted มีแนวโน้มที่ใช้เวลาในการสุ่มน้อยกว่าแบบ weighted เล็กน้อย และจากกราฟดังปรากฏในรูปที่ 4.15 ยังแสดงให้เห็นว่าเมื่อเพิ่มอัตราการสุ่มข้อมูลมากขึ้น ทั้งอัลกอริธึม DBS และ DBSPACE จะใช้หน่วยความจำมากขึ้น ในขณะที่อัลกอริธึม DBSPACE จะใช้หน่วยความจำมากกว่าอัลกอริธึม DBS เพราะอัลกอริธึม DBSPACE ต้องใช้หน่วยความจำส่วนหนึ่งในการเก็บค่าต่างๆ เพื่อใช้ในการคำนวณค่า discordancy

เมื่อนำชุดข้อมูลสุ่มของอัลกอริธึม DBS และ DBSPACE มาผ่านเข้าสู่กระบวนการจัดกลุ่มข้อมูลอัตโนมัติเพื่อค้นหาคลัสเตอร์แท้ที่ยังคงสามารถพบได้บนชุดข้อมูลสุ่มเหล่านั้น ผลที่ได้ดังรูปที่ 4.17 และ 4.18 แสดงให้เห็นว่า ทั้งอัลกอริธึม DBS และ DBSPACE (ทั้งแบบ weighted และ non-weighted) ต่างให้ผลลัพธ์ของจำนวน *NC* ที่ใกล้เคียงกันซึ่งเป็นการยืนยันผลการทดลองที่ได้จากหัวข้อที่ 4.3 (รูปที่ 4.10) และยังใช้เวลาในการค้นหาคลัสเตอร์ใกล้เคียงกันอีกด้วย (ดังรูปที่ 4.16)

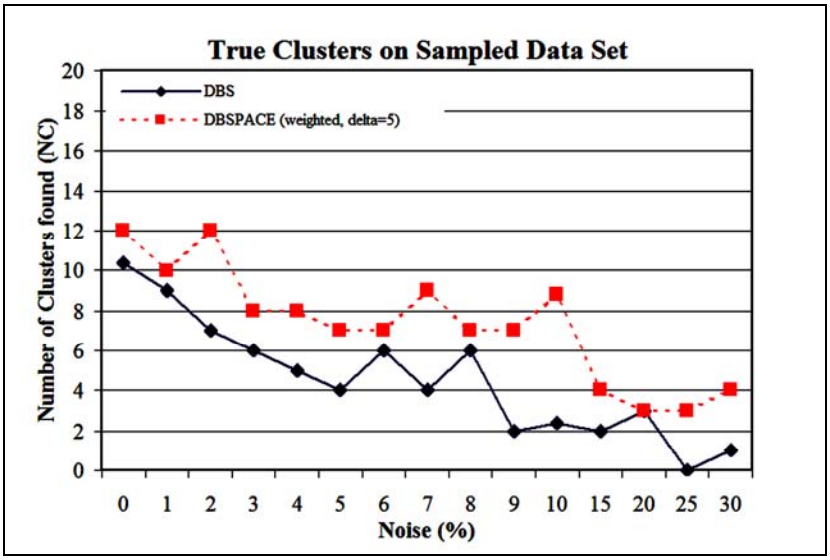
4.5 ผลการเปรียบเทียบความทนทานต่อข้อมูลรบกวนของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS

เนื้อหาในส่วนนี้เป็นการนำเสนอผลการทดลองเพื่อเปรียบเทียบคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึม DBSPACE กับอัลกอริธึม DBS ซึ่งดำเนินรูปแบบการทดลองคล้ายกับการทดลองในหัวข้อที่ 4.2 โดยในส่วนนี้จะทำการทดสอบอัลกอริธึม DBS และอัลกอริธึม DBSPACE ทั้งแบบ weighted และ non-weighted ด้วยค่า delta = 5 ซึ่งผลการทดลองแสดงไว้ดังตารางที่ 4.15 ซึ่งจะแสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBS และอัลกอริธึม DBSPACE ทั้งแบบ weighted และ non-weighted บนชุดข้อมูลตั้งต้น (DS2) ขนาด 2 มิติ 20 คลัสเตอร์ ที่ถูกสร้างข้อมูลรบกวนในปริมาณตั้งแต่ 1% จนถึง 30% ของจำนวนข้อมูลทั้งหมด

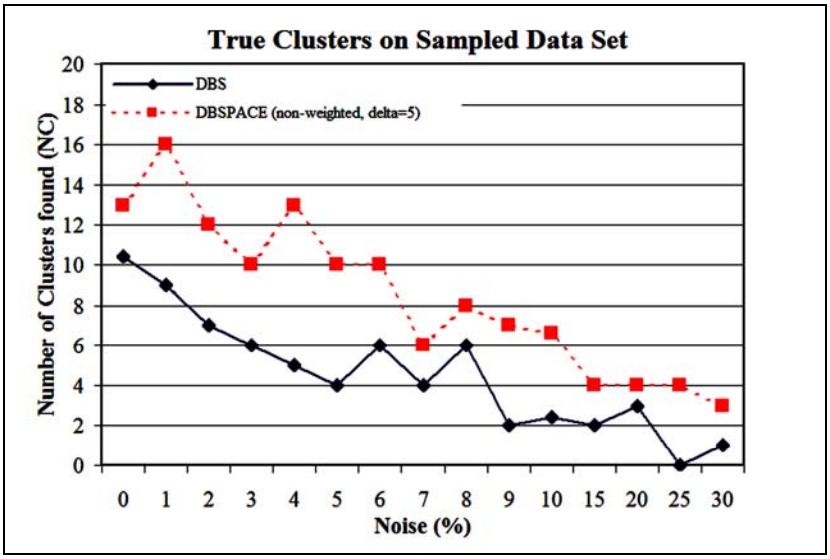
และเมื่อนำข้อมูลจากตารางที่ 4.15 มาสร้างเป็นกราฟแสดงความสัมพันธ์ระหว่างปริมาณข้อมูลรบกวนกับจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มเปรียบเทียบระหว่างอัลกอริธึม DBS กับอัลกอริธึม DBSPACE ทั้งแบบ weighted และ non-weighted จะได้ผลลัพธ์ดังกราฟที่แสดงไว้ดังรูปที่ 4.19 และ 4.20

ตารางที่ 4.15 แสดงจำนวน *NC* ที่พบบนชุดข้อมูลสุ่มขนาด 5% ของแต่ละอัลกอริธึมบนชุดข้อมูลตั้งต้นที่ถูกสร้างข้อมูลรบกวนในปริมาณตั้งแต่ 1% จนถึง 30%

อัลกอริธึมสุ่มข้อมูล / ปริมาณข้อมูลรบกวน (%)	DBS	DBSPACE (weighted, δ=5)	DBSPACE (non-weighted, δ=5)
0% (ข้อมูลปกติ)	10.4	12.0	13.0
1 %	9.0	10.0	16.0
2 %	7.0	12.0	12.0
3 %	6.0	8.0	10.0
4 %	5.0	8.0	13.0
5 %	4.0	7.0	10.0
6 %	6.0	7.0	10.0
7 %	4.0	9.0	6.0
8 %	6.0	7.0	8.0
9 %	2.0	7.0	7.0
10 %	2.4	8.8	6.6
15 %	2.0	4.0	4.0
20 %	3.0	3.0	4.0
25 %	0.0	3.0	4.0
30 %	1.0	4.0	3.0



รูปที่ 4.19 แสดงผลกระทบที่เกิดจากข้อมูลรบกวนบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBS กับอัลกอริธึม $DBSPACE_{(weighted,\ delta=5)}$



รูปที่ 4.20 แสดงผลกระทบที่เกิดจากข้อมูลรบกวนบนชุดข้อมูลสุ่มขนาด 5% ของอัลกอริธึม DBS กับอัลกอริธึม $DBSPACE_{(non\text{-}weighted,\ delta=5)}$

จากกราฟดังรูปที่ 4.19 และ 4.20 แสดงให้เห็นว่าเมื่อข้อมูลตั้งต้นมีข้อมูลรบกวนมากขึ้นจะส่งผลกระทบต่อคุณภาพของชุดข้อมูลสุ่มที่ได้จากทั้งสองอัลกอริธึม ดังแสดงได้จากจำนวน *NC* ที่พบมีจำนวนลดลงเมื่อมีข้อมูลรบกวนเพิ่มมากขึ้น โดยอัลกอริธึม DBSPACE ทั้งแบบ weighted

และ non-weighted จะได้ผลลัพธ์ของจำนวน *NC* ที่สูงกว่าอัลกอริธึม DBS อย่างเห็นได้ชัด ซึ่งเป็นการยืนยันผลการทดลองที่ได้จากหัวข้อที่ 4.3 (รูปที่ 4.11)

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

กระบวนการจัดกลุ่มข้อมูลอัตโนมัติเป็นกระบวนการทำเหมืองข้อมูลรูปแบบหนึ่งซึ่งมีวัตถุประสงค์เพื่อการอธิบายรูปแบบหรือลักษณะของความสัมพันธ์ของข้อมูล โดยการวิเคราะห์ลักษณะของข้อมูลที่มีความสอดคล้องกัน เพื่อรวบรวมและจัดข้อมูลเหล่านั้นให้เป็นหมวดหมู่ ข้อมูลที่มีความคล้ายคลึงกันจะถูกจัดให้อยู่ในกลุ่มเดียวกัน และข้อมูลที่อยู่ต่างกลุ่มกันจะต้องมีลักษณะที่แตกต่างกันมากที่สุด ซึ่งกระบวนการจัดกลุ่มข้อมูลอัตโนมัติจำเป็นจะต้องใช้เทคนิคที่ซับซ้อนในการค้นหากลุ่มข้อมูลเพื่อให้สามารถจัดกลุ่มข้อมูลได้อย่างถูกต้องและสมบูรณ์ กระบวนการจัดกลุ่มข้อมูลอัตโนมัติบนชุดข้อมูลที่มีขนาดใหญ่มากๆ เป็นกระบวนการที่ต้องใช้เวลาในการประมวลผลและสิ้นเปลืองหน่วยความจำเป็นจำนวนมาก ซึ่งอาจทำให้ไม่สามารถนำผลการวิเคราะห์ข้อมูลมาใช้ประโยชน์ได้ทันในเวลาที่ต้องการ จากปัญหาดังกล่าวจึงนำไปสู่คำถามที่ว่าจะทำอย่างไรเพื่อให้กระบวนการจัดกลุ่มข้อมูลอัตโนมัติบนชุดข้อมูลที่มีขนาดใหญ่มากๆ สามารถดำเนินไปจนเสร็จสิ้นได้ภายในเวลาและทรัพยากรที่มีอยู่อย่างจำกัด

การลดขนาดข้อมูลเป็นแนวทางหนึ่งที่จะช่วยแก้ปัญหานี้ได้ เนื่องจากข้อมูลแต่ละตัวมีความสำคัญต่องานการจัดกลุ่มข้อมูลไม่เท่ากัน ข้อมูลที่มีการรวมกลุ่มกันอย่างหนาแน่นจะเป็นข้อมูลที่มีความสำคัญต่อการจัดกลุ่มข้อมูลในอนาคต ในทางกลับกันข้อมูลที่มีการกระจายตัวต่างไปจากข้อมูลส่วนใหญ่จะต้องถูกขจัดออกไป เพราะข้อมูลเหล่านี้อาจเป็นข้อมูลรบกวนซึ่งไม่เพียงแต่จะทำให้กระบวนการจัดกลุ่มข้อมูลดำเนินไปได้ช้าเท่านั้น แต่ยังส่งผลกระทบต่อความแม่นตรงของผลลัพธ์ที่ต้องการอีกด้วย

งานวิจัยนี้มุ่งที่จะศึกษาค้นคว้า ตลอดจนพัฒนาเทคนิคการลดขนาดข้อมูลที่เหมาะสมเพื่องานการจัดกลุ่มข้อมูลขนาดใหญ่ โดยการพิจารณาการรวมกลุ่มกันของข้อมูลหรือความหนาแน่นของข้อมูลเป็นหลัก โดยข้อมูลที่ใช้จริงอาจมีการกระจายของข้อมูลแบบไม่ปกติ จึงทำให้ไม่สามารถใช้เทคนิคการลดขนาดข้อมูลแบบสม่ำเสมอทั่วไปได้ ฉะนั้นวัตถุประสงค์ของงานวิจัยนี้คือเพื่อพัฒนาเทคนิคการลดขนาดข้อมูลด้วยน้ำหนักความหนาแน่นที่มีความทนทานต่อข้อมูลรบกวนสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ที่มีการกระจายแบบไม่ปกติ โดยจะศึกษาเปรียบเทียบตลอดจนพัฒนาวิธีการลดขนาดข้อมูลด้วยเทคนิคการสุ่มข้อมูลเป็นหลัก เพื่อค้นหารูปแบบการสุ่มที่มีประสิทธิภาพ และสามารถสร้างชุดข้อมูลสุ่มที่สามารถเป็นตัวแทนชุดข้อมูลต้นฉบับได้อย่าง

สมบูรณ์ที่สุด โดยจะต้องสามารถสุ่มข้อมูลที่มีการรับเข้ามาอย่างต่อเนื่องได้โดยที่ไม่สามารถทราบจำนวนข้อมูลทั้งหมดล่วงหน้าก่อนการสุ่ม และจะต้องสามารถสร้างชุดข้อมูลสุ่มที่สมบูรณ์ได้ภายในการสแกนข้อมูลเพียงหนึ่งรอบเท่านั้น

ขั้นตอนการดำเนินงานวิจัยนี้จะแบ่งออกเป็นสองส่วนคือ การศึกษาและทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูลที่มีคุณสมบัติตรงตามวัตถุประสงค์ของงานวิจัยนี้ที่เคยมีนักวิจัยท่านอื่นเสนอไว้แล้ว เพื่อวิเคราะห์หาลักษณะเด่นของแต่ละอัลกอริธึมและใช้ประกอบการออกแบบอัลกอริธึมใหม่ที่มีคุณสมบัติดีขึ้น อัลกอริธึมสุ่มข้อมูลเหล่านั้นได้แก่ RVS, DBS, และ DBRVS โดยอัลกอริธึม RVS เป็นอัลกอริธึมสุ่มข้อมูลแบบต่อเนื่องที่เป็นที่รู้จัก ซึ่งถูกใช้ในกระบวนการจัดกลุ่มข้อมูลของอัลกอริธึมจัดกลุ่มข้อมูล CURE ส่วนอัลกอริธึม DBS เป็นอัลกอริธึมสุ่มข้อมูลที่พัฒนาขึ้นสำหรับงานการจัดกลุ่มข้อมูลโดยเฉพาะ ซึ่งสามารถสร้างชุดข้อมูลสุ่มจากการเบี่ยงเบนให้ค่าความน่าจะเป็นในการเลือกข้อมูลแปรผกผันกับค่าความหนาแน่นของข้อมูลบริเวณนั้น และอัลกอริธึม DBRVS ซึ่งเป็นอัลกอริธึมที่ผสมผสานเทคนิคของทั้ง RVS และ DBS เข้าด้วยกัน และเสริมเทคนิคที่ใช้กรองข้อมูลรบกวนที่อาจมีอยู่ในชุดข้อมูลตั้งต้น โดยทั้ง RVS, DBS, และ DBRVS สามารถสร้างชุดข้อมูลสุ่มที่สมบูรณ์ได้ภายในการสแกนข้อมูลเพียงรอบเดียว

การวิจัยในส่วนที่สองคือการพัฒนาและทดสอบประสิทธิภาพของอัลกอริธึมสุ่มข้อมูล DBSPACE ซึ่งเป็นอัลกอริธึมสุ่มข้อมูลที่พัฒนาขึ้นใหม่โดยผู้วิจัย และเปรียบเทียบผลลัพธ์ที่ได้กับอัลกอริธึมอื่นๆ โดยอัลกอริธึม DBSPACE ถูกพัฒนาขึ้นเพื่อเพิ่มประสิทธิภาพในการกรองข้อมูลรบกวนที่อาจมีอยู่ในชุดข้อมูลตั้งต้นโดยการเพิ่มเทคนิคเฉพาะซึ่งจะพิจารณาค่าความเป็นปึกแผ่น (compactness) และค่าความไม่ประสานกันของข้อมูล (discordancy) ประกอบกับค่าความหนาแน่นของข้อมูลในการสุ่มข้อมูลด้วย โดยมีสมมุติฐานที่ว่าข้อมูลที่มีความสำคัญต่อกระบวนการจัดกลุ่มข้อมูลและสามารถเป็นตัวแทนของกลุ่มข้อมูลที่ดีจะต้องเป็นข้อมูลที่มีค่าความเป็นปึกแผ่นสูง และในทางกลับกันข้อมูลรบกวนจะเป็นข้อมูลที่มีค่าความเป็นปึกแผ่นต่ำกว่า อันจะส่งผลกระทบต่อความแม่นตรงของกระบวนการจัดกลุ่มข้อมูลซึ่งจำเป็นต้องกรองข้อมูลเหล่านี้ไม่ให้ถูกเลือกเข้าไปในชุดข้อมูลสุ่ม

ในการวิจัยนี้จะทดสอบประสิทธิภาพของแต่ละอัลกอริธึมด้วยชุดข้อมูลสังเคราะห์ ซึ่งข้อมูลแต่ละตัวถูกจัดคลัสเตอร์เรียบร้อยแล้ว และใช้อัลกอริธึม SimpleKMeans ที่มีอยู่ในโปรแกรม WEKA ทำการค้นหาคลัสเตอร์จากชุดข้อมูลสุ่มของแต่ละอัลกอริธึม โดยจะสร้างชุดข้อมูลสุ่มด้วยอัตราการสุ่มต่างๆ เพื่อวิเคราะห์ผลลัพธ์จำนวนคลัสเตอร์ที่สามารถพบได้บนชุดข้อมูลสุ่มขนาดต่างๆ กัน โดยการเปรียบเทียบคลัสเตอร์แท้ที่ได้จากชุดข้อมูลตั้งต้นกับคลัสเตอร์ที่พบจากชุดข้อมูลสุ่ม และทำการศึกษาคุณสมบัติความทนทานต่อข้อมูลรบกวนของอัลกอริธึมนั้นๆ ด้วย โดยการ

สร้างข้อมูลรบกวนในปริมาณต่างๆ ลงบนชุดข้อมูลตั้งต้น จากนั้นจึงทำการสุ่มข้อมูลเหล่านั้นด้วยแต่ละอัลกอริธึมในอัตราการสุ่มคงที่ และวิเคราะห์ผลกระทบของปริมาณข้อมูลรบกวนที่มีต่อผลลัพธ์จำนวนคลัสเตอร์ที่สามารถพบได้บนชุดข้อมูลสุ่ม

5.1 สรุปผลการวิจัย

5.1.1 สรุปผลการเปรียบเทียบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS

1) อัลกอริธึม DBS เป็นอัลกอริธึมที่มีความแม่นตรงสูงที่สุด โดยจากผลการทดลองแสดงให้เห็นว่า การสุ่มข้อมูลด้วยอัลกอริธึม DBS ขนาด 2% สามารถให้ผลลัพธ์ของจำนวนคลัส-เตอร์ที่ถูกต้องได้เทียบเท่ากับการใช้ข้อมูลทั้งหมด ซึ่งมีส่วนช่วยลดเวลาที่ใช้ในกระบวนการจัดกลุ่มข้อมูลได้มากกว่า 95% (เมื่อเทียบกับการใช้ข้อมูลทั้งหมด) แต่อย่างไรก็ตามอัลกอริธึม DBS ต้องการหน่วยความจำเพื่อใช้ในกระบวนการสุ่มข้อมูลสูงกว่าอัลกอริธึมอื่น เนื่องจากอัลกอริธึม DBS ต้องใช้หน่วยความจำบางส่วนเพื่อเก็บตัวแปรไว้ใช้ในการคำนวณค่าความน่าจะเป็นในระหว่างการสุ่มข้อมูล อีกทั้งอัลกอริธึม DBS ยังใช้เวลาในการสุ่มข้อมูลนานกว่าทุกอัลกอริธึม แต่ถึงแม้อัลกอริธึม DBS จะใช้เวลาในการสุ่มข้อมูลนานกว่าอัลกอริธึมอื่น แต่เมื่อเทียบกับความถูกต้องที่ได้และเวลาที่สามารถลดลงได้ในกระบวนการจัดกลุ่มข้อมูล จึงถือว่าอัลกอริธึม DBS เป็นอัลกอริธึมสุ่มข้อมูลที่มีประสิทธิภาพสูงอัลกอริธึมหนึ่ง

2) อัลกอริธึม RVS เป็นอัลกอริธึมสุ่มข้อมูลที่ใช้ทรัพยากรน้อยที่สุด เพราะอัลกอริธึม RVS ใช้หน่วยความจำน้อยที่สุดและสามารถสร้างชุดข้อมูลสุ่มที่สมบูรณ์ได้รวดเร็วที่สุด แต่จำนวนคลัสเตอร์ที่สามารถพบได้จากชุดข้อมูลสุ่มมีแนวโน้มที่จะลดลงเมื่อเพิ่มอัตราการสุ่มถึงจุดๆ หนึ่ง เพราะเมื่ออัลกอริธึมตัดสินใจจะรับข้อมูลใหม่เข้ามาเก็บใน reservoir แล้ว อัลกอริธึมจะสุ่มข้อมูลเก่าทิ้งไปโดยไม่มีการพิจารณาเปรียบเทียบความสำคัญหรือความน่าจะเป็นที่ข้อมูลเก่าหรือข้อมูลใหม่จะถูกเลือกมากกว่ากัน ด้วยเหตุนี้ข้อมูลเก่าจึงอาจถูกแทนที่ด้วยข้อมูลใหม่เสมอ ซึ่งหากคลัสเตอร์ที่มีขนาดเล็กเรียงตัวอยู่ในข้อมูลส่วนต้นๆ อาจทำให้คลัสเตอร์ดังกล่าวไม่ปรากฏในชุดข้อมูลสุ่มได้

3) อัลกอริธึม DBRVS เป็นอัลกอริธึมสุ่มข้อมูลที่แสดงถึงคุณสมบัติการทนทานต่อข้อมูลรบกวนสูงสุด ถึงแม้จำนวนคลัสเตอร์ที่พบบนชุดข้อมูลสุ่มของอัลกอริธึม DBRVS จะมีค่าน้อยกว่าอัลกอริธึมอื่นก็ตาม แต่อัลกอริธึม DBRVS ยังคงรักษาคุณภาพของชุดข้อมูลสุ่มไว้ได้แม้ชุดข้อมูลตั้งต้นจะมีข้อมูลรบกวนปะปนในปริมาณมากก็ตาม สังเกตได้จากจำนวนคลัสเตอร์ที่พบบนชุดข้อมูลสุ่มสามารถคงค่าเดิมไว้ได้แม้จะมีข้อมูลรบกวนปะปนมาถึง 20% ในขณะที่อัลกอริธึม DBS เป็นอัลกอริธึมที่มีความอ่อนไหวต่อข้อมูลรบกวนมากที่สุด

5.1.2 สรุปผลการทดสอบประสิทธิภาพของอัลกอริธึม DBSPACE

อัลกอริธึม DBSPACE ถูกพัฒนาขึ้นต่อยอดจากอัลกอริธึม DBS โดยมีจุดมุ่งหมายเพื่อให้สามารถสร้างชุดข้อมูลสุ่มที่คงลักษณะสำคัญของกลุ่มข้อมูลได้อย่างแม่นยำ อีกทั้งยังสามารถทำงานได้ดีแม้ข้อมูลที่ใช้จะมีข้อมูลรบกวนปะปนอยู่ โดยเพิ่มเทคนิคที่ใช้ประกอบการพิจารณาในการเลือกข้อมูล ซึ่งจะพิจารณาค่าความเป็นปึกแผ่นของข้อมูลร่วมกับความหนาแน่นของข้อมูลบริเวณนั้นด้วย โดยค่า δ (delta) เป็นค่ากำลังที่ใช้ปรับความสำคัญของค่าความเป็นปึกแผ่นของข้อมูลต่อการคำนวณค่าความน่าจะเป็นที่ข้อมูลจะถูกเลือก จากผลการทดลองพบว่า

1) ในกรณีที่ข้อมูลปราศจากข้อมูลรบกวน อัลกอริธึม DBSPACE สามารถสร้างชุดข้อมูลสุ่มที่มีคุณภาพดีเทียบเท่ากับอัลกอริธึม DBS ในการทดลองปรับค่า delta ตั้งแต่ 0 ถึง 8 แล้ววิเคราะห์หาจำนวน *NC* จากชุดข้อมูลสุ่ม ปรากฏว่าจำนวน *NC* ที่ได้จากอัลกอริธึม DBSPACE (ทั้งแบบ weighted discordancy และ non-weighted discordancy) มีจำนวนใกล้เคียงกันกับ DBS (delta = 0)

2) ในกรณีที่ข้อมูลมีข้อมูลรบกวนปะปนอยู่ อัลกอริธึม DBSPACE สามารถสร้างชุดข้อมูลสุ่มที่มีคุณภาพสูงกว่าอัลกอริธึม DBS ในการทดลองปรับค่า delta ตั้งแต่ 0 ถึง 8 แล้ววิเคราะห์หาจำนวน *NC* จากชุดข้อมูลสุ่ม ปรากฏว่าเมื่อปรับค่าพารามิเตอร์ delta เพิ่มขึ้น จะทำให้ข้อมูลสุ่มมีการรวมกลุ่มกันและมีความเป็นปึกแผ่นสูงขึ้น และยังมีผลให้ *k*-means สามารถพบจำนวน *NC* ได้มากขึ้น โดยที่อัลกอริธึม DBSPACE ทั้งแบบ weighted discordancy และ non-weighted discordancy สามารถให้ผลลัพธ์ค่อนข้างใกล้เคียงกัน

3) เมื่อพิจารณาเวลาที่ใช้ในการสุ่มข้อมูลของอัลกอริธึม DBSPACE พบว่าอัลกอริธึม DBSPACE ใช้เวลาในการสุ่มข้อมูลสูงกว่าอัลกอริธึม DBS โดยที่อัลกอริธึม DBSPACE แบบ non-weighted มีแนวโน้มที่ใช้เวลาในการสุ่มน้อยกว่าแบบ weighted เล็กน้อย ดังผลการทดลองพบว่า อัลกอริธึม DBSPACE แบบ non-weighted ใช้เวลาในการสุ่มข้อมูลโดยเฉลี่ยสูงกว่าอัลกอริธึม DBS ประมาณ 32.5% ในขณะที่อัลกอริธึม DBSPACE แบบ weighted ใช้เวลาในการสุ่มข้อมูลโดยเฉลี่ยสูงกว่าอัลกอริธึม DBS ประมาณ 37.4%

4) เมื่อพิจารณาหน่วยความจำที่ใช้ในระหว่างการสุ่มข้อมูลของอัลกอริธึมสุ่มข้อมูล DBSPACE พบว่าอัลกอริธึม DBSPACE ใช้หน่วยความจำสูงกว่าอัลกอริธึม DBS โดยอัลกอริธึม DBSPACE จะใช้หน่วยความจำส่วนหนึ่งเพื่อเก็บค่าตัวแปรของกลุ่มข้อมูลย่อยเพื่อใช้ในการคำนวณค่าความเป็นปึกแผ่นประกอบการสุ่มข้อมูล

5) เมื่อพิจารณาเวลาที่ใช้ในระหว่างกระบวนการจัดกลุ่มข้อมูลจากชุดข้อมูลสุ่มของอัลกอริธึม DBSPACE และอัลกอริธึม DBS พบว่า *k*-means ใช้เวลาใกล้เคียงกันในการจัดกลุ่มข้อ

มูลบนชุดข้อมูลสุ่มของทั้งสองอัลกอริธึม

จากข้อสรุปที่ได้กล่าวไปทั้งหมดแล้วนั้น จะเห็นได้ว่าแต่ละอัลกอริธึมต่างมีข้อดีและข้อเสียที่แตกต่างกันออกไป แต่เมื่อพิจารณาถึงวัตถุประสงค์หลักของการลดขนาดข้อมูลด้วยการสุ่มข้อมูลแล้ว ความถูกต้องและความครบถ้วนของกลุ่มข้อมูลบนชุดข้อมูลสุ่มจึงเป็นสิ่งสำคัญที่สุด รองลงมาคือเวลาและหน่วยความจำที่ใช้ในระหว่างการสุ่มข้อมูล จากการศึกษาและทดสอบประสิทธิภาพของอัลกอริธึม RVS, DBS, และ DBRVS ที่เคยมีนักวิจัยท่านอื่นเสนอไว้แล้ว พบว่าอัลกอริธึม DBS เป็นอัลกอริธึมที่ดีที่สุดในด้านที่สามารถเลือกข้อมูลที่เป็นตัวแทนของข้อมูลส่วนใหญ่ได้อย่างแม่นยำ แต่อัลกอริธึม DBS กลับมีความอ่อนไหวเมื่อข้อมูลที่ใช้ปะปนไปด้วยข้อมูลรบกวน ด้วยเหตุดังกล่าว อัลกอริธึม DBSPACE จึงถูกออกแบบขึ้นเพื่อเพิ่มศักยภาพในการทนทานต่อข้อมูลรบกวน โดยการพิจารณาค่าความเป็นปึกแผ่นของข้อมูล ซึ่งบริเวณที่มีความน่าจะเป็นที่จะเป็นกลุ่มข้อมูลที่สนใจจะมีค่าความเป็นปึกแผ่นของข้อมูลสูงกว่าบริเวณที่มีความน่าจะเป็นที่จะเป็นข้อมูลรบกวน จากผลการทดลองพบว่า อัลกอริธึม DBSPACE สามารถให้ผลลัพธ์ที่ดีเทียบเท่ากับอัลกอริธึม DBS ในกรณีที่ข้อมูลปราศจากข้อมูลรบกวน และอัลกอริธึม DBSPACE สามารถให้ผลลัพธ์ที่ดีกว่าในกรณีที่ข้อมูลมีข้อมูลรบกวนปะปนอยู่

จึงอาจกล่าวได้ว่าอัลกอริธึม DBSPACE เป็นอัลกอริธึมสุ่มข้อมูลด้วยน้ำหนักความหนาแน่นและความเป็นปึกแผ่นของข้อมูลที่มีความทนทานต่อข้อมูลรบกวน ซึ่งพัฒนาขึ้นสำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ที่มีการกระจายของข้อมูลแบบไม่ปกติ

5.2 การประยุกต์ผลการวิจัย

อัลกอริธึม DBSPACE สามารถนำมาประยุกต์ใช้ในงานทางด้านวิศวกรรมข้อมูล เพื่อใช้ในการเพิ่มประสิทธิภาพในกระบวนการจัดกลุ่มข้อมูลอัตโนมัติ ในกรณีที่ข้อมูลมีปริมาณมาก โดยสามารถนำมาใช้เพื่อลดขนาดข้อมูลและเพื่อเป็นการเตรียมข้อมูลให้มีขนาดและคุณรูปที่เหมาะสมต่องานการจัดกลุ่มข้อมูล ซึ่งสามารถนำอัลกอริธึม DBSPACE มาประยุกต์ใช้งานในกรณีต่างๆ เช่น

5.2.1 เมื่อต้องการวิเคราะห์ข้อมูลที่มีปริมาณมากๆ และไม่สามารถนำข้อมูลทั้งหมดมาวิเคราะห์ได้ เพราะการนำข้อมูลทั้งหมดมาวิเคราะห์จำเป็นจะต้องใช้ทั้งเวลาในการประมวลผลและหน่วยความจำเป็นจำนวนมาก ซึ่งอาจทำให้ไม่สามารถนำผลลัพธ์มาใช้ประโยชน์ได้ทันในเวลาที่จำกัด

5.2.2 เมื่อข้อมูลมีการกระจายตัวแบบไม่สม่ำเสมอ เช่น ข้อมูลที่ใช้ในงานทางภูมิศาสตร์ (GIS) ข้อมูลภาพถ่ายทางดาวเทียม หรือในกรณีที่ข้อมูลมีลักษณะที่เกี่ยวข้องกับระยะหรือพิกัดของวัตถุต่างๆ (spatial data) ซึ่งข้อมูลจะถูกเก็บรวบรวมไว้ในปริมาณมาก

5.2.3 เมื่อข้อมูลที่ต้องการนำมาวิเคราะห์คาดว่าจะมีเอ้าท์ไลเออร์หรือข้อมูลรบกวนปะปนอยู่ ในกรณีนี้ควรปรับค่าพารามิเตอร์ delta ให้มีค่าสูงขึ้นตามจำนวนของข้อมูลรบกวนที่ปะปนอยู่

5.2.4 เมื่อต้องการจัดกลุ่มข้อมูลในเวลาอันรวดเร็ว เพื่อนำผลเบื้องต้นที่ได้มาวิเคราะห์และปรับปรุงกระบวนการจัดกลุ่มข้อมูลในรอบต่อไปให้มีความถูกต้องสูงขึ้น

5.3 ข้อเสนอแนะในการวิจัยต่อไป

สำหรับงานการจัดกลุ่มข้อมูลขนาดใหญ่ การลดขนาดข้อมูลเป็นเทคนิคที่มีความสำคัญยิ่งและเป็นที่ทราบดีถึงปัญหาของการจัดกลุ่มของข้อมูลที่มีขนาดใหญ่มากๆ ดังนั้นเทคนิคการลดขนาดข้อมูลโดยการสุ่มข้อมูลด้วยอัลกอริธึม DBSPACE จึงน่าจะเป็นประโยชน์แก่นักวิจัยท่านอื่นที่สนใจที่จะพัฒนาองค์ความรู้เพื่องานการจัดกลุ่มข้อมูลต่อไป โดยแนวทางวิจัยที่จะพัฒนาอัลกอริธึม DBSPACE สามารถทำได้หลายแนวทางด้วยกัน อาทิ เช่น

5.3.1 การวิเคราะห์เพื่อหาแนวทางในการกำหนดค่า delta ที่เหมาะสมสำหรับข้อมูลที่ใช้ในงานการจัดกลุ่มข้อมูล โดยมีจุดประสงค์เพื่อให้อัลกอริธึม DBSPACE สามารถสร้างชุดข้อมูลสุ่มที่คงลักษณะของกลุ่มข้อมูลไว้ได้มากที่สุด

5.3.2 การวิจัยเพื่อเพิ่มประสิทธิภาพของอัลกอริธึม DBSPACE โดยการปรับปรุงเทคนิคการจัดเก็บข้อมูลในหน่วยความจำเพื่อเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล และเพื่อให้อัลกอริธึมสามารถทำงานได้รวดเร็วขึ้นบนหน่วยความจำอันจำกัด

5.3.3 การพิจารณาเลือกใช้เทคนิคที่แม่นยำสำหรับการค้นหาตัวแทนของแต่ละกลุ่มข้อมูลย่อยที่มีผลกระทบน้อยที่สุดเมื่อชุดข้อมูลที่ต้องการนำมาวิเคราะห์มีข้อมูลรบกวนปะปนอยู่จำนวนมาก ซึ่งปัจจุบัน DBSPACE ใช้เซนทรอยด์เป็นตัวแทนของแต่ละกลุ่มข้อมูลย่อย และจะนำเซนทรอยด์นี้มาใช้สำหรับการคำนวณค่า discordancy ต่อไป โดยหากข้อมูลที่ต้องการนำมาวิเคราะห์มีข้อมูลรบกวนปะปนอยู่ อาจทำให้เซนทรอยด์ที่ได้ผิดเพี้ยนไปจากที่ควรจะเป็น และยังอาจส่งผลให้การคำนวณค่า discordancy ผิดพลาดได้ และยังอาจส่งผลกระทบต่อความแม่นตรงในการเลือกข้อมูลของอัลกอริธึมสุ่มข้อมูลอีกด้วย

5.3.4 การนำเอาเทคนิคที่ใช้สุ่มข้อมูลของอัลกอริธึม DBSPACE ไปประยุกต์ใช้ในขั้นตอนการจัดกลุ่มข้อมูล โดยอาจนำเอาเทคนิคดังกล่าวไปทำงานผสานกับอัลกอริธึมจัดกลุ่มข้อมูลที่มีอยู่เพื่อให้อัลกอริธึมนั้นสามารถรองรับข้อมูลได้มากขึ้น และมีความเป็นอัตโนมัติมากขึ้น ซึ่งเป็นแนวทางในการเพิ่มประสิทธิภาพของอัลกอริธึมนั้นได้อีกแนวทางหนึ่ง

5.3.5 การพัฒนาอัลกอริธึม DBSPACE ไปสู่อัลกอริธึมจัดกลุ่มข้อมูลอัตโนมัติ โดยอาจมีลักษณะคล้ายกับอัลกอริธึม CURE ซึ่งผสานเทคนิคการสุ่มข้อมูลเข้าด้วยกัน และใช้ตัวแทนของ

กลุ่มข้อมูลย่อยเป็นตัวแทนย่อยๆ ของคลัสเตอร์ โดยแต่ละคลัสเตอร์สามารถมีตัวแทนได้มากกว่าหนึ่งตัว ซึ่งจะทำให้อัลกอริธึมสามารถค้นหาคลัสเตอร์ที่มีรูปร่างที่ซับซ้อนได้บนข้อมูลขนาดใหญ่ที่มีการกระจายแบบไม่ปกติ

รายการอ้างอิง

Breunig, M. M., H.-P. Kriegel, R. T. Ng and J. Sander (2000). LOF: Identifying density-based local outliers. In **Proc. ACM SIGMOD 2000 Int. Conf. on Management of Data**, Dalles.

Ester, M., H.-P. Kriegel, J. Sander and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In **Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96)**, Porland.

Frank, E., M. Hall, G. Holmes, B. Martin, M. Mayo, B. Pfahringer, T. Smith and I. Witten (2005). **Waikato Environment for Knowledge Analysis: Weka-3-4-7**. University of Waikato: New Zealand.

Guha, S., R. Rastogi and K. Shim (1998). CURE: An efficient clustering algorithm for large databases. In **Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data**.

Guha, S., R. Rastogi and K. Shim (1999). ROCK: A robus clustering algorithm for categorical attributes. In **Proc. of the 15th Int'l Conf. on Data Engineering**.

Han, J. and M. Kamber (2001). **Data mining: Concepts and techniques**. San Diego: Academic Press.

Handl, J. and J. Knowles (n.d.). **Cluster generators for large high-dimensional data sets with large numbers of clusters**. The University of Manchester.

Karypis, G., E.-H. S. Han and V. Kumar (1999). CHAMELEON: A hierarchycal clustering algorithm using dynamic modeling. In the **IEEE computer: Spatial issue on data analysis and mining**.

Kaufman, L. and P. J. Rousseeuw (1989). **Finding groups in data: An introduction to cluster analysis**: John Wiley and Sons.

Kerdprasop, K., N. Kerdprasop and J. Sun (2005). Density biased reservoir sampling for clustering. In **Proc. of IASTED Int. Conf. on Artificial Intelligence and Applications**, Austria.

Kollios, G., D. Gunopulos, N. Koudas and S. Berchtold (2003). Efficient biased sampling for approximate clustering and outlier detection in large data sets. In the **IEEE Transaction on Knowledge and Data Engineering**.

Motwani, R. and P. Raghavan (1995). **Randomized algorithms**: Cambridge University Press.

Ng, R. T. and J. Han (1994). Efficient and effective clustering methods for spatial data mining. In **Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)**, Santiago.

Olken, F. (1993). **Random sampling from databases**. Ph.D. Dissertation, University of California at Berkeley, California.

Palmer, C. R. and C. Faloutsos (2000). Density biased sampling: An improved method for data mining and clustering. In **ACM-SIGMOD Int. Conf. on Management of Data**.

Thianniwet, T., K. Kerdprasop and N. Kerdprasop (2005). The sampling methods for determining clusters in large data sets. In **Proc. of the 31th Congress on Science and technology of Thailand**, Suranaree University of Technology, Thailand.

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

Zhang, T., R. Ramakrishnan and M. Livny (1996). BIRCH: An efficient data clustering method for very large databases. In **ACM SIGMOD Int. Conf. on Management of Data**, Montreal, Canada.

ภาคผนวก ก

บทความผลงานวิจัยที่นำเสนอในการประชุมวิชาการวิทยาศาสตร์และเทคโนโลยีแห่งประเทศไทย ครั้งที่ 31

เทคนิคการสุ่มข้อมูลเพื่องานการจัดกลุ่มข้อมูลขนาดใหญ่

THE SAMPLING METHODS FOR DETERMINING CLUSTERS IN LARGE DATA SETS

ธรรมศักดิ์ เธียรนิเวศน์, กิตติศักดิ์ เกิดประสพ และ นิตยา เกิดประสพ

Thammasak Thianniwet, Kittisak Kerdprasop and Nittaya Kerdprasop
Data Engineering and Knowledge Discovery (DEKD) Research Unit, School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasima, Thailand
E-mail address: thammasak_th@hotmail.com, kerdpras@ccs.sut.ac.th, nittaya@ccs.sut.ac.th

บทคัดย่อ: กระบวนการจัดกลุ่มข้อมูลอัตโนมัติบนชุดข้อมูลที่มีขนาดใหญ่มากๆ เป็นกระบวนการที่ต้องใช้เวลาและสิ้นเปลืองหน่วยความจำเป็นจำนวนมาก การลดขนาดข้อมูลเป็นแนวทางหนึ่งที่จะช่วยแก้ปัญหานี้ได้ สำหรับงานวิจัยนี้จะศึกษาและเปรียบเทียบวิธีการลดขนาดข้อมูลด้วยเทคนิคการสุ่มที่เหมาะสมสำหรับงานการจัดกลุ่มข้อมูล เพื่อค้นหารูปแบบการสุ่มที่มีประสิทธิภาพและสามารถสร้างชุดข้อมูลสุ่มที่สามารถเป็นตัวแทนชุดข้อมูลต้นฉบับได้อย่างสมบูรณ์ที่สุด จากการศึกษาพบว่า การสุ่มข้อมูลด้วยเทคนิคการสุ่มแบบเบี่ยงเบนตามความหนาแน่นเพียง 2% สามารถให้ผลลัพธ์ของกลุ่มข้อมูลได้เทียบเท่ากับข้อมูลทั้งหมด อีกทั้งยังสามารถลดเวลาในการจัดกลุ่มข้อมูลได้มากกว่า 95% และจากผลการทดลองยังพบว่าเทคนิคการสุ่มแบบเบี่ยงเบนตามความหนาแน่นแบบสะสมสามารถสร้างชุดข้อมูลสุ่มที่มีคุณสมบัติของการทนทานต่อข้อมูลรบกวนบนชุดข้อมูลตั้งต้นได้อีกด้วย

Abstract: Determining clusters in large data sets takes a very long time and consumes many resources. Data reduction is an important step to increase the efficiency of determining clusters in large data sets. Our work is intended to examine the appropriate sampling techniques as a data reduction scheme for clustering which require only a single data set scan. A good sample of the data set shall be a good substitute for the original data set while also keeping as much important cluster information as possible. Our experiments show that 2% of density-biased sampling (DBS) of the original data set can group clusters as well as clustering on the whole original data set and also help reduce time to cluster by over 95%. And the sampled data sets with density-biased reservoir sampling (DBRVS) technique report a noise tolerance property while the original data set is surrounded by many noises.

Introduction: Clustering in data mining is the process of discovering the clusters in a set of data, by maximizing the intra-cluster similarity and minimizing the inter-cluster similarity between clusters ([1], [3]). An efficient clustering method needs many difficult and complicated techniques to find the correct clusters from a very large data set. Clustering on large data sets is time and resource consuming ([3]). Many researchers have proposed sampling techniques to efficiently reduce the size of the data set, while keeping all important cluster information as much as possible. In this paper, we study three sampling techniques which can draw the samples by only a single scan over the data.

Random Sampling with a Reservoir (called RVS) ([1]) is the sampling technique which selects a random sample of size n from a data set of size N in a single data set pass

within $O(n(1+\log N/n))$ expected time, where the size of the data set is unknown prior to sampling. The process is started by initializing first n objects to the reservoir output buffer of size n. And then, randomly select a number of objects, k, to be skipped in the main data set to select a new sample object. When the object is selected, it becomes a candidate and randomly replaces one object in the reservoir buffer. This step is repeated until the end of file has been reached. Finally, all objects in the reservoir buffer of size n become a final sampled data set.

Density Biased Sampling (called DBS) ([2]) is the sampling technique that is proposed to take into account the sampling over a data set which follows the Zipf's distribution. The sampling process is started by partitioning data into groups using a hashing function. And the biasing process is to draw an object by considering the reverse density of its group. It probabilistically over-samples sparse regions and under-samples dense regions. With this biased sampling, small clusters will not be missed.

Density Biased Reservoir Sampling (called DBRVS) ([3]) is the adapted sampling technique which combines the density biased scheme together with the reservoir scheme. The sampling process is started by partitioning the data space into a finite number of equiwidth bins in the quantized space. Then apply random sampling with a reservoir scheme to the series of binning groups. The biased reservoir sampling draws a group by the consideration of two consecutive binning groups. The denser group is a candidate to be included in the sample if the density difference of two groups is above some threshold δ or the sum of the density on both groups is above the threshold ε.

Methodology: In our experiments, we studied both efficiency and effectiveness of each sampling technique with some synthetic data sets. We monitored sampling time and memory usage on various sizes of samples. Then we determined the clusters for the sampled data sets using k-means (the partitioning clustering algorithm) and monitor time to discover clusters. After finishing the clustering process, we compare all found clusters with the original clusters and calculate the value of *Number of Clusters found* (NC), which is defined in [2].

Results, Discussion and Conclusion: From the experimental results, we found that DBS is the most accurate sampling technique. It shows that a 2% DBS sample of the original data set can produce the same result as the whole original data set as show in Figure 3, and also help reduce time to find the clusters by over 95%. On the other hand, it requires more memory and takes longer time of sampling process than the others (Figures 1 and 2). RVS is the most resource saving technique. It consumes a small amount of memory and runs faster than the others, but its NC tends to decrease after the sample size has reached some value (Figure 3).

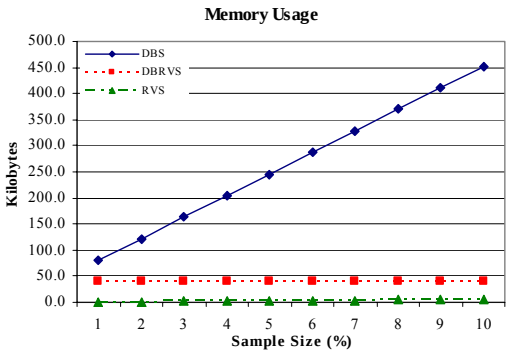


Figure 1: Memory usage (kb) for each sampling technique.

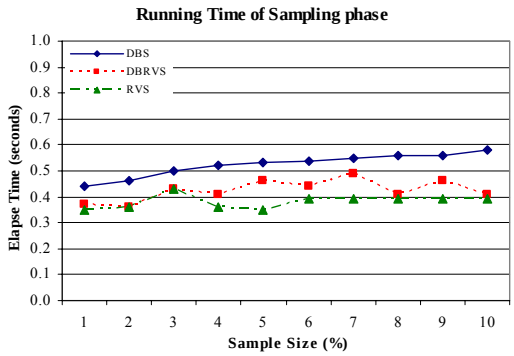


Figure 2: Running time of sampling phase.

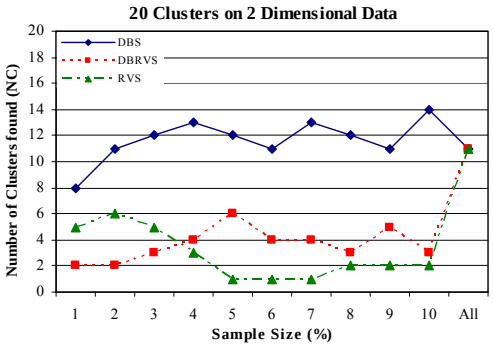


Figure 3: Number of clusters found on sampled data sets.

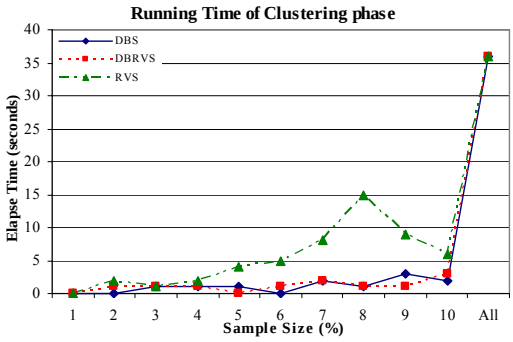


Figure 4: Running time of clustering phase.

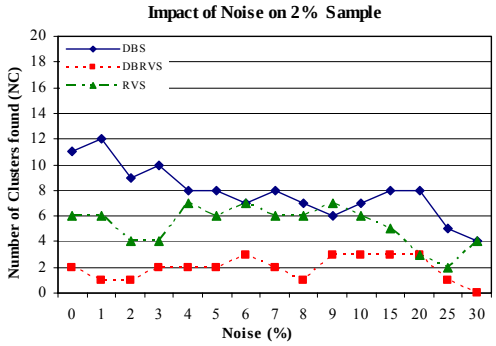


Figure 5: The Impact of noise on 2% samples.

Figure 5 show that DBRVS has a noise tolerance property. It has the least impact when many noises occurred although its NC is less than satisfactory, while DBS is very sensitive to noises. Our future research is to extend our study and to design such a sampling technique that computes densities accurately and efficiently and also is less sensitive to noise.

Acknowledgement: We would like to thank C. Palmer for putting his density biased sampling source code on the Web. And thank J. Handl for sharing his Cluster Generator and some synthetic data sets. This research has been supported by the Data Engineering and Knowledge Discovery Research Unit (DEKD), which is fully supported by the Suranaree University of Technology.

References: [1] J. S. Vitter, "Random Sampling with a Reservoir," *ACM Transactions on Mathematical Software*, vol. 11, pp. 37-57, 1985.

[2] C. R. Palmer and C. Faloutsos, "Density Biased Sampling: An Improved Method for Data Mining and Clustering," *ACM-SIGMOD Int. Conf. on Management of Data*, pp. 82-92, 2000.

[3] K. Kerdprasop, N. Kerdprasop, and J. Sun, "Density Biased Reservoir Sampling for Clustering," *Proc. of IASTED Int. Conf. on Artificial Intelligence and Applications*, pp. 95-100, 2005.

Keywords: Data mining, Data reduction, Sampling, Clustering, Density biased, Reservoir

ภาคผนวก ข

รหัสต้นฉบับภาษาซี ของอัลกอริธึม DBSPACE

```
// DBSPACE.cpp : Defines the entry point for the console application.
//
/*********************************************************************
   This source code is (c) Copyright 2006 by Thammasak Thainniwet.
   It may be freely redistributed and included in other packages
   provided this copyright notice is included.

  A Density-Biased Sampling using Partial Approximate Compactness
Estimator: DBSPACE v1.0
      This is the algorithm to draw the sample from any data sets
which is biasing by
      number of object, weighted discordancy(d1) or non-weighted
discordancy(d2).

*********************************************************************/


// #include "stdafx.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include <assert.h>
#include <limits.h>
#include <time.h>
#include <sys/timeb.h>
#include <dos.h>

#define BUFFER_FACTOR   1.1   // How many points do we want to keep
#define Dimension       50    // Maximum dimension allowed
#define ALPHA           65599
#define PRINT_GROUPS    0
#define USE_RANDOM      1


typedef struct hashS {
    int     *v;
    int  n;
    struct hashS *next;
} tab_t;

typedef double DOT;
typedef struct {
      DOT               *ls;  // Linear Sum of object vectors
                              // [on all attribute]
      double            lsd;  // D1: Linear Sum of weighted distance
                              // between object and centroid of other
                              // objects
                              // D2: Linear Sum of distance between
                              // object and centroid of other objects
      unsigned long     n;    // Number of object in the cell
} CLS;
```

```
TYtab_t ** tab;
int tabsize = 0;
int ntab = 0;

typedef unsigned long hash_t;

int d;
int quanta;

// ************************************************************ //
// Procedures and Functions
//

int quant(double d, int n)
{
    int q = 0;

    if (d < 0) d = 0;
    if (d > 1) d = 1;

    n = n / 2;
    while (n) {
        if (d < .5) d = 2*d;
        else {
            q += n;
            d = (d - .5) * 2;
        }
        n = n/2;
    }
    return q;
}

int ahash(int *v, int H)
{
    int i;
    unsigned long h = 0;

    for (i = 0; i < d; i++) {
      h = ((unsigned long) v[i]) + h*((unsigned long) ALPHA);
    }
    return h%H;
}

int newhash(int *v, int H)
{
    int i, j;
    tab_t **tt = &tab[ahash(v, H)];
    for (i = 0; *tt; i++, tt = &(*tt)->next) {
            for (j = 0; j < d; j++)
                  if ((*tt)->v[j] != v[j])
                        break;
            if (j >= d)
                  break;
    }

    if (! *tt) {
#if PRINT_GROUPS > 1
            printf( "ng %d", ntab);
```

```
            for (j = 0; j < d; j++)
                  printf( " v[%d]=%d", j, v[j]);
            printf( "\n");
#endif
            *tt = (tab_t *)malloc(sizeof(**tt));
            (*tt)->v = (int *)malloc(sizeof((*tt)->v)*d);
            for (j = 0; j < d; j++)
                  (*tt)->v[j] = v[j];
            (*tt)->n = ntab;
            (*tt)->next = NULL;
            ntab++;

            if (ntab >= H) {
                  printf( "ERROR: Hash table is too small!\n");
                  exit(1);
            }
    }

    assert ((*tt)->n < ntab);

    return (*tt)->n;
}

int quanthash(double *v, int *res, int H)
{

    int i;
    int vv[Dimension];

    for (i = 0; i < d; i++) {
      vv[i] = quant(v[i], 1<<res[i]);
    }
    return newhash(vv, H);
}

static int flip1(int M, double sn, unsigned long n, double e, double
dist, double delta)
{
    double P;

    P = ((double) M) / (sn * pow(n, e) * pow((dist==0?1:dist),delta));
#if USE_RANDOM
    if (rand()/(1.0+RAND_MAX) <= P) return 1;
#else
    if (drand48() <= P) return 1;
#endif
    return 0;
}

static int flip2(int M, double sn0, double sn, unsigned long nj,
unsigned long n0, double e, double dist0, double distj, double delta)
{
    double P, Pp, PP;
    P = ((double) M)/( sn0*pow(n0,e)*pow((dist0==0?1:dist0),delta) );
    Pp= ((double) M)/( sn*pow(nj,e)*pow((distj==0?1:distj),delta) );

    if (P > 1) P = 1;
    if (Pp >= 1) return 1;
```

```
/* P = (sn0*pow(n0, e))/(sn*pow(nj, e)); */

#if USE_RANDOM
    PP = rand()/(1.0+RAND_MAX);
#else
    PP = drand48();
#endif
    return (PP <= Pp / P);
}

static int
trim(int M, int *res, int H, double sn, hash_t *n, double *bv,
unsigned long *bn, double *bs, int nb, double e, double *bd, double
dist, double delta)
{
    int j, k, l;
    hash_t n_now;

#if VERBOSE
    printf( "trim %d buffer slots", nb);
#endif

    for (j = k = 0; j < nb; j++) {
            n_now = n[quanthash(&bv[j*d], res, H)];

            if (flip2(M,bs[j],sn,n_now,bn[j],e,bd[j],dist,delta)) {
                  for (l = 0; l < d; l++)
                        bv[k*d+l] = bv[j*d+l];
                  bn[k] = n_now;
                  bs[k] = sn;
                  bd[k] = dist;
                  k++;
            }
    }
#if VERBOSE
    printf( " left with %d\n", k);
#endif
    return k;
}

double getDistance(double ls, unsigned long n, int dm)
{
      double dist;

      if(dm==1)
            dist = ( ls / (n<=1?1:n*(n-1)/2) );
      else // if(dm==2)
            dist = ( ls / (n<=1?1:n-1) );
      return dist;
}

double euclidist(DOT *v1, DOT *v2, int dim)
{
      int   i;
      double SS=0;      // Sum Square

      for(i=0;i<dim;i++)
            SS += pow(v1[i]-v2[i],2);
```

```
      return sqrt(SS) + 1;    // +1 to garantee that the power of the
                              // distance is an increasing function
}

void getCentroid(DOT *v, unsigned long    n, int dim, DOT *centroid)
{
      int   i;

      for(i=0;i<dim;i++)
            centroid[i] = v[i]/(n==0?1:n);

      return;
}

char *filetype_cv(char *oldfile, char *type)
{
      int i;
      char *newfile = (char *)malloc(100);
      strcpy(newfile,oldfile);
      for(i=strlen(newfile)-1;i>=0;--i){
            if(newfile[i]=='.'){
                  newfile[i] = NULL;
                  break;
            }
            if(newfile[i]=='\\')
                  break;
      }
      strcat(newfile, type);
      return newfile;
}

//*************************************************************//
// Main
//

void main(int argc, char **argv)
{

    int quanthash(double *v, int *res, int H);
    int    H;           /* size of the hash table */
    int    M;           /* number of points that we want to output */
    int    nb;          /* number of buffer entries */
    int    nib;         /* number in buffer */
    double        *bv;  /* buffer of object being output */
    unsigned long *bn;  /* buffer of n[i] when entered into buffer */
    double        *bs;  /* buffer of sn */
    double        *bd;  /* buffer of distance */
    int           *res; /* number of cuts to apply */
    double  resp;
    hash_t *n;    // array of hash tables counting # in bucket
#if PRINT_GROUPS
    int *no;      // array of hash tables counting # output by bucket
    double *wo;   // array of weighted sum of points output by bucket
#endif
    double sn;          /* sum ni^(1-e)*dist^(-delta) */
    double *v;          /* vector for each input point */
    int    i, j, k;
    unsigned long    N;       /* number of points read so far */
```

```
    int    weighted = 0;
    int    binary = 0;
    int    warned = 0;
    int    mem;
    double e = 1;
    double delta = 1;
      int      savelog=0;     // savelog=1 to save log file to *.log
      int      nLoop=0;       // Number of loop to do multiple
                              // sampling.
      char   outType[] = ".arff";
      unsigned long    byteBuff, byteHash;

      // Plus Variable
      DOT         *centroid;
      CLS         **clsinfo;  // Store cluster info list, Linear Sum
                              // and Number of data points.
      CLS         **clsCur;   // Store current cluster info, Linear
                              // Sum and Number of data points.
      int         dm;         // Distance Measure Type (1:weighted,
                              // 2:non-weighted).
      double      dist;       // Distance Measure Value.


   //
   // Additional Variables
   //
   struct _timeb startTime, endTime;
   char *timeline;
   clock_t startc, endc;
   char *inFile, *outFile;
   FILE *inset, *outset;


#if USE_RANDOM
    srand(time(NULL));
#else
    srand48(time(NULL));
#endif

      printf( "\n\\>DBSPACE---------------------------------\n");
      for(i=0;i<argc;i++)
            printf( "%s ", argv[i]);
      printf( "\n");
      inFile      = argv[1];  argc--; argv++;
      outFile     = argv[1];  argc--; argv++;
      while (argc > 1) {
           if(strcmp(argv[1],"-binary")==0||strcmp(argv[1],"-b")== 0)
           {
                 binary++;
                 argc--;
                 argv++;
            } else if (strcmp(argv[1], "-weighted") == 0
                        || strcmp(argv[1], "-w") == 0) {
                 weighted++;
                 argc--;
                 argv++;
            } else if (argc > 2 && strcmp(argv[1], "-exp") == 0) {
                 e = atof(argv[2]);
```

```
                argc -= 2;
                argv += 2;
        } else if (argc > 2 && strcmp(argv[1], "-loop") == 0) {
                nLoop = atoi(argv[2]);
                argc -= 2;
                argv += 2;
        } else if (strcmp(argv[1], "-log") == 0
                        || strcmp(argv[1], "-l") == 0) {
                savelog = 1;
                argc--;
                argv++;
        } else
                break;
    }

    if(argc!=7||(d=atoi(argv[1]))<=0||(M=atoi(argv[2]))<=0
        ||(mem=atoi(argv[3]))<=0||(resp=atof(argv[4]))<=0
        ||(dm = atoi(argv[5]))<=0) {
        printf( "usage: inputFile outputFile [-binary | -weighted
                    | -exp e | -loop n | -log] d M mem res dmtype
                    delta\n");
        printf( "\n----------------------------------------\n");
        exit(1);
    }
    delta = atof(argv[6]);

//*************************************************************/
// Take loop
//
char  sIndex[20];
char  *oldOut = outFile;
int         lcount            = 1;
do{
    if(nLoop>0){
        printf(">>Sample no.: %d\n",lcount);
        sprintf(sIndex, "_no%02d%s", lcount,outType);
        outFile     = filetype_cv(oldOut,sIndex);
    }
    //*************************************************************/
    // BEGIN: [Exact] Density Biased Sampling
    //

    //
    //  Time stamp (BEGIN: startc)
    //
    fflush(stdin);
    srand(time(NULL));

    _ftime( &startTime );
    timeline = ctime( & ( startTime.time ) );
    printf( "Process begin:\t\t%.19s.%hu %s", timeline,
        startTime.millitm, &timeline[20] );

    startc = clock();

    //
    // BEGIN: DBS algorithm
    //
```

```
      // Open input file
      if(!(inset=fopen(inFile,"r"))){
       printf("Cannot read %s",inFile);
       getch();
       return;
      }

      tabsize = H = mem;
      tab = (tab_t **)malloc(sizeof(*tab)*tabsize);

      nb = M*BUFFER_FACTOR;

      byteBuff = (unsigned long)nb*(sizeof(*bv)*d + sizeof(*bn)
                  + sizeof(*bs) + sizeof(*bd));
      byteHash = (unsigned long)H*(sizeof(*n)+sizeof((*clsCur)->ls)*d
                  + sizeof((*clsCur)->lsd));
      printf( "MEMORY: buffer %d entries %lu bytes, hash %d buckets
                  %lu bytes\n", nb, byteBuff, H, byteHash);
      printf( "        Total buffer %lu bytes\n", byteBuff +
                  byteHash);

      // ****************************************** //
      // Initialize variables
      //
      n  = (hash_t *)malloc(H*sizeof(*n));
#if PRINT_GROUPS
    no = (int *)malloc(H*sizeof(*no));
    wo = (double *)malloc(H*sizeof(*wo));
#endif
    v  = (double *)malloc(sizeof(*v)*d);
    bv = (double *)malloc(sizeof(*bv)*nb*d);
    bn = (unsigned long *)malloc(sizeof(*bn)*nb);
    bs = (double *)malloc(sizeof(*bs)*nb);
    bd = (double *)malloc(sizeof(*bd)*nb);
    res= (int *)malloc(sizeof(*res)*d);

      for (i = 0; i < H; i++){
            n[i] = 0;
            tab[i] = NULL;
      }
      for (i = 0; i < nb; i++)
            bn[i] = 0;
      for (i = 0; i < d; i++) {
            if (resp >= 1) res[i] = resp;
#if USE_RANDOM
            else if (rand()/(1.0+RAND_MAX) < resp) res[i] = 1;
#else
            else if (drand48() < resp) res[i] = 1;
#endif
            else res[i] = 0;
      }

      N = nib = ntab = 0;
      sn = 0;

      // DB+ variables
      centroid= (DOT *)malloc(sizeof(*centroid)*d);
      clsinfo = (CLS **)malloc(sizeof(*clsinfo)*H);
```

```
for(i=0;i<H;i++){
      clsCur      = &clsinfo[i];
      *clsCur     = (CLS *)malloc(sizeof(**clsCur));
      (*clsCur)->ls= (DOT *)malloc(sizeof(*(*clsCur)->ls)*d);
      for(j=0;j<d;j++)
            (*clsCur)->ls[j]  = 0;
      (*clsCur)->n      = 0;
      (*clsCur)->lsd    = 0;
}

//
// End Init
// ****************************************** //

// ****************************************** //
// Start read input file
//
for (;;) {
if (binary) {
      if (fread(v, sizeof(v[0]), d, stdin) != d)
            break;
}
else {
    for (i = 0; i < d; i++)
            if (fscanf(inset, "%lf", &v[i]) != 1)
                  break;
    if (i < d)
            break;
}
if (! warned)
    for (i = 0; i < d; i++)
            if (v[i] < 0 || v[i] > 1) {
                  printf( "WARNING: All inputs must be in unit
                        hypercube(truncated value:%f).\n",v[i]);
                  warned = 0;
            }


N++;
#if PRINT_GROUPS
      if (N%10000 == 0) printf( "[%d %g %d]\n", N, sn, nib);
#endif

i = quanthash(v, res, H);
clsCur = &clsinfo[(int)i];

if (n[i]){  // if cell containing some object, delete old value
            // first.(and will update later)
      sn -= pow(n[i],1-e)*pow(getDistance((*clsCur)->lsd
                              ,(*clsCur)->n,dm),-delta);
}

// DBSPACE zone
// Count an object to be a member of cell
(*clsCur)->n            += 1;
n[i]++;

// Calculate Linear Sum of distance between previous centroid
```

```
	// and current object.
	if(dm==1)
		(*clsCur)->lsd    += ((*clsCur)->n<=1?1:((*clsCur)->n-1)
					*euclidist(centroid,v,d));
	else  // if(dm==2)
		(*clsCur)->lsd    += ((*clsCur)->n<=1
					?1:euclidist(centroid,v,d));

	// Update Linear Sum including current object
	for(j=0;j<d;j++)
		(*clsCur)->ls[j] += v[j];

	// Update centroid including current object
	getCentroid((*clsCur)->ls, (*clsCur)->n, d, centroid);

	// Update sn to current object
	dist  =     getDistance((*clsCur)->lsd,(*clsCur)->n,dm);
	sn    +=    pow(n[i], 1-e)*pow(dist, -delta);

	if (flip1(M, sn, n[i], e, dist, delta)) {
	    if (nib >= nb) {
			nib = trim(M, res, H, sn, n, bv, bn, bs, nb, e, bd,
					dist, delta);
			#if PRINT_GROUPS > 2
				for (i = 0; i < H; i++) {
					no[i] = 0;
					wo[i] = 0;
				}
				for (i = 0; i < nib; i++) {
					no[quanthash(&bv[i*d], res, H)]++;
					wo[quanthash(&bv[i*d], res, H)] +=
					    pow(bd[i],delta)*pow(bn[i], e)*sn/M;
				}
				for (i = 0; i < ntab; i++){
					printf( "gc %4d %4ld %4d %9.4f %6.4f
(%6.4f | %6.4f) %6.4f %6.4f\n", i, n[i], no[i], wo[i],
((double)no[i])/n[i], M/(sn*pow(n[i], e)*pow(dist,delta)),
nib/(sn*pow(n[i], e)*pow(dist,delta)), wo[i]/n[i], dist);
				}
			#endif

		}
		if (nib >= nb) {
			#if VERBOSE
				printf( "panic nib >= nb for point %d\n", N);
			#endif
			#if USE_RANDOM
				j = rand()/(1.0+RAND_MAX)*nb;
			#else
				j = drand48()*nb;
			#endif
		} else {
			// ************************* //
			// Add <P,x> into buffer (only if n[i]>1)
			//
			if(n[i]>1){
				j = nib++;
				for (k = 0; k < d; k++)
```

```
                              bv[j*d+k] = v[k];
                        bn[j] = n[i];
                        bs[j] = sn;
                        bd[j] = dist;
                  }
                  //
                  // ************************* //
            }

      }

      } // end of file

      #if PRINT_GROUPS
            printf( "sn %g\n", sn);
            for (i = 0; i < H; i++) {
                  no[i] = 0;
                  wo[i] = 0;
            }
      #endif
    nib = trim(M, res, H, sn, n, bv, bn, bs, nib, e, bd, dist, delta);

      // ******************************************************/
      // BEGIN: Output the sample to output file
      //

      int outFlag=0;

      // Open output file
      do{
       if(outFlag){
         printf("Enter new output file: ");
         scanf("%s",outFile);
       }
       if(!(outset=fopen(outFile,"w"))){
         printf("Cannot save file to %s!\n",strstr(outFile,"\\"));
         outFlag = 1;
       }else
         outFlag = 0;
      }while(outFlag);

      //******************/
      // BEGIN: Writing process
      //

      // Write arff header
      if(!strcmpi(outType,".arff")){
            fprintf(outset, "@RELATION %s\n\n",
strrchr(outFile,'\\')+1);
            for(i=0;i<d;i++)
                  fprintf(outset, "@ATTRIBUTE ATTR%02d numeric\n", i);
            fprintf(outset, "\n@DATA\n\n");
      }

      for (i = 0; i < nib; i++) {
            #if PRINT_GROUPS
                  no[quanthash(&bv[i*d], res, H)]++;
                  wo[quanthash(&bv[i*d], res, H)] +=
```

```
                        pow(bd[i],delta)*pow(bn[i], e)*sn/M;
      #endif
      for (j = 0; j < d; j++)
            fprintf(outset, "%g ", bv[i*d+j]);
      if (weighted)
           fprintf(outset, "%g %d", pow(bd[i],delta)*pow(bn[i],
                        e)*sn/M, quanthash(&bv[i*d], res, H));
      fprintf(outset, "\n");
}

//
// END: Write file
//******************/

fclose(inset);          // Close input file
fclose(outset);         // Close output file
//
// END: Output the sample to output file
// *******************************************************/

//
//  Time stamp (END: endc)
//
endc = clock();

_ftime( &endTime );
timeline = ctime( & ( endTime.time ) );
printf( "output %d/%d points (%.2f%%) from %d groups\n", nib, N,
      (double)nib*100/N, ntab);
printf( "Process finished:\t%.19s.%hu %s", timeline,
      endTime.millitm, &timeline[20] );
printf("Processing time is\t%lg seconds\n",(double)(endc-startc)
      /CLK_TCK);
printf( "\n----------------------------------------\n");

//
// END: [Exact] Density Biased Sampling
// *******************************************************/

// *******************************************************/
// BEGIN: Writing log file
//
if(savelog){
      char *logFile;
      FILE *logset;
      logFile = filetype_cv(outFile, ".log");

      // Open log file
      outFlag=0;
      do{
      if(outFlag){
      printf("Enter new log file: ");
      scanf("%s",logFile);
      }
      if(!(logset=fopen(logFile,"w"))){
      printf("Cannot open %s\n",*logFile);
      outFlag = 1;
      }else
```

```
            outFlag = 0;
            }while(outFlag);

            fprintf(logset, "\\>DBSPACE v1.0 %02d------------------
                        ---\n", lcount);
            fprintf(logset, "Input file : %s\n",inFile);
            fprintf(logset, "Weighted: %d\n",weighted);
            fprintf(logset, "exp e (epsilon): %f\n",e);
            fprintf(logset, "Dimension: %d\n",d);
            fprintf(logset, "Sampling size: %d\n",M);
            fprintf(logset, "Buffer Factor: %g\n"
                        ,(double)BUFFER_FACTOR);
            fprintf(logset, "Hash size: %d\n",H);
            fprintf(logset, "Hash Resolution: %g\n",resp);
            fprintf(logset, "delta: %f\n",delta);
            fprintf(logset, "Dist. type: D%d\n",dm);
            fprintf(logset, "Output file: %s\n\n",outFile);
            timeline = ctime( & ( startTime.time ) );
            fprintf(logset, "Process begin:\t%.19s.%hu %s", timeline,
                        startTime.millitm, &timeline[20] );
            timeline = ctime( & ( endTime.time ) );
            fprintf(logset, "Process finished:\t%.19s.%hu %s",
                        timeline, endTime.millitm, &timeline[20] );
            fprintf(logset, "Processing time is\t%lg seconds\n\n",
                        (double)(endc-startc)/CLK_TCK);
            fprintf(logset, "output %d/%d points (%.2f%%) from %d
                        groups\n", nib, N, (double)nib*100/N, ntab);
            fprintf(logset, "MEMORY: buffer %d entries %lu bytes,
                        hash %d buckets %lu bytes\n", nb, byteBuff, H,
                        byteHash);
            fprintf(logset, "        Total buffer %lu bytes\n",
                        byteBuff + byteHash);
            fprintf(logset, "\nOutput file has been saved to
                        \"%s\".\nProcess completed.",outFile);
            fprintf(logset, "\n-----------------------------------
                        ----\n");

            #if PRINT_GROUPS
                  fprintf(logset, "sn %g\n", sn);
                  fprintf(logset, "|- %4s %4s %4s %9s %6s (%6s | %6s)
%8s %8s %8s %8s\n", "[i]", "ni", "no", "wo", "no/ni", "P[M]","P[nib]",
"wo/ni", "ni^e", "di^del", "dist");
                  for (i = 0; i < ntab; i++){
                        clsCur      = &clsinfo[(int)i];
                        dist  = getDistance((*clsCur)->lsd
                                                ,(*clsCur)->n,dm);
                        fprintf(logset, "gc %4d %4ld %4d %9.4f %6.4f
(%6.4f | %6.4f) %8.4f %8.4f %8.4f %8.4f\n", i, n[i], no[i], wo[i],
((double)no[i])/n[i], M/(sn*pow(n[i], e)*pow(dist,delta)),
nib/(sn*pow(n[i], e)*pow(dist,delta)), wo[i]/n[i], pow(n[i], e),
pow(dist,delta), dist);
                  }
            #endif

      }
      //
      // END: Writing log file
      // ******************************************************/
```

```
    // Free memory
    free(n);
    #if PRINT_GROUPS
      free(no);
      free(wo);
    #endif
    free(v);
    free(bv);
    free(bn);
    free(bs);
    free(bd);
    free(res);
    free(centroid);
    free(clsinfo);

}while(lcount++ < nLoop);
//
// Take loop
//
*********************************************************************/

    return;

}
```

ประวัติผู้เขียน

นายธรรมศักดิ์ เธียรนิเวศน์ เกิดเมื่อวันที่ 3 ธันวาคม พ.ศ. 2522 เริ่มเข้าศึกษาในระดับปริญญาตรีในปีการศึกษา 2540 ที่สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี และสำเร็จการศึกษาเมื่อปี พ.ศ. 2544 ภายหลังสำเร็จการศึกษาได้เข้าร่วมงานกับบริษัทซอฟต์สแควร์ 1999 จำกัด ซึ่งเป็นบริษัทที่รับออกแบบและพัฒนาระบบงานทางธุรกิจ ด้วยความสนใจระบบงานทางด้านฐานข้อมูลเป็นพิเศษจึงทำให้เกิดแรงจูงใจที่จะศึกษาต่อในระดับที่สูงขึ้น โดยได้เข้าศึกษาต่อในระดับปริญญาโทที่สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2547

ในระหว่างการศึกษาได้รับความอนุเคราะห์อย่างดียิ่งจากคณาจารย์ในสาขาวิชา โดยได้รับความไว้วางใจให้เป็นผู้ช่วยวิจัย และผู้สอนปฏิบัติการรายวิชา Database System