บทที่ 2
ทฤษฎี งานวิจัย และเอกสารที่เกี่ยวข้อง

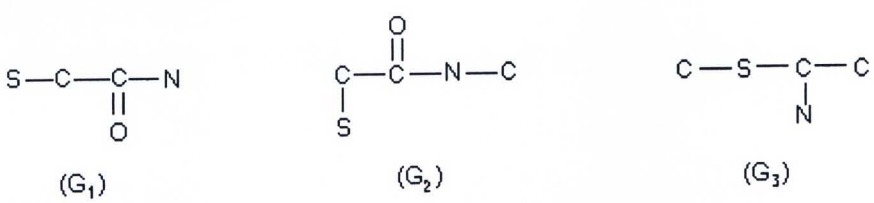
2.1 การทำเหมืองกราฟ

การทำเหมืองกราฟ (Graph Mining) เป็นการวิเคราะห์ข้อมูลที่อยู่ในรูปของกราฟ เพื่อค้นหารูปแบบและความสัมพันธ์ในชุดข้อมูล โดยอาศัยแนวคิดทฤษฎีพื้นฐานทางด้านกราฟ เพื่อใช้ในการจัดเก็บรวบรวมและเตรียมข้อมูลที่ใช้ในการทำเหมืองกราฟ (Cook and Holder, 2007) เทคนิคการทำเหมืองกราฟมีหลายเทคนิคดังต่อไปนี้

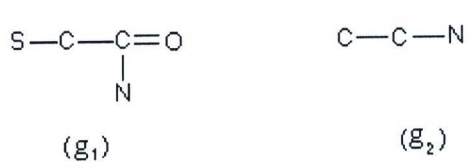
2.1.1 การค้นหาความถี่ของโครงสร้างย่อย (Discovery of frequent substructures)

เทคนิคการค้นหาความถี่ของโครงสร้างย่อย เป็นเทคนิคหนึ่งในการทำเหมืองกราฟ โดยปกติกระบวนการค้นหาความถี่ของโครงสร้างย่อยจะประกอบไปด้วย การหาความถี่ทั้งหมดของโครงสร้างย่อยและตรวจสอบความถี่ของแต่ละโครงสร้างย่อยว่ามีค่าความถี่น้อยกว่า minimum support หรือไม่

บทนิยาม 2.1 กราฟที่ปรากฏบ่อย (Frequent Graph) กำหนดให้ ชุดข้อมูลกราฟ $D = \{G_1, G_2, ..., G_n\}$ และ *support(g)* เป็นเปอร์เซ็นต์หรือจำนวนของกราฟใน D เมื่อ g เป็นกราฟย่อย กราฟ g เป็นกราฟที่ปรากฏบ่อย ถ้าค่า *support(g)* มีค่ามากกว่าหรือเท่ากับค่า minimum support threshold



รูปที่ 2.1 ตัวอย่างชุดข้อมูลกราฟ (graph dataset)



รูปที่ 2.2 กราฟที่ปรากฏบ่อย

รูปที่ 2.1 แสดงตัวอย่างของชุดข้อมูลที่เป็นโครงสร้างทางเคมี ในรูปที่ 2.2 แสดงกราฟย่อยที่ปรากฏบ่อยที่อยู่ในชุดข้อมูลในรูปที่ 2.1 จะเห็นว่า กราฟ g_1 มีค่า $support = 2$ และ g_2 มีค่า $support = 3$

วิธีการค้นหาความถี่ของโครงสร้างย่อยมีหลายวิธี ดังต่อไปนี้

(1) Apriori-base Approach

ขั้นตอนวิธีที่ใช้หาความถี่ของโครงสร้างย่อยถูกเสนอขึ้นครั้งแรกโดย Inokuchi และคณะ ขั้นตอนวิธีของ Apriori-base โดยทั่วไปเป็นไปตามรูปที่ 2.3 เมื่อ S_k เป็นเซตของโครงสร้างย่อยที่ปรากฏบ่อยที่มีขนาด k ซึ่งโครงสร้างย่อยใหม่จะถูกสร้างโดยการรวมกันของโครงสร้างย่อยที่ปรากฏบ่อย ดังแสดงในบรรทัดที่ 4 กราฟใหม่ที่ได้จะถูกหาความถี่และใช้ในการสร้างกราฟในรอบถัดไป ในแต่ละรอบขนาดของโครงสร้างย่อยที่ปรากฏบ่อยจะเพิ่มขึ้นทีละ 1

Apriori (D, *min_support*, S_k)
Input : A graph dataset D and *min_support* .
Output : A frequent substructure set S_k.
1 : $S_{k+1} \leftarrow \phi$;
2 : for each frequent $g_i \in S_k$ do
3 : for each frequent $g_j \in S_k$ do
4 : for each size (k + 1) graph g fromed by the merge of g_i and g_j do
5 : if g is frequent in D and $g \notin S_{k+1}$ then
6 : insert g to S_{k+1} ;
7 : if $S_{k+1} \neq \phi$ then
8 : call Apriori (D, *min_support*, S_{k+1}) ;
9 : return ;

รูปที่ 2.3 ขั้นตอนวิธี Apriori

ในขั้นตอนวิธี Apriori ขั้นตอนในการสร้างกราฟใหม่จากการรวมกันของโครงสร้างย่อยที่ปรากฏบ่อย เป็นขั้นตอนที่ยุ่งยากซับซ้อน ซึ่งเป็นปัญหาในการทำ frequent substructure mining ภายหลัง Apriori-based Graph Mining (AGM) เสนอให้ใช้จุดยอดในการเพิ่มขนาดของโครงสร้างย่อย โดยเพิ่มจุดยอดทีละ 1 จุดในแต่ละรอบของขั้นตอนวิธี และ Frequent subgraph (FSG) เสนอโดย Kuramochi และ Karypis ใช้เส้นเชื่อมในการเพิ่มขนาดของโครงสร้างย่อย โดยในแต่ละรอบของขั้นตอนวิธี Apriori เส้นเชื่อมจะถูกเพิ่มขึ้นทีละ 1 เส้น เพื่อแก้ปัญหาดังกล่าว

(2) Pattern Growth Approach

กราฟ g สามารถขยายได้โดยการเพิ่มเส้นเชื่อม e เขียนสัญลักษณ์แทนด้วย $g\Diamond_x e$ ถ้าเส้นเชื่อม e เกิดจากการเพิ่มจุดยอดใหม่สามารถเขียนสัญลักษณ์แทนด้วย $g\Diamond_{xf} e$ แต่ถ้าเส้นเชื่อม e ไม่ได้เกิดจากการเพิ่มจุดยอดใหม่จะเขียนสัญลักษณ์แทนด้วย $g\Diamond_{xb} e$

รูปที่ 2.4 เป็นขั้นตอนวิธีของ pattern growth-base frequent substructure mining สำหรับการค้นหากราฟ g โดยจะทำการขยายไปเรื่อยๆจนกว่าจะพบกราฟปรากฏบ่อยทั้งหมดที่มีกราฟ g อยู่

PatternGrowth(g, D, *min_support*, S)
Input : A frequent graph g, a graph dataset D, and *min_support* .
Output : A frequent substructure set S .
1 : if $g \in S$ then return ;
2 : else insert g to S ;
3 : scan D once, find all the edges e such that g can be extended to $g\Diamond_x e$;
4 : for each frequent $g\Diamond_x e$ do
5 : Call PatternGrowth($g\Diamond_x e$, D, *min_support*, S) ;
6 : return ;

รูปที่ 2.4 ขั้นตอนวิธี PatternGrowth

ขั้นตอนวิธีดังกล่าวเป็นขั้นตอนวิธีที่ง่ายแต่ไม่มีประสิทธิภาพ จากขั้นตอนวิธีนี้ได้นำไปสู่การออกแบบขั้นตอนวิธีใหม่หลายขั้นตอนวิธี เช่น graph-based Substructure pattern mining (gSpan) และ Molecular Fragment Miner (MoFa) เป็นต้น

(3) Variant substructure patterns

Closed Frequent Substructure

รูปแบบที่ปรากฏบ่อยจะเป็น closed ก็ต่อเมื่อไม่มี superpattern ที่มีค่า support เท่ากัน และรูปแบบที่ปรากฏบ่อยจะเป็น maximal ก็ต่อเมื่อรูปแบบนั้นไม่ได้เป็น superpattern ที่ปรากฏบ่อย จากรูปที่ 2.2 กราฟทั้งสองเป็น close frequent graph ในขณะที่ กราฟ g_1 เป็น maximal frequent graph

Approximate Substructure

เป็นทางเลือกหนึ่งในการลดจำนวนของรูปแบบในการทำเหมืองข้อมูลโดยการประมาณความถี่ของโครงสร้างย่อย Holder และคณะ ได้นำทฤษฎี minimum description

length (MDL) มาใช้ในระบบการค้นหาโครงสร้างย่อย ที่เรียกว่า "SUBDUE" ซึ่งทำการค้นหากราฟย่อยบนพื้นฐานของ MDL (Yan and Han, 2007)

2.1.2 การจัดหมวดหมู่ (Classification)

การทำเหมืองกราฟแบบการจัดหมวดหมู่ แบ่งออกได้เป็น 2 ประเภทคือ graph classification และ vertex classification ซึ่งมีรายละเอียดดังต่อไปนี้

(1) Graph classification

ในการจัดหมวดหมู่กราฟจะใช้กราฟเคอร์เนล (graph kernel) เป็นหลัก ซึ่งการออกแบบกราฟเคอร์เนลส่วนใหญ่จะอยู่บนพื้นฐานของการเปรียบเทียบกราฟแต่ละกราฟเพื่อแบ่งแยกรูปแบบที่มีอยู่ในกราฟและกำหนดกราฟเคอร์เนลสำหรับกราฟแต่ละคู่ กราฟเคอร์เนลมี 2 ประเภท ดังต่อไปนี้

Walk-Based Graph Kernels เป็นการ mapping กราฟที่เป็นเซตลำดับชื่อแนวเดิน (walk) ในกราฟ ดังนั้นกราฟเคอร์เนลจะถูกเรียกว่า walk kernel ซึ่งเป็นการกำหนดลักษณะสำหรับทุกๆลำดับชื่อที่เป็นไปได้และนับจำนวนแนวเดินในการเทียบลำดับชื่อกราฟ

Cycle-Base Graph Kernels เป็นการ mapping กราฟโดยแบ่งแยกเป็นเซตของวัฏจักร (cycle) และต้นไม้ (tree) เมื่อวัฏจักรและต้นไม้ที่ถูก mapped แล้วจะเรียกว่า รูปแบบที่เป็น cyclic และต้นไม้

(2) Vertex classification

ไฮเปอร์กราฟ (hypergraph) เป็นเซตของจุดยอด V และเซตของเส้นเชื่อม E เมื่อเส้นเชื่อมแต่ละเส้นเป็นสับเซตของจุดยอด ไฮเปอร์กราฟนิยามโดย $|V|\times|E|$ เมทริกซ์ B โดย $B_{ij}=1$ ก็ต่อเมื่อ $v_i \in e_j$ ไม่เช่นนั้น $B_{ij}=0$ เคอร์เนลเมทริกซ์ นิยามโดย $K=\sum_{i=0}^{\infty}\lambda_i(B^TB)^i$ (Gärtner *et al.*, 2007)

2.2 รายละเอียดเกี่ยวกับ SUBDUE โดยสังเขป

SUBDUE เป็นระบบการเรียนรู้เชิงสัมพันธ์ด้วยกราฟที่ครอบคลุมการเรียนรู้หลายแบบทั้งการค้นหา (discovery) การจัดกลุ่ม (clustering) การเรียนรู้แบบไม่มีผู้สอน (unsupervised learning) และการเรียนรู้แบบมีผู้สอน (supervised learning) ข้อมูลเข้าและข้อมูลออกของ SUBDUE เป็นกราฟแบบมีป้ายชื่อ $G=(V,E,u,v)$ เมื่อ $V=\{v_1,v_2,...,v_n\}$ เป็นเซตของจุดยอด $E=\{(v_i,v_j)\mid v_i,v_j\in V\}$ เป็นเซตของเส้นเชื่อม และ u, v แทนฟังก์ชันป้ายชื่อของจุดยอดและ

เส้นเชื่อมตามลำดับ โดยที่กราฟ G สามารถมีเส้นเชื่อมระบุทิศทาง และเส้นเชื่อมไม่ระบุทิศทาง โดยข้อมูลออกจะเป็นชุดของโครงสร้างย่อยที่มีการจัดอันดับตามความสามารถในการย่อกราฟซึ่งใช้หลักของ MDL โดยเทคนิคการย่อกราฟจะกล่าวในส่วนที่ 2.3

2.2.1 การค้นหาของ SUBDUE (Substructure Discovery in SUBDUE)

ขั้นตอนวิธีการค้นหาของ SUBDUE ใช้ beam search เป็นหลักในการค้นหากราฟย่อย โดยขั้นตอนวิธีที่ใช้ในการค้นหากราฟย่อยแสดงในรูปที่ 2.5

```
SUBDUE(Graph, BeamWidth, MaxBest, MaxSubSize, Limit)
    ParentList = {}
    ChildList = {}
    BestList = {}
    ProcessedSubs = 0
    Create a substructure from each unique vertex label and its single - vertex instances;
        insert the resulting substructures in ParentList
    while ProcessedSubs <= Limit and ParentList is not empty do
        while ParentList is not empty do
            Parent = RemoveHead(ParentList)
            Extend each instance of Parent in all possible ways
            Group the extended instances into Child substructures
            foreach Child do
                if SizeOf(Child) <= MaxSubSize then
                    Evaluate the Child
                    Insert Child in ChildList in order by value
                    if Length(ChildList) > BeamWidth then
                        Destroy the substructure at the end of ChildList
            ProcessedSubs = ProcessedSubs + 1
            Insert Parent in BestList in order by value
            if Length(BestList) > MaxBest then
                Destroy the substructure at the end of BestList
        Switch ParentList and ChildList
    return BestList
```

รูปที่ 2.5 ขั้นตอนวิธีการค้นหาของ SUBDUE

ขั้นแรกของขั้นตอนวิธีเริ่มจากสร้างลิงค์ลิสต์ที่มีชื่อว่า ParentList (ประกอบไปด้วยโครงสร้างย่อยที่จะทำการขยาย), ChildList (ประกอบไปด้วยโครงสร้างย่อยที่ถูกขยาย) และ BestList (ประกอบไปด้วยโครงสร้างย่อยที่มีค่าสูงสุด) เป็นค่าว่าง และให้ ProcessedSubs มีค่าเท่ากับ 0 หลังจากนั้นสร้างโครงสร้างย่อยจากแต่ละป้ายชื่อของจุดยอดที่ไม่ซ้ำกันทั้งหมดโดยเอาไปใส่ไว้ใน ParentList ภายในลูป while แต่ละโครงสร้างย่อยจะถูกเอาออกจาก ParentList และจะถูกขยายทุกทางที่เป็นไปได้โดยการเพิ่มเส้นเชื่อมและจุดยอด หรือเพิ่มเส้นเชื่อมระหว่างจุดยอดสองจุด การหาค่าของ Child จะใช้ MDL ซึ่งหาได้จากสมการ $value(S,G) = \frac{DL(G)}{DL(S) + DL(G \mid S)}$ เมื่อ ParentList มีค่าว่าง จะทำการสลับระหว่าง ParentList กับ ChildList (Cook and Holder, 2000)

2.2.2 การย่อและการหาค่าของโครงสร้างย่อย (Compression and Evaluation of Substructures)

SUBDUE มีวิธีการในการหาค่าของโครงสร้างย่อยที่ดีที่สุด 3 วิธี ดังนี้

(1) การย่อกราฟบนพื้นฐานของ MDL ค่าของโครงสร้างย่อย S ในกราฟ G คือ

$$value(S,G) = \frac{DL(G)}{DL(S) + DL(G \mid S)} \tag{2.1}$$

เมื่อ S คือโครงสร้างย่อยที่ค้นหา

G คือ กราฟนำเข้า

$DL(G)$ คือ จำนวนบิตที่ต้องการในการเข้ารหัสกราฟ

$DL(S)$ คือ จำนวนบิตที่ต้องการในการเข้ารหัสโครงสร้างย่อยที่ค้นหา และ

$DL(G \mid S)$ คือ จำนวนบิตที่ต้องการในการเข้ารหัสกราฟ G หลังจากโครงสร้างย่อย S ถูกย่อ

ในกรณีการเรียนรู้แบบมีผู้สอน ค่าของโครงสร้างย่อย S คือ

$$value(S,GpGn) = \frac{DL(Gp) + DL(Gn)}{DL(S) + DL(Gp \mid S) + DL(Gn) - DL(Gn \mid S)} \tag{2.2}$$

เมื่อ Gp คือ กราฟนำเข้าที่เป็นบวก (positive example)

Gn คือ กราฟนำเข้าที่เป็นลบ (negative example)

(2) การย่อกราฟบนพื้นฐานขนาดของกราฟ ค่าของโครงสร้างย่อย S ในกราฟ G คือ

$$value(S,G) = \frac{size(G)}{size(S) + size(G \mid S)} \tag{2.3}$$

เมื่อ *size* คือ จำนวนของจุดยอดบวกกับจำนวนของเส้นเชื่อม

ถ้าเป็นการเรียนรู้แบบมีผู้สอน คือ

$$value(S,Gp,Gn) = \frac{size(Gp) + size(Gn)}{size(S) + size(Gp \mid S) + size(Gn) - size(Gn \mid S)} \tag{2.4}$$

(3) Set Cover ค่าของโครงสร้างย่อย S คำนวณจากจำนวนของตัวอย่างกราฟเชิงบวก (positive examples) ที่มีโครงสร้างย่อย S ปรากฏอยู่ บวกกับจำนวนของตัวอย่างกราฟเชิงลบ (negative examples) ที่ไม่มีโครงสร้างย่อย S และหารด้วยจำนวนของตัวอย่างกราฟทั้งหมด

2.2.3 ตัวแปรสำหรับการค้นหาโครงสร้างย่อยของ SUBDUE

(1) BEAM เป็นตัวแปรที่ระบุจำนวนสูงสุดของโครงสร้างย่อยที่จะทำการขยายโครงสร้างย่อยในการวนซ้ำของขั้นตอนวิธีการค้นพบ โดยค่าที่เลือกไว้ของ SUBDUE คือ 4

(2) ITERATIONS ใช้ระบุจำนวนรอบในการวนซ้ำเพื่อทำการย่อกราฟ โครงสร้างย่อยที่ดีที่สุดจากการทำซ้ำก่อนหน้านี้สามารถนำไปย่อกราฟสำหรับการวนซ้ำในรอบต่อไป

(3) LIMIT คือจำนวนของโครงสร้างย่อยที่แตกต่างกันในแต่ละรอบของการค้นพบ โดยค่าที่เลือกไว้ของ SUBDUE คือ จำนวนเส้นเชื่อมของกราฟนำเข้าหารด้วยสอง

(4) NSUBS ตัวแปรนี้ใช้เพื่อระบุจำนวนของโครงสร้างย่อยทั้งหมดที่เป็นผลลัพธ์ของการค้นพบโครงสร้างย่อยของ SUBDUE

(5) MAXSIZE ตัวแปรนี้จะระบุจำนวนสูงสุดของจุดยอดที่สามารถมีได้ในโครงสร้างย่อย ซึ่งค่าเริ่มต้นที่ตั้งไว้คือ จำนวนจุดยอดทั้งหมดของกราฟนำเข้า

(6) MINSIZE คือจำนวนของจุดยอดที่น้อยที่สุดในโครงสร้างย่อย โดยค่าที่เลือกไว้ของ SUBDUE คือ 1

(7) OVERLAP โดยปกติ SUBDUE จะไม่อนุญาตให้เกิดการซ้อนกันของโครงสร้างย่อย ตัวแปร overlap จะอนุญาตให้กราฟเกิดการซ้อนกันได้ ในระหว่างที่มีการย่อกราฟ เส้นเชื่อมที่มีการซ้อนทับกันจะถูกเพิ่มขึ้นมาระหว่างโครงสร้างย่อยที่มีการซ้อนกัน

(8) PRUNE ตัวแปรนี้จะตัดโครงสร้างย่อยทิ้ง ถ้าจุดยอดที่เป็นลูกมีค่าน้อยกว่าจุดยอดที่เป็นพ่อแม่ โดยปกติ SUBDUE กำหนดค่าเป็น no pruning

(9) OUTPUT เป็นการควบคุมหน้าแสดงผลของ SUBDUE ซึ่งมีดังต่อไปนี้

1 : แสดงโครงสร้างย่อยที่ดีที่สุดในแต่ละรอบ

2 : แสดงโครงสร้างย่อยในแต่ละรอบตามจำนวนที่กำหนดไว้ใน nsubs

3 : แสดงผลเหมือนใน 2 และแสดงตัวอย่างของโครงสร้างย่อยนั้น

4 : แสดงผลเหมือนใน 3 และแสดงโครงสร้างย่อยในแต่ละรอบที่ค้นพบ

5 : แสดงผลเหมือนใน 4 และแสดงแต่ละโครงสร้างย่อยที่พิจารณา

(10) THRESHOLD ตัวแปรนี้ใช้วัดความคล้ายคลึงกันสำหรับกราฟที่ไม่เหมือนกัน (inexact graph match) กราฟจะคล้ายคลึงกันถ้า *matchcost*(*sub*,*inst*)<=*size*(*inst*)**threshold* โดยค่าเริ่มต้นคือ 0.0 ซึ่งหมายถึงกราฟจะต้องเหมือนกัน

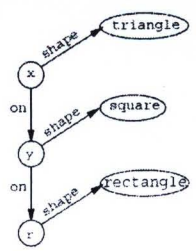
(11) UNDIRECTED โดยปกติ SUBDUE จะกำหนดให้เส้นเชื่อมที่ใช้สัญลักษณ์ e เป็นเส้นเชื่อมแบบระบุทิศทาง ถ้าต้องการให้เส้นเชื่อมเหล่านั้นเป็นเส้นเชื่อมแบบไม่ระบุทิศทางจะต้องใช้ตัวแปร undirected

2.3 การเข้ารหัส Minimum Description Length ของกราฟ

Minimum Description Length (MDL) เสนอโดย Rissanen (Rissanen, 1989) ในหัวข้อนี้เป็นการอธิบายการนำ MDL มาใช้ในการค้นหาโครงสร้างย่อยในกราฟ ซึ่งกำหนดให้ minimum description length ของกราฟเป็นจำนวนบิตที่ใช้ในการอธิบายกราฟ

MDL เป็นการอธิบายกลุ่มของข้อมูลโดยใช้ค่า $I(S)+I(G\mid S)$ เมื่อ S คือโครงสร้างย่อยที่ค้นหาได้, G เป็นกราฟนำเข้า, $I(S)$ คือจำนวนบิตที่ต้องการในการเข้ารหัสโครงสร้างย่อยที่ค้นหาได้ และ $I(G\mid S)$ คือจำนวนบิตที่ต้องการในการเข้ารหัสกราฟนำเข้าเมื่อโครงสร้างย่อย S ถูกย่อ

การเชื่อมโยงของกราฟสามารถแทนโดยเมทริกซ์ประชิด ถ้ากราฟมีจุดยอด n จุด เมทริกซ์ประชิด A จะมีขนาด $n\times n$ ซึ่ง $A[i,j]$ เป็นเซตของ 0 หรือ1 ถ้า $A[i,j]=0$ แสดงว่าไม่มีการเชื่อมโยงจากจุดยอด i ไป j ถ้า $A[i,j]=1$ แสดงว่ามีการเชื่อมโยงจากจุดยอด i ไป j



รูปที่ 2.6 ตัวอย่างกราฟ

กราฟในรูปที่ 2.6 สามารถแทนเป็นเมทริกซ์ประชิดได้ดังนี้

$$\begin{array}{c} x \\ triangle \\ y \\ square \\ r \\ rectangle \end{array} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

การเข้ารหัสของกราฟมีขั้นตอนดังต่อไปนี้ สมมุติให้ผู้ถอดรหัสมีตารางของ l_U ซึ่งเป็นป้ายชื่อที่เป็นหนึ่งเดียว (unique labels)ในกราฟ G

(1) กำหนดจำนวนบิต *vbits* ใช้สำหรับการเข้ารหัสป้ายชื่อของจุดยอดของกราฟ ต้องการ $\lg v$ บิตในการเข้ารหัสจำนวนของจุดยอด v จุดในกราฟ ดังนั้นการเข้ารหัสป้ายชื่อของจุดยอดทุกจุดต้องการ $v \lg l_U$ บิต ฉะนั้นจำนวนบิตทั้งหมดที่ใช้ในการเข้ารหัสป้ายชื่อของจุดยอด คือ

$$vbits = \lg v + v \lg l_U \tag{2.5}$$

จากรูปที่ 2.6 $v = 6$ และสมมุติให้ $l_U = 8$ จำนวนบิตที่ต้องการใช้ในการเข้ารหัสจุดยอดเท่ากับ $\lg 6 + 6\lg 8 = 20.58$ บิต

(2) กำหนดจำนวนบิต *rbits* เป็นจำนวนบิตที่ต้องการใช้ในการเข้ารหัสแถวของเมทริกซ์ประชิด A การเข้ารหัสสายอักขระที่มีความยาว n ของ k ที่มีค่าในเมทริกซ์เป็น 1 และ $(n-k)$ ที่มีค่าเป็น 0 เมื่อ $k << (n-k)$ ในกรณีนี้ แถว $i\,(1 \le i \le v)$ สามารถแทนสายอักขระของความยาว v ที่ประกอบด้วย k_i ถ้าให้ $b = \max_i k_i$ แล้วแถวที่ i ของเมทริกซ์ประชิด สามารถเข้ารหัสดังต่อไปนี้

(2.1) การเข้ารหัสค่าของ k_i ต้องการ $\lg(b+1)$ บิต

(2.2) ให้ k_i ที่มีค่าเป็น 1 ในแถวสายอักขระของความยาว v, ทุกสายอักขระต้องการ $\lg\binom{v}{k_i}$ บิตในการเข้ารหัสของแถวที่ i ที่มีค่าเป็น1

สุดท้าย ต้องใช้ $\lg(b+1)$ บิตเพื่อใช้ในการเข้ารหัสจำนวนบิตที่เป็นค่าเฉพาะของ k_i สำหรับแต่ละแถว ดังนั้นการเข้ารหัสของเมทริกซ์ประชิดคือ

$$\begin{aligned} rbits &= \lg(b+1) + \sum_{i=1}^{v} \lg(b+1) + \lg\binom{v}{k_i} \\ &= (v+1)\lg(b+1) \sum_{i=1}^{v} \lg\binom{v}{k_i} \end{aligned} \tag{2.6}$$

สำหรับตัวอย่างในรูปที่2.6 $b = 2$ และจำนวนบิตที่ต้องการสำหรับเมทริกซ์ประชิดคือ

$$(7\lg 3) + \lg\binom{6}{2} + \lg\binom{6}{0} + \lg\binom{6}{2} + \lg\binom{6}{0} + \lg\binom{6}{1} + \lg\binom{6}{0} = 21.49 \text{ บิต}$$

(3) กำหนดให้ *ebits* เป็นจำนวนบิตที่ต้องการเข้ารหัสเส้นเชื่อมที่แทนโดย $A[i, j] = 1$ ของเมทริกซ์ประชิด A การเข้ารหัส $A[i, j]$ ใช้ $(\lg m) + e(i, j)[1 + \lg l_U]$ เมื่อ $e(i, j)$ คือจำนวนของเส้นเชื่อมในกราฟและ $m = \max_{i,j} e(i, j)$ ดังนั้นจำนวนบิตที่ต้องการใช้ในการเข้ารหัสทั้งหมด คือ

$$\begin{aligned} ebits &= \lg m + \sum_{i=1}^{v}\sum_{j=1}^{v} \lg m + e(i, j)[1 + \lg l_U] \\ &= \lg m + e(1 + \lg l_U) + \sum_{i=1}^{v}\sum_{j=1}^{v} A[i,j]\lg m \\ &= e(1 + \lg l_U) + (K + 1)\lg m \end{aligned} \tag{2.7}$$

เมื่อ e เป็นจำนวนเส้นเชื่อมในกราฟและ K เป็นจำนวนของค่า 1 ที่มีอยู่ในเมทริกซ์ประชิด A สำหรับตัวอย่างในรูปที่ 2.6 $e = 5, K = 5, m = 1, l_U = 8$ ดังนั้น $ebits = 5(1 + \lg 8) + 6\lg 1 = 20$

ดังนั้นการเข้ารหัสของกราฟจะใช้จำนวนบิตทั้งหมด (*vbits* + *rbits* + *ebits*) บิต จากตัวอย่างในรูปที่ 2.6 ใช้จำนวนบิตทั้งหมด 62.07 บิต

ทั้งกราฟนำเข้าและการค้นหาโครงสร้างย่อยสามารถเข้ารหัสโดยใช้หลักการข้างบนได้ หลังจากค้นหาโครงสร้างย่อยได้ แต่ละโครงสร้างย่อยที่ค้นหาได้จะถูกแทนด้วยจุดยอดจุดเดียวในกราฟซึ่งแทนด้วย $I(G \mid S)$ บิต การค้นหาโครงสร้างย่อยถูกแทนด้วย $I(S)$ บิต (Cook and Holder, 1993)

2.4 โครงสร้างคำภาษาล้านนา

ภาษาล้านนา หรือตัวเมือง มีต้นกำเนิดมาจากอักษรมอญโบราณ มักใช้บันทึกบนศิลาจารึก คัมภีร์ใบลาน พับสา เอกสารโบราณ โดยเฉพาะเรื่องราวต่างๆเกี่ยวกับศาสนา เช่น ตำราภาษาบาลี ชาดก ตลอดจนเรื่องราวการกัลปนาไทยทานแก่วัด นอกจากนี้ยังใช้บันทึกเอกสารต่างๆอันเกี่ยวเนื่องในพุทธศาสนา ตำราโหราศาสตร์ ตำราแพทย์ บทกวีนิพนธ์และคร่าวซอ โครงสร้างคำในภาษาล้านนามีความคล้ายคลึงกับภาษาไทย ซึ่งจะประกอบไปด้วย พยัญชนะ สระ ตัวสะกด วรรณยุกต์และสัญลักษณ์พิเศษต่างๆ โดยพยัญชนะและสระจะมีการออกเสียงคล้ายกับในภาษาไทย แต่ภาษาล้านนามีวิธีการเขียนที่แตกต่างจากภาษาไทย ซึ่งสามารถเทียบพยัญชนะและสระของภาษาล้านนากับภาษาไทยได้ ดังแสดงในตารางที่ 2.1 และตารางที่ 2.2 ตามลำดับ

ตารางที่ 2.1 การเทียบพยัญชนะภาษาล้านนากับพยัญชนะภาษาไทย

พยัญชนะ	
ล้านนา	ไทย
ᨣ	ก
ᨡ	ข
ᨤ	ฃ
ᨦ	ค
ᨧ	ฅ
ᨩ	ฆ
ᨦ	ง
ᨨ	จ
ᨪ	ฉ

พยัญชนะ	
ล้านนา	ไทย
ᨫ	ช
ᨬ	ซ
ᨭ	ฌ
ᨮ	ญ
ᨯ	ฎ
ᨰ	ฏ
ᨱ	ฐ
ᨲ	ฑ
ᨳ	ฒ

พยัญชนะ	
ล้านนา	ไทย
ᨵ	ณ
ᨴ	ด
ᨶ	ต
ᨷ	ถ
ᨸ	ท
ᨹ	ธ
ᨺ	น
ᨻ	บ
ᨼ	ป

พยัญชนะ	
ล้านนา	ไทย
ᨽ	ผ
ᨾ	ฝ
ᨿ	พ
ᩀ	ฟ
ᩁ	ภ
ᩂ	ม
ᩃ	ย
ᩅ	ร
ᩆ	ล

พยัญชนะ	
ล้านนา	ไทย
ᩇ	ว
ᩈ	ศ
ᩉ	ษ
ᩊ	ส
ᩋ	ห
ᩌ	ฬ
ᩍ	อ
ᩎ	ฮ

ที่มา : เกษม ศิริรัตน์พิริยะ (2548 : 3)

ตารางที่ 2.2 การเทียบสระภาษาล้านนากับสระภาษาไทย

ชื่อเรียกตามภาษาล้านนา	รูปสระในภาษาล้านนา	รูปสระในภาษาไทย
ไม้ก๋ะ	◌ᩣ	-ะ
ไม้ก๋า	◌ᩤ	-า
ไม้กิ๋	◌ᩥ	◌ิ
ไม้กี๋	◌ᩦ	◌ี
ไม้กึ๋	◌ᩧ	◌ึ
ไม้กื๋	◌ᩨ	◌ื
ไม้กุ๋	◌ᩩ	◌ุ
ไม้กู๋	◌ᩪ	◌ู
ไม้เก๋ะ	เ◌ᩣ	เ◌ะ
ไม้เก๋	เ◌	เ◌
ไม้แก๋ะ	แ◌ᩣ	แ◌ะ
ไม้แก๋	แ◌	แ◌

ชื่อเรียกตามภาษาล้านนา	รูปสระในภาษาล้านนา	รูปสระในภาษาไทย
ไม้โก๋ะ	โ◌ᩣ	โ◌ะ
ไม้โก๋	โ◌	โ◌
ไม้โก๋ (บาลี)	◌ᩮ◌ᩣ	โ◌
ไม้เก๋าะ	เ◌ᩮᩣ	เ◌าะ
ไม้ก๋อ	◌ᩮ	◌อ
ไม้กั๋วะ	◌ᩭᩣ	◌ัวะ
ไม้กั๋ว	◌ᩭ	◌ัว
ไม้เกี๋ยะ	เ◌ᩭᩣ	เ◌ียะ
ไม้เกี๋ย	เ◌ᩭ	เ◌ีย
ไม้เกื๋อะ	เ◌ᩮᩬᩣ	เ◌ือะ
ไม้เกื๋อ	เ◌ᩮᩬ	เ◌ือ
ไม้เก๋อะ	เ◌ᩮᩣ	เ◌อะ

ตารางที่ 2.2 (ต่อ)

ชื่อเรียกตาม ภาษาล้านนา	รูปสระใน ภาษาล้านนา	รูปสระใน ภาษาไทย
ไม้เก๋อ	◌ᩮ	เ - อ
ไม้ไก๋	◌ᩰ	ไ- , ใ-
ไม้ไก๋ย	◌ᩰᩬ	ไ - ย
ไม้ไก๋ยย	◌ᩰᩬᩬ	เ – ยย

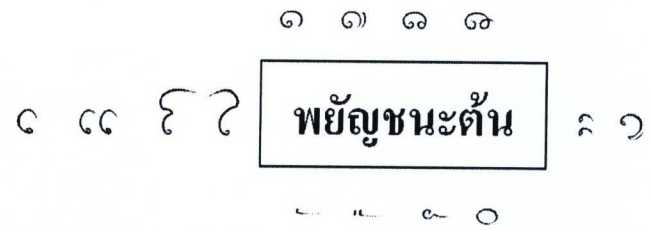
ชื่อเรียกตาม ภาษาล้านนา	รูปสระใน ภาษาล้านนา	รูปสระใน ภาษาไทย
ไม้เก๊า	◌ᩮᩣ	เ – า
ไม้ก๋ำ	◌ᩴᩣ	◌ำ
ไม้กั๋ง	◌ᩴ	◌ั◌
ไม้กิ๋ง	◌ᩤᩴ	◌ิง

ที่มา : เกษม ศิริรัตน์พิริยะ (2548 : 4-5)

การเขียนคำในภาษาล้านนาจะเป็นการเขียนในลักษณะของการผสมคำ โดยเป็นการผสมกันระหว่างพยัญชนะและสระ ซึ่งคำในภาษาล้านนามีการเขียนสระและพยัญชนะในคำไว้หลายระดับ จึงทำให้รูปแบบการเขียนคำในภาษาล้านนามีรูปแบบที่ซับซ้อนกว่าการเขียนคำในภาษาไทย โครงสร้างคำในภาษาล้านนาสามารถแบ่งออกได้เป็น 4 กลุ่มคือ โครงสร้างคำผสมสระ โครงสร้างคำมีตัวสะกด โครงสร้างคำตัวข่มตัวซ้อน และโครงสร้างคำตัวไหล

2.4.1 โครงสร้างคำผสมสระ

คำผสมสระจะวางตำแหน่งของสระไว้รอบๆพยัญชนะต้นคือ ด้านบน ด้านล่าง ด้านหน้า และด้านหลัง ซึ่งสระจะวางไว้ตำแหน่งเดียวกันกับภาษาไทย ยกเว้นไม้ก๋อหรือสระออ และ ไม้กั๋วหรือสระอัว จะวางไว้ด้านล่างของพยัญชนะต้นดังรูปที่ 2.7



รูปที่ 2.7 โครงสร้างคำผสมสระ

2.4.2 โครงสร้างคำมีตัวสะกด

คำที่มีตัวสะกดในภาษาล้านนานั้นจะมีการวางตัวสะกดแตกต่างจากภาษาไทยกลาง ซึ่งในภาษาไทยกลางจะวางตัวสะกดไว้ด้านหลังพยัญชนะต้นเท่านั้น แต่ในภาษาล้านนาตัวสะกดจะถูก

วางไว้ทั้งด้านล่าง และด้านหลังของพยัญชนะต้นขึ้นอยู่กับสระที่ใช้ในคำนั้น ดังนั้นโครงสร้างคำมีตัวสะกดจึงสามารถจัดกลุ่มตามลักษณะตัวสะกดได้ดังต่อไปนี้

(1) โครงสร้างคำมีตัวสะกดวางไว้ด้านล่างพยัญชนะต้น เมื่อพยัญชนะต้นผสมกับสระอะ อา อิ อี อึ อือ เอะ เอ แอะ แอ โอะ โอ และ เออ ตามการออกเสียงในภาษาไทย ตัวสะกดจะถูกวางไว้ด้านล่างของพยัญชนะต้น โดยภาษาล้านนาจะใช้สัญลักษณ์ “ปฏิย๊อก” หรือ (ᩢ) กำกับไว้ด้านบนเมื่อมีการผสมกับสระโอะ และโอ เช่น ᩁᩢ᩠ᨠ (รัก) ᩈᩣ᩠ᨾ (สาม) ᩃᩥ᩠ᨦ (ลิง) ᩮᨠ᩠᩵ᨦ (เก่ง) ᩃᩢ᩠᩶ᨶ (ล้น) ᩰᩋ᩠ᨷ(โอบ) ᨯᩮᩥ᩠ᨶ (เดิน) เป็นต้น

(2) โครงสร้างคำมีตัวสะกดวางไว้ด้านหลังพยัญชนะต้น ตัวสะกดจะถูกวางไว้ด้านหลังพยัญชนะต้นเมื่อมีการผสมกับสระ อุ อู ออ อัว เอีย และ เอือ ตามการออกเสียงในภาษาไทย เช่น ᩈᩪᨦ (สูง) ᨠᩬᨯ (กอด) ᨯ᩶ᩅ᩠ᨿ (ด้วย) ᩁ᩠ᨿᩁ (เรียน) ᨯᩮᩬᩨᩁ (เดือน) เป็นต้น

2.4.3 โครงสร้างคำที่มีพยัญชนะซ้อน

พยัญชนะซ้อนในภาษาล้านนา คือ ตัวสะกดตัวตามในภาษาไทย คำที่มีพยัญชนะซ้อนเป็นการผสมพยัญชนะต้นกับสระจม ร่วมกับตัวสะกดและตัวซ้อนรวมกันอยู่ในหนึ่งคำ โดยตำแหน่งของตัวสะกดจะวางไว้ด้านหลังพยัญชนะต้นและตัวซ้อนจะวางไว้ด้านล่างตัวสะกด ถ้าคำที่มีพยัญชนะซ้อนผสมด้วยสระโอะจะต้องมีเครื่องหมายปฏิย๊อกกำกับไว้ด้านบนพยัญชนะต้นเสมอ โครงสร้างคำที่มีพยัญชนะซ้อนมี 3 ประเภทดังนี้

(1) โครงสร้างคำที่มีพยัญชนะซ้อนชั้นเดียว ตำแหน่งตัวสะกดจะวางไว้ด้านหลังพยัญชนะต้น ส่วนตัวซ้อนจะวางไว้ด้านล่างตัวสะกด เช่น ᩋᨲ᩠ᨲ (อัตตะ) ᩈᨾ᩠ᨾᩣ (สัมมา) ᩅᨲ᩠ᨳᩩ (วัตถุ) ᨽᨶ᩠ᨲᩮ (ภันเต) ᩈᩥᨴ᩠ᨵᩴ (สิทธํ) ᨾᩮᨲ᩠ᨲᩮᩣ (เมตโต) เป็นต้น

(2) โครงสร้างคำที่มีพยัญชนะซ้อนสองชั้น ตำแหน่งตัวสะกดตัวที่ 1 จะวางไว้ด้านหลังพยัญชนะต้น ส่วนตัวซ้อนตัวที่ 1 จะวางไว้ด้านล่างตัวสะกดตัวที่ 1 โดยตัวซ้อนตัวที่ 1 ทำหน้าที่เป็นพยัญชนะต้นผสมกับตัวสะกดตัวที่ 2 สำหรับตำแหน่งของตัวสะกดตัวที่ 2 จะวางไว้ด้านหลังตัวสะกดตัวที่ 1 ทำหน้าที่เป็นตัวสะกด และตัวซ้อนตัวที่ 2 จะวางไว้ด้านล่างตัวสะกดตัวที่ 2 ตัวอย่างเช่น ᨶᨠ᩠ᨡᨲ᩠ᨲ (นักขัตตะ) ᩏᨸ᩠ᨸᨩ᩠ᨫᩣ (อุปัชฌา) ᩈᨾ᩠ᨻᩩᨴ᩠ᨵᩮᩣ (สัมพุทโธ) เป็นต้น

(3) โครงสร้างคำที่มีพยัญชนะซ้อนสามชั้น มีลักษณะคล้ายกับคำที่มีพยัญชนะซ้อนสองชั้น แต่จะมีตัวสะกดตัวที่ 3 และตัวซ้อนตัวที่ 3 เพิ่มมา โดยตัวสะกดตัวที่ 3 จะวางไว้ด้านหลังตัวสะกดตัวที่ 2 ซึ่งทำหน้าที่เป็นตัวสะกด และตัวซ้อนตัวที่ 3 จะวางไว้ด้านล่างตัวสะกดตัวที่ 3 เช่น ᨸᩩᨬ᩠ᨬᨠ᩠ᨡᩮᨲ᩠ᨲᩴ (ปุญญักเขตตํ) ᩈᨾ᩠ᨻᩩᨴ᩠ᨵᩔ (สัมพุทธัสสะ) เป็นต้น

2.4.4 โครงสร้างคำที่มีตัวไหล

ตัวไหลเป็นคำที่ใช้เรียกพยัญชนะควบกล้ำด้วย “ตัวระ” (ᩁ) เทียบได้กับการควบกล้ำด้วย “ร” ในภาษาไทย ซึ่งในภาษาล้านนาเมื่อมีการใช้พยัญชนะควบกล้ำด้วยตัวระ จะมีการเปลี่ยนรูปจาก ᩁ กลายเป็น ᩋ หรือ “ตัวระโฮง” ในภาษาล้านนา (บางครั้งเรียกว่า ระวง ระโว้ง หรือ โฮง) ซึ่งตำแหน่งของตัวระโฮงนี้จะเขียนไว้ด้านหน้าพยัญชนะเสมอ เช่น ᨠᩋᨩ (กราบ) ᨲᩋᩈ (ตรัส) ᨻᩋᨠ (พริก) ᩉᩋ (สระ) เป็นต้น (เกษม ศิริรัตน์พิริยะ, 2548)