TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
LỚP CỬ NHÂN TÀI NĂNG

NGUYỄN THỊ TÌNH - NGUYỄN THANH TUẤN

KHAI THÁC LUẬT KẾT HỢP DỰA TRÊN MẪU ĐỒ THỊ CON PHỔ BIẾN

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP. HCM - 2020

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
LỚP CỬ NHÂN TÀI NĂNG

NGUYỄN THỊ TÌNH - 1612703
NGUYỄN THANH TUẤN - 1612774

KHAI THÁC LUẬT KẾT HỢP DỰA TRÊN MẪU ĐỒ THỊ CON PHỔ BIẾN

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

GIẢNG VIÊN HƯỚNG DẪN
GS. TS. LÊ HOÀI BẮC

KHÓA 2016 - 2020

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

TP.HCM, ngày.....tháng.....năm 2020

GIẢNG VIÊN HƯỚNG DẪN

GS.TS LÊ HOÀI BẮC

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Khóa luận đáp ứng yêu cầu của Khóa luận Cử nhân CNTT.

TP.HCM, ngày.....tháng.....năm 2020

GIẢNG VIÊN PHẢN BIỆN

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin bày tỏ lòng biết ơn sâu sắc đến thầy **GS. TS. Lê Hoài Bắc**, người đã tận tình hướng dẫn, động viên chúng em trong suốt thời gian thực hiện và hoàn thành khóa luận này.

Chúng em xin chân thành gửi lời cảm ơn đến các thầy cô trong Khoa Công nghệ Thông tin - Trường Đại học Khoa học Tự nhiên - Đại học Quốc gia Thành phố Hồ Chí Minh đã tận tình giảng dạy, trang bị cho chúng em nhiều kiến thức quý báu giúp chúng em để hoàn thành khóa học.

Chúng em cũng xin gửi lời cảm ơn đến anh Trương Quốc Bình, đã gián tiếp cung cấp tài liệu để chúng em tham khảo và thực hiện khóa luận dễ dàng hơn.

Mặc dù chúng em đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép, nhưng có thể vẫn còn thiếu sót, rất mong nhận được những đóng góp quý báu từ quý thầy cô và các bạn.

Xin chân thành cảm ơn!

TP.HCM, ngày 10 tháng 08 năm 2020
Nhóm sinh viên thực hiện

Nguyễn Thị Tình - Nguyễn Thanh Tuấn

DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

KTMĐTPB	Khai thác Mẫu đồ thị Phổ biến.
KTLKH	Khai thác Luật kết hợp.
DFS	Depth First Search.
BFS	Breadth First Search.
GPAR	Graph Pattern Association Rules.
MNI	Minimum Image.
FPM	Frequent Pattern Mining.
img(u)	Tập các đỉnh so khớp của đỉnh u trong một đồ thị cho trước.
Q(G)	Tập các ánh xạ của mẫu đồ thị Q trong đồ thị G cho trước.
supp(Q, G)	Độ hỗ trợ của mẫu đồ thị Q trong đồ thị G cho trước.
conf(R, G)	Độ tin cậy của luật kết hợp R trong đồ thị G cho trước.
minsup	Ngưỡng hỗ trợ.

DANH MỤC THUẬT NGỮ

DFS: Tìm kiếm - duyệt theo chiều sâu.

BFS: Tìm kiếm - duyệt theo chiều rộng.

DFS Tree: Một cấu trúc cây được xây dựng dựa trên phương pháp duyệt đồ thị theo chiều sâu.

DFS Code: Mã DFS, một chuỗi cạnh của một đồ thị cho trước được sắp xếp theo thứ tự duyệt theo chiều sâu của đồ thị đó.

Edge Ordering: Quan hệ thứ tự của hai cạnh, được xây dựng theo luật mở rộng trong quá trình duyệt theo chiều sâu của đồ thị. Cạnh nào được mở rộng trước thì đứng bên trái của quan hệ.

DFSEdge: Cấu trúc cạnh mở rộng, ngoài hai đỉnh còn kèm theo thuộc tính của hai đỉnh đó và thuộc tính cạnh tương ứng.

DFS Lexicographic Order: Một quan hệ tuyến tính kết hợp quan hệ thứ tự cạnh và thêm quan hệ thứ tự của phần mở rộng trong hai DFSEdges (thuộc tính đỉnh và cạnh - định nghĩa như thứ tự từ điển).

GPAR: Luật Kết hợp Mẫu đồ thị.

FPM: Bài toán Khai thác Mẫu đồ thị Phổ biến.

MỤC LỤC

LỜI CẢM ƠN **i**

DANH MỤC KÝ KIỆU VÀ CHỮ VIẾT TẮT **ii**

DANH MỤC THUẬT NGỮ **iii**

DANH MỤC HÌNH VẼ **vi**

DANH MỤC BẢNG **vii**

TÓM TẮT KHÓA LUẬN **viii**

CHƯƠNG 1. TỔNG QUAN **1**

1.1 Phát biểu bài toán 1

1.1.1 Bài toán Khai thác Mẫu Đồ thị Phổ biến 1

1.1.2 Bài toán Khai thác Luật Kết hợp 5

1.2 Các thách thức của bài toán 5

1.3 Các công trình nghiên cứu liên quan 6

1.4 Động lực thực hiện 9

1.5 Cấu trúc khóa luận 10

CHƯƠNG 2. KIẾN THỨC NỀN TẢNG **12**

2.1 Các khái niệm liên quan đến đồ thị 12

2.2 Các khái niệm liên quan đến DFS code 16

2.3 Luật Kết hợp Mẫu đồ thị 21

2.4 Khai thác Luật Kết hợp Mẫu đồ thị 22

CHƯƠNG 3. PHƯƠNG PHÁP TIẾN HÀNH **25**

3.1 Khai thác Mẫu Đồ thị phổ biến 25

3.1.1 Tổng quan bài toán FPM 25

3.1.2 Thuật toán tiến hành 26

3.2 Khai thác Luật kết hợp 34

CHƯƠNG 4. THỰC NGHIỆM **37**

4.1 Dữ liệu thực nghiệm 37

4.2 Kết quả thực nghiệm FPM 38

4.3 Kết quả thực nghiệm RuleGen 45

4.3.1 Kết quả thực nghiệm 45

4.3.2 Đánh giá kết quả 49

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN **53**

5.1 Kết luận 53

5.2 Hướng phát triển 54

TÀI LIỆU THAM KHẢO **55**

DANH MỤC HÌNH VẼ

Hình 1.1 Quy trình bài toán 1
Hình 1.2 Mạng xã hội G 4

Hình 2.1 Minh họa thứ tự mở rộng RightMost Path 18
Hình 2.2 Xây dựng DFS Code theo chiến lược mở rộng RightMost Path 19
Hình 2.3 Minh họa DFS Tree 20

Hình 3.1 Kiểm tra Minimum DFS Code 29
Hình 3.2 Ví dụ đề xuất cạnh mới của RuleGen 36

Hình 4.1 Tổng hợp Kết quả thực nghiệm FPM - đồ thị có hướng . . . 41
Hình 4.2 Tổng hợp Kết quả thực nghiệm FPM - đồ thị vô hướng . . . 44

DANH MỤC BẢNG

Bảng 2.1 Đồ thị con đẳng cấu 15
Bảng 2.2 Minh họa DFS Code cho DFS Tree ở Hình 2.3 21

Bảng 3.1 Đồ thị g cho DFS Code γ 29
Bảng 3.2 Các cạnh đơn của đồ thị g 30
Bảng 3.3 Mở rộng RightMost Path trong Minimum DFS Code 30
Bảng 3.4 Các bước kiểm tra Minimum DFS Code 32

Bảng 4.1 Các tập dữ liệu thực nghiệm 38
Bảng 4.2 Kết quả thực nghiệm FPM - đồ thị có hướng 40
Bảng 4.3 Kết quả thực nghiệm FPM - đồ thị vô hướng 43
Bảng 4.4 Kết quả thực nghiệm RuleGen - đồ thị có hướng 46
Bảng 4.5 Kết quả thực nghiệm RuleGen - đồ thị vô hướng 48
Bảng 4.6 Đánh giá kết quả theo Jaccard và Simrank - đồ thị có hướng (Đơn vị: %) 51
Bảng 4.7 Đánh giá kết quả theo Jaccard và Simrank - đồ thị vô hướng (Đơn vị: %) 52

TÓM TẮT KHÓA LUẬN

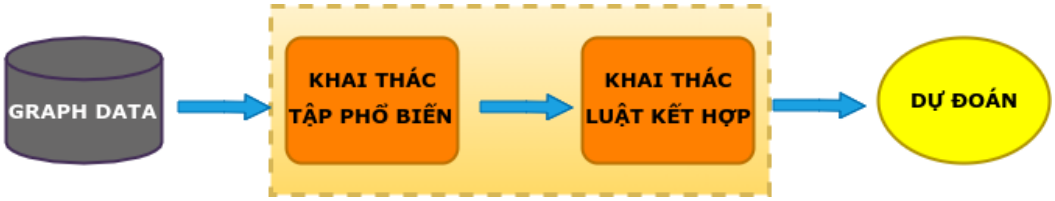
Trong thời đại hiện nay, cùng với sự phát triển mạnh mẽ của Internet cũng như Công nghệ thông tin và các cơ sở hạ tầng liên quan đã giúp việc thu thập và lưu trữ thông tin ngày một tăng lên. Bên cạnh đó, thuật ngữ Khai thác Dữ liệu (Data Mining) đã ra đời cách đây vài thập niên, mục đích là tìm kiếm khối thông tin có ích (tri thức), tiềm ẩn và mang tính dự đoán trong khối cơ sở dữ liệu lớn. Đặc biệt, bài toán Khai thác Tập phổ biến và Luật kết hợp được chú trọng hơn cả với mục đích tìm các thành phần hay xuất hiện cùng nhau và khai thác mối quan hệ giữa chúng. Bài toán này được áp dụng rộng rãi trong nhiều lĩnh vực: thương mại (ví dụ: phân tích các hàng hóa hay được mua bán cùng nhau để xem xét các chương trình khuyến mãi, tăng giá, đầu tư, ...), y học (ví dụ: nghiên cứu các yếu tố gây bệnh thường xuất hiện cùng nhau để có biện pháp phòng ngừa, trị bệnh, chế tạo thuốc, ...), phát hiện gian lận, đạo văn, công cụ tìm kiếm, hệ thống đề xuất, ...

Đã có nhiều phương pháp được đề xuất để giải quyết các bài toán liên quan đến Tập Phổ biến và Luật Kết hợp, tuy nhiên chủ yếu là ở cơ sở dữ liệu dạng giao dịch (transaction). Vì vậy, trong khóa luận này, chúng tôi tiến hành nghiên cứu và xây dựng thuật toán minh họa để khai thác Tập Phổ biến và sinh Luật Kết hợp trên dữ liệu dạng đồ thị. Kết quả thực nghiệm cho ra kết quả Tập Phổ biến chính xác và sinh ra Luật Kết hợp có ích với yêu cầu về thời gian và không gian chấp nhận được.

CHƯƠNG 1. TỔNG QUAN

1.1 Phát biểu bài toán

Bài toán khai thác luật kết hợp từ đồ thị được chia thành hai bài toán con, tổng thể quy trình này được thể hiện trong Hình 1.1.



Hình 1.1: *Quy trình bài toán*

Hai bài toán con Khai thác Mẫu Đồ thị Phổ biến (KTMĐTPB) và bài toán Khai thác Luật kết hợp (KTLKH) được trình bày chi tiết bên dưới.

1.1.1 Bài toán Khai thác Mẫu Đồ thị Phổ biến

Khai thác tập phổ biến: Bài toán khai thác tập phổ biến (Frequent Itemset - về mặt tổng quan) là bài toán rất quan trọng trong lĩnh vực khai thác dữ liệu. Mục tiêu của nó là tìm tất cả các tập mẫu, liên kết, tương quan hoặc cấu trúc nhân quả có độ phổ biến cao (lớn hơn một ngưỡng nào đó) trong tập hợp tất cả các hạng mục hoặc đối tượng trong cơ sở dữ liệu (tập này kí hiệu là $\boldsymbol{I}$). Các tập mẫu, liên kết, cấu trúc khai thác được được gọi chung là **Tập Phổ biến** với định nghĩa như sau: là tập thỏa mãn độ phổ biến tối thiểu cho trước, tức thỏa mãn công thức sau: $supp(X) \geq minsup$.

Kí hiệu: Gọi $\boldsymbol{F}$ là tập phổ biến.

$$F = \{X | X \subseteq I; supp(X) \geq minsup\} \tag{1.1}$$

Trong đó:

$supp(X)$ là độ hỗ trợ của tập X

$minsup$ là ngưỡng hỗ trợ cho trước.

Trên thực tế, các cơ sở dữ liệu tồn tại theo nhiều cấu trúc khác nhau, do đó bài toán khai thác tập phổ biến cũng được phân ra nghiên cứu theo các nhánh khác nhau. Thông thường, hai nhánh nghiên cứu chính là khai thác tập phổ biến trên cấu trúc dữ liệu tập hạng mục - giao dịch (nhiều hạng mục, có hoặc không thứ tự) và trên dữ liệu dạng đồ thị (gồm các đỉnh và mối liên hệ giữa chúng).

Khai thác tập phổ biến trên dữ liệu dạng giao dịch: Đối với dữ liệu này, các tập mục, chuỗi con hoặc cấu trúc có tần suất cao là mục tiêu khai thác của bài toán.

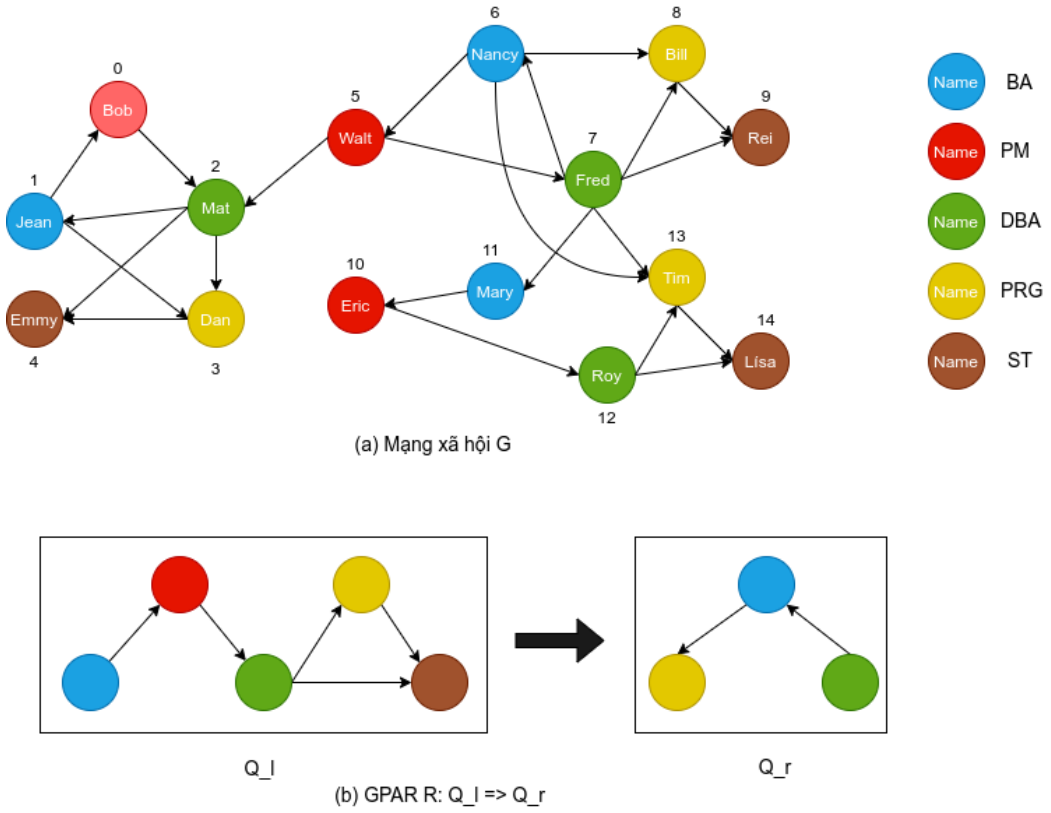
Ví dụ 1. Một tập bánh mì, sữa cùng nhau xuất hiện thường xuyên trong một tập dữ liệu giao tác. Hoặc một chuỗi hành động, thường gọi là mẫu liên tục, như {mua laptop → mua túi chống sốc → mua chuột, bàn phím rời} xuất hiện thường xuyên trong cơ sở dữ liệu mua hàng cũng là mục tiêu của bài toán khai thác của tập phổ biến trên dữ liệu dạng này.

Khai thác tập phổ biến trên đồ thị: Đối với dữ liệu dạng này, do có sự liên kết (thường gọi là cạnh) giữa các thực thể (thường gọi là node/đỉnh) trên đồ thị, nên việc khai thác phải đảm bảo không mất đi vai trò của mối liên

hệ này. Do đó, mục tiêu lúc này là khai thác một cấu trúc con như đồ thị con (subgraph), mẫu đồ thị con (subgraph pattern) có tần suất xuất hiện cao trong cơ sở dữ liệu đồ thị đã cho. Trên thực tế, có hai biến thể của bài toán:

1. **Khai thác tập phổ biến trên một đồ thị lớn:** Tìm các đồ thị hoặc mẫu đồ thị con xuất hiện nhiều lần trên cùng một đồ thị.
2. **Khai thác tập phổ biến trong tập gồm nhiều đồ thị:** Tìm các đồ thị hoặc mẫu đồ thị con xuất hiện phổ biến trên nhiều đồ thị (các đồ thị này cùng một cơ sở dữ liệu, cùng tính chất, có thể coi là các phân mảnh). Hay nói cách khác, mẫu đồ thị con cần tìm đồng thời cùng xuất hiện trên nhiều đồ thị trong tập các đồ thị đã cho.

Ví dụ 2. Một phần của mạng xã hội G được minh họa trong Hình 1.2(a), với mỗi đỉnh biểu diễn một người với tên và nghề nghiệp tương ứng với màu sắc (màu xanh dương - BA - Business Analyst - Phân tích kinh doanh, màu đỏ - PM - Project Manager - Quản lý dự án, màu xanh lá - DBA - Database Administrator - Quản trị viên Cơ sở dữ liệu, màu vàng - PRG - Programmer - Lập trình viên và màu nâu - ST - Software Tester - Kiểm thử phần mềm). Mỗi cạnh có hướng biểu thị mối quan hệ theo dõi (follow) trên mạng xã hội của họ. Hình 1.2(b) minh họa một luật kết hợp mẫu đồ thị dựa vào mạng xã hội ở Hình 1.2(a). Có thể dễ dàng nhận thấy rằng mạng xã hội này tập trung vào năm nhóm người với các nghề nghiệp lần lượt là BA, PM, DBA, PRG, ST. Nếu BA theo dõi PM, PM theo dõi DBA, DBA lần lượt theo dõi PRG và ST, PRG theo dõi ST thì có thể suy ra mối quan hệ sau có thể tồn tại: BA theo dõi PRG, DBA theo dõi BA. Từ đó, có thể đề xuất Mary với Tim, Roy với Mary.



Hình 1.2: *Mạng xã hội G*

Trong khóa luận này, chúng tôi sẽ chú trọng vào vấn đề khai thác các mẫu đồ thị con phổ biến trong một đồ thị lớn.

1.1.2 Bài toán Khai thác Luật Kết hợp

Khai thác luật kết hợp (KTLKH): Khai thác luật kết hợp là bài toán tìm các mối liên hệ, tương quan, sự kết hợp giữa một số thành phần trong cơ sở dữ liệu dựa trên các tập/cấu trúc phổ biến trong cơ sở dữ liệu này. Trong đó, bài toán con khai thác tập phổ biến là bước khó khăn nhất trong quá trình giải quyết bài toán KTLKH. Phát biểu này đúng cho cả cơ sở dữ liệu dạng giao dịch và dạng đồ thị.

KTLKH trên đồ thị: Phần lớn là tìm các mối liên kết ẩn giữa các thành phần trên đồ thị dựa trên việc kết hợp có quy tắc của các đồ thị hoặc mẫu đồ thị phổ biến.

Trong khóa luận này, chúng tôi thực hiện KTLKH trên dữ liệu đồ thị có nhãn (thuộc tính) để tìm ra mối liên hệ giữa các thuộc tính này trong đồ thị.

1.2 Các thách thức của bài toán

Bài toán KTLKH tổng quan và bài toán con khai thác đồ thị con, mẫu đồ thị con phổ biến trên dữ liệu đồ thị là vấn đề khó khăn trong lĩnh vực khai phá các hệ tri thức. Dưới đây là một số khó khăn và thách thức của bài toán đang phải đối mặt:

- Trên các bộ dữ liệu thực tế, đồ thị có kích thước rất lớn, có thể tới hàng triệu đỉnh, vài chục triệu cạnh. Do đó, độ phức tạp của bài toán khá lớn, đặc biệt, việc tìm các đồ thị con phổ biến chiếm rất nhiều thời gian, tài nguyên máy tính. Đây là khó khăn lớn nhất trong bài toán KTLKH nói

chung và việc cải tiến hiệu năng của bài toán này (trên cả giải thuật và cách triển khai) đang được nghiên cứu không ngừng.

- Các dữ liệu đồ thị thực tế có cấu trúc đa dạng: Có hướng, vô hướng, có hoặc không có nhãn. Do đó, khó có phương pháp khai thác nào có thể áp dụng triệt để hiệu quả trên các loại dữ liệu này, hầu hết đều phải phân ra xử lý theo từng loại riêng biệt.
- Đối với dữ liệu có nhãn (thuộc tính), số nhãn dữ liệu nhiều, tuy nhiên việc chọn nhãn phù hợp để bài toán khai thác trở nên ý nghĩa cũng là vấn đề rất được chú trọng.
- Vấn đề chọn các ngưỡng tham số độ hỗ trợ (support), độ tin cậy (confidence) phù hợp để bài toán khai thác trở nên có ý nghĩa. Hiện tại, hầu như chưa có phương pháp chính thống để chọn các ngưỡng này cho hợp lý, do đó hầu như chỉ dựa vào kinh nghiệm hoặc thử nghiệm thực tế để tìm ngưỡng nào tốt nhất. Vấn đề này làm việc giải quyết bài toán mất thêm một khoảng thời gian khá lớn.
- Khai thác tập phổ biến có ý nghĩa thực sự. Các phương pháp hiện tại, mất nhiều chi phí cho việc khai thác tập phổ biến, tuy nhiên, trong các tập khai thác được có thể có rất ít tập có thể sinh ra luật kết hợp thực sự.

1.3 Các công trình nghiên cứu liên quan

Bài toán KTLKH trên đồ thị là bài toán được đánh giá khá phức tạp, đặc biệt là vấn đề về hiệu năng của bài toán, nên bài toán thường được chia thành các bước nhỏ hơn để nghiên cứu. Trong đó, bài toán con, KTMĐTPB (bước

chiếm phần lớn thời gian trong toàn quá trình khai thác luật kết hợp của đồ thị) là vấn đề được khá nhiều nhà khoa học quan tâm và đã có nhiều đề xuất, cách tiếp cận cho việc giải quyết và cải tiến cho bài toán con này.

Trong thời gian đầu giải quyết bài toán KTMĐTPB, các thuật toán được đề xuất thường dựa trên hướng tiếp cận của giải thuật Apriori. Với hướng tiếp cận này, dựa trên nguyên tắc đồ thị con phổ biến thì tất cả các đồ thị con của nó cũng là đồ thị phổ biến, đầu tiên khởi tạo các đồ thị phổ biến ban đầu là các đồ thị đơn giản nhất thỏa mãn *minsup*. Sau đó, thực hiện bước lặp để mở rộng các đồ thị phổ biến mới dựa trên các đồ thị phổ biến cũ bằng cách ghép hai đồ thị đồng dạng nhưng khác nhau về đồ thị phổ biến. Như vậy, ở mỗi bước lặp kích thước của các đồ thị con phổ biến mới có kích thước tăng thêm 1 và có độ hỗ trợ nhỏ hơn hoặc bằng độ hỗ trợ của hai đồ thị con tạo ra nó. Thực hiện việc lặp cho đến khi tập đồ thị mới được khai thác bằng rỗng, tức không còn đồ thị mới nào tạo ra thỏa mãn *minsup* thì dừng lại. Khác với khai thác tập phổ biến trên dữ liệu giao dịch, việc phát sinh tập ứng viên không quá phức tạp thì việc này thực hiện trên đồ thị khá khó khăn bởi vì có rất nhiều cách để ghép hai đồ thị phổ biến để tạo ra các đồ thị phổ biến mới. Hai phương pháp đề xuất nổi bật giải quyết cho vấn đề này là AGM (Inokuchi et al. 2000[1]) và FSG (Kuramochi & Karypis 2001[2]).

AGM (**A**priori-based **G**raph **M**ining) đề xuất phương thức phát sinh ứng viên đó là tăng kích thước các đồ thị con lên 1 đỉnh tại mỗi bước lặp. Hai đồ thị phổ biến kích thước k chỉ được ghép khi hai đồ thị này có cùng đồ thị con kích thước $k-1$. Ở đây kích thước của một đồ thị nghĩa là số các đỉnh trong một đồ thị. Các ứng viên mới được hình thành bao gồm các đồ thị con kích thước $k-1$

và bổ sung hai đỉnh từ hai mẫu kích thước k.

FSG (**F**requent **S**ub-**G**raph mining) đề xuất phương thức phát sinh ứng viên đó là tăng kích thước các đồ thị con lên 1 cạnh tại mỗi bước lặp. Trong FSG, hai đồ thị thước k được ghép lại khi và chỉ khi chúng chia sẻ cùng đồ thị con có $k-1$ cạnh được gọi là core. Ở đây kích thước của một đồ thị nghĩa là số các cạnh trong một đồ thị. Các ứng viên mới được tạo thành bao gồm core và hai cạnh bổ sung từ các đồ thị kích thước k.

Các phương pháp tiếp cận theo hướng trên thường còn được gọi tiếp cận theo hướng BFS. Mặc dù khá hiệu quả về phương diện cắt tỉa giảm thiểu khá nhiều cho chi phí phát sinh ứng viên, các phương pháp này mất nhiều chi phí I/O và bộ nhớ vì muốn phát sinh tập đồ thị phổ biến kích thước $k+1$ phải biết tất cả các đồ thị phổ biến kích thước k trước đó. Đặc biệt, đối với đồ thị kích thước lớn, số đồ thị phổ biến ở mỗi bước lớn thì việc lưu lại chúng là bất khả thi. Để giải quyết vấn đề này, một hướng tiếp cận ngược với các phương pháp cũ, theo chiều sâu - DFS, được nghiên cứu. Thuật toán gSpan (Yan & Han 2002 [3]) được xem là một trong các đề xuất tiên phong đi theo hướng tiếp cận này và cũng là công trình được trích dẫn khá nhiều trong những năm gần đây với những đóng góp nổi bật của nó.

gSpan sử dụng một đại diện duy nhất để thể hiện cho mỗi đồ thị phổ biến. Thuật toán này sử dụng thứ tự từ điển cho DFS (DFS lexicographic) để xây dựng một cấu trúc tương tự như cấu trúc cây phục vụ cho việc lưu trữ và xử lý tất cả các mẫu phổ biến khả thi, việc này tạo thành một không gian tìm kiếm phân cấp gọi là DFS code tree. Mỗi nút của không gian tìm kiếm này đại diện

cho một mã DFS (DFS code), cấp (bậc) thứ $k+1$ của cây là các nút chứa mã DFS cho k đồ thị con. Các đồ thị con k được tạo thành bằng cách mở rộng thêm cạnh từ cấp thứ k của cây. Cây tìm kiếm này được duyệt theo cách DFS hoạt động và tất các đồ thị con không là min của DFS code (minimum DFS code - định nghĩa quan trọng đề xuất của gSpan) sẽ được tỉa bớt nên sẽ tránh được việc dư thừa trong phát sinh tập ứng viên. Như vậy, việc tìm kiếm các đồ thị đẳng cấu sẽ giảm thiểu dựa trên phương pháp tỉa này, điều này giúp giảm thiểu yêu cầu bộ nhớ cho bài toán. Trong khóa luận này, thuật toán chúng tôi triển khai trong bước khai thác tập phổ biến cũng được dựa trên phần lớn ý tưởng của thuật toán gSpan.

1.4 Động lực thực hiện

Bài toán KTLKH có ý nghĩa rất lớn trong việc xây dựng các hệ tri thức và được áp dụng thành công trên nhiều lĩnh vực tài chính, kinh doanh thương mại, hóa học, truyền thông xã hội,.... Đặc biệt, dữ liệu đồ thị ngày càng đa dạng và phổ biến, nên việc khai thác tri thức trên dữ liệu loại này có ý nghĩa vô cùng to lớn và đang được các công ty, doanh nghiệp quan tâm.

Bên cạnh đó, bài toán Khai thác Tập Phổ biến trên tập hạng mục đã phổ biến rộng rãi và có nhiều cách tiếp cận giải quyết khác nhau với hiệu năng khá tốt. Song, đối với Khai thác Tập Phổ biến trên đồ thị, nhiều nghiên cứu đã thực hiện và áp dụng thành công trên nhiều dữ liệu thực tế như google+, twitter, facebook,.... Tuy nhiên, vấn đề hiệu năng vẫn đang cần cải thiện thêm để giải quyết bài toán nhanh hơn, ít tốn chi phí hơn.

Trong khóa luận này, chúng tôi chủ yếu nghiên cứu để cải tiến hiệu năng bài toán trong quá trình triển khai thuật toán (chủ yếu ở giai đoạn tìm đồ thị con phổ biến) để giúp bài toán được giải quyết nhanh hơn.

1.5 Cấu trúc khóa luận

Cấu trúc của khóa luận được chia thành năm chương chính có nội dung lần lượt như sau:

Chương 1: Tổng quan

Giới thiệu về bài toán khai thác luật kết hợp dựa trên mẫu đồ thị con phổ biến, các ứng dụng cũng như thách thức, kèm theo các công trình nghiên cứu liên quan và động lực thúc đẩy chúng tôi thực hiện khóa luận này.

Chương 2: Kiến thức nền tảng

Trình bày các kiến thức nền tảng mà chúng tôi sử dụng trong khóa luận này: Khái niệm đồ thị, đồ thị con, mẫu đồ thị, đồ thị đẳng cấu, các khái niệm liên quan DFS Code, bài toán KTMĐTPB, bài toán KTLKH.

Chương 3: Phương pháp tiến hành

Trình bày đầy đủ và chi tiết thuật toán đã triển khai bao gồm hai thuật toán chính: Khai thác Mẫu Đồ thị con phổ biến (**FPMiner**) và Khai thác Luật Kết hợp từ Mẫu Đồ thị con phổ biến (**RuleGen**).

Chương 4: Thực nghiệm

Giới thiệu về dữ liệu thực nghiệm, trình bày các kết quả thu được.

Chương 5: Kết luận và hướng phát triển

Trình bày kết luận chung về các kết quả đạt được, ưu nhược điểm và hướng phát triển đề tài khóa luận trong tương lai.

CHƯƠNG 2. KIẾN THỨC NỀN TẢNG

2.1 Các khái niệm liên quan đến đồ thị

Chúng tôi bắt đầu với những khái niệm và lưu ý về đồ thị, mẫu, đồ thị con đẳng cấu và đồ thị phân tán.

Đồ thị: (Data Graph - Graph) được định nghĩa là $G = (V, E, L)$:

1. V là một tập hợp các đỉnh.
2. $E \subseteq V \times V$ là một tập hợp các cạnh vô hướng mà ở đó (v, v') hay (v', v) đều biểu thị một cạnh vô hướng giữa hai đỉnh v và v'.
3. Mỗi đỉnh v thuộc V có chứa một danh sách thuộc tính cố định $L(v) = (A_1 = a_1, ..., A_n = a_n)$, trong đó, $A_i = a_i$ với $i \in [1, n]$ biểu diễn rằng đỉnh v có một giá trị a_i ứng với thuộc tính A_i.

Đồ thị có hướng được định nghĩa tương tự, ứng với mỗi cạnh (v, v') biểu thị một cạnh có hướng từ v đến v'.

Mẫu đồ thị: (pattern graph - pattern) là một đồ thị $Q = (V_p, E_p, f)$:

1. V_p và E_p là tập các đỉnh và cạnh tương ứng.
2. f là một hàm được định nghĩa trên V_p sao cho với mọi đỉnh $u \in V_p, f(u)$ là một thuộc tính được xác định theo công thức có dạng $A = a$, với A là thuộc tính của đỉnh và a là giá trị của A.

Chú ý: Trong các dữ liệu thực tế, mỗi đỉnh trên đồ thị (một thực thể trong thực tế) có rất nhiều thuộc tính cần xét. Do đó, để tăng tính ngữ nghĩa cho bài toán, có thể áp dụng bài toán khai thác nhiều thuộc tính cùng lúc (thông qua các phép biến đổi - embedding). Để đơn giản, trong khóa luận này, chúng tôi lấy đồ thị mẫu chỉ có 1 thuộc tính cho các đỉnh.

Đồ thị con: (Subgraph) Một đồ thị $G' = (V', E', L')$ được gọi là đồ thị con của đồ thị $G = (V, E, L)$, kí hiệu $G' \subseteq G$ nếu thỏa mãn các điều kiện sau: $V' \subseteq V$, $E' \subseteq E$, và với mỗi đỉnh $v \in V'$, $L'(v) = L(v)$.

Định nghĩa tương tự cho mẫu đồ thị $Q' = (V'_p, E'_p, f')$ là mẫu đồ thị con của mẫu đồ thị $Q = (V_p, E_p, f)$, kí hiệu $Q' \subseteq Q$ nếu $V'_p \subseteq V_p$, $E'_p \subseteq E_p$ và với mỗi đỉnh $u \in V'_p$, $f'(u) = f(u)$.

Phép đẳng cấu: (Isomorphism) Xét đồ thị G và mẫu đồ thị Q, một đỉnh v trong G thỏa mãn các điều kiện tìm kiếm của một đỉnh u trong Q. Tức là, nếu với mỗi thành phần của "$A = a$" trong $f(u)$ tồn tại một thuộc tính A sao cho $v.A = a$. Lúc này, quan hệ u và v được ký hiệu là $v \sim u$. Như vậy, một phép đẳng cấu là một hàm song ánh h từ các đỉnh của Q đến các đỉnh của G. Chi tiết hơn, phép đẳng cấu được định nghĩa theo từng loại đồ thị như sau:

1. **Đối với đồ thị có hướng:** Cho hai đồ thị vô hướng $G1 = (X1, E1)$ và $G2 = (X2, E2)$. Hai đồ thị $G1$ và $G2$ được gọi là đẳng cấu với nhau nếu tồn tại hai song ánh $f1$ và $f2$ thỏa mãn điều kiện sau:
   - $f1 : X1 \to X2$ và $f2 : E1 \to E2$
   - Nếu cạnh $e \in E1$ liên kết với cặp đỉnh $(x, y) \subseteq X1$ xét trong đồ thị $G1$ thì cạnh $\mu(e)$ sẽ liên kết với cặp đỉnh $(f2(x), f2(y))$ xét trong đồ thị $G2$ (điều này được gọi là sự tương ứng cạnh).

2. **Đối với đồ thị có hướng:** Cho hai đồ thị có hướng $G1 = (X1, E1)$ và $G2 = (X2, E2)$. Hai đồ thị G1 và G2 được gọi là đẳng cấu với nhau nếu tồn tại hai song ánh $f1$ và $f2$ thỏa mãn điều kiện sau:
   - $f1 : X1 \to X2$ và $f2 : E1 \to E2$
   - Nếu cạnh $e \in E1$ liên kết với cặp đỉnh $(x, y) \in X1$ xét trong đồ thị $G1$ thì cạnh $\mu(e)$ sẽ liên kết với cặp đỉnh $(f2(x), f2(y))$ xét trong đồ thị $G2$ (điều này được gọi là sự tương ứng cạnh).

Đồ thị con đẳng cấu: (Subgraph isomorphism) Một đồ thị con đẳng cấu là một phép đẳng cấu từ Q đến một đồ thị con G_s của G. Khi một phép đẳng cấu h từ Q đến đồ thị con G_s của G tồn tại, G_s được ký hiệu là một bản so khớp (match) của Q trong G. Như vậy, v trong G_s là một bản so khớp với u trong Q, khi đó $v \sim u$. Ký hiệu mở rộng:

- $Q(G)$ tập các bản so khớp của Q trong G.
- Ảnh $img[Q, G]$ của Q trong G được ký hiệu bằng tập $\{(u, img(u))|u \in V_p\}$, trong đó $img(u)$ là ảnh của u trong G, bao gồm các đỉnh v duy nhất trong G là các bản khớp của u trong Q.

Ví dụ 3. Xét đồ thị có hướng G và mẫu đồ thị Q_l ở hình 1.2, ta có tập các đồ thị con đẳng cấu tương ứng với Q_l (hay các so khớp của Q_l trong G) được liệt kê trong Bảng 2.1 dưới đây:

Trong đó, thứ tự tập cạnh trong các đồ thị con đẳng cấu trên là: {(BA, PM), (PM, DBA), (DBA, PRG), (ST, DBA), (DBA, ST)}. Cạnh (BA, PM) là cạnh có hướng từ đỉnh BA (tức đỉnh có thuộc tính BA trong G) đến đỉnh PM (đỉnh có thuộc tính PM trong G).

STT	BA	PM	DBA	PRG	ST
1	Jean	Bob	Mat	Dan	Emmy
2	Nancy	Walt	Mat	Dan	Emmy
3	Nancy	Walt	Fred	Bill	Rei
4	Mary	Eric	Roy	Tim	Lisa

Bảng 2.1: *Đồ thị con đẳng cấu*

Xét đồ thị đẳng cấu 1 ở ví dụ trên, ta có ánh xạ $f(u) = a$ từ mẫu đồ thị Q_l đến đồ thị con tương ứng trong G: $\{f(PM) = Jean, f(BA) = Bob, f(DBA) = Mat, f(PRG) = Dan, f(ST) = Emmy\}$.

Dữ liệu đồ thị phân tán: (Distributed data graphs) Trong thực tế, một đồ thị G thường được phân thành một tập gồm các đồ thị con và được lưu trữ ở những nơi khác nhau (gọi là site). Một tập các phần con (*fragmentation*) F của đồ thị $G = (V, E, L)$ là $(F_1, ..., F_n)$, trong đó phần con, *fragment* F_i được định nghĩa là $(V_i \cup F_i.O, E_i \cup cE_i, L_i)$, sao cho:

1. $(V_1, ..., V_n)$ là một phần của G.
2. $F_i.O$ là tập các đỉnh v' sao cho tồn tại cạnh e=(v, v') trong E, $v \in V_i$ và đỉnh v' thuộc một phần con khác. v' được xem như một đỉnh ảo *(virtual node)*, e là cạnh chéo *(crossing edge)* và cE_i là tập các cạnh chéo.
3. $(V_i \cup F_i.O, E_i \cup cE_i, L_i)$ là một đồ thị con của G được tạo bởi $V_i \cup F_i.O$.

Đồ thị liên thông: (Connected Graph)

Đối với đồ thị vô hướng: Cho đồ thị vô hướng $G = (E, V, L)$ và $v_a \in V$, $v_b \in V$. Đồ thị G được gọi là liên thông nếu với mọi cặp đỉnh (v_a, v_b) bất kỳ

trong G, luôn tồn tại một đường đi từ điểm này đến điểm kia.

Đối với đồ thị có hướng:

1. **Liên thông mạnh: (Strongly connected)** Đồ thị G được gọi là liên thông mạnh nếu với mọi cặp đỉnh (v_a, v_b) trong G luôn tồn tại đường đi từ v_a đến v_b và ngược lại.
2. **Liên thông yếu: (Weakly connected)** Đồ thị G được coi là liên thông yếu nếu có đường đi giữa 2 đỉnh (v_a, v_b) bất kỳ của đồ thị vô hướng tương ứng với đồ thị G (hay đồ thị G hủy đi hướng của các cạnh).
3. **Liên thông một phần: (Unilaterally connected)** Đồ thị G gọi là liên thông một phần nếu với mọi cặp đỉnh (v_a, v_b) có tồn tại ít nhất một đường đi từ một đỉnh đến đỉnh còn lại.

2.2 Các khái niệm liên quan đến DFS code

Phần này sẽ trình bày khái niệm về DFS code, DFS code tree, chiến lược mở rộng DFS code và các khái niệm liên quan đến thứ tự trong DFS gồm có egde order và DFS lexicographic order.

Cây DFS: (DFS tree) Cho đồ thị G, *DFS tree* T_G của nó là cây được xây dựng thông qua phương pháp duyệt đồ thị theo chiều sâu (DFS - **D**epth **F**irst **S**earch) bắt đầu từ một đỉnh trong G. Chuỗi các đỉnh được duyệt trong quá trình xây dựng T_G tuân theo một thứ tự tuyến tính.

Theo lý thuyết trên, ta có $i \in [1, n]$ là chuỗi thứ tự duyệt trong quá trình xây dựng DFS tree, v_i được duyệt trước v_j khi $i < j$. Từ đó, trên cây được xây dựng, v_0 và v_n tương ứng là node gốc (*root* node) và node phải nhất (rightmost

node), đường đi từ v_0 đến v_n được gọi là *rightmost path* (đường đi/cạnh phải nhất) và cạnh $e = (v, v')$ có thể được thể hiện bằng một cặp có thứ tự (i, j).

Cạnh tiến và cạnh lùi: (Forward Edge - Backward Edge) Xét một cạnh $e = (i, j)$, nếu $i < j$ thì cạnh e là cạnh forward, ngược lại thì cạnh e là cạnh backward.

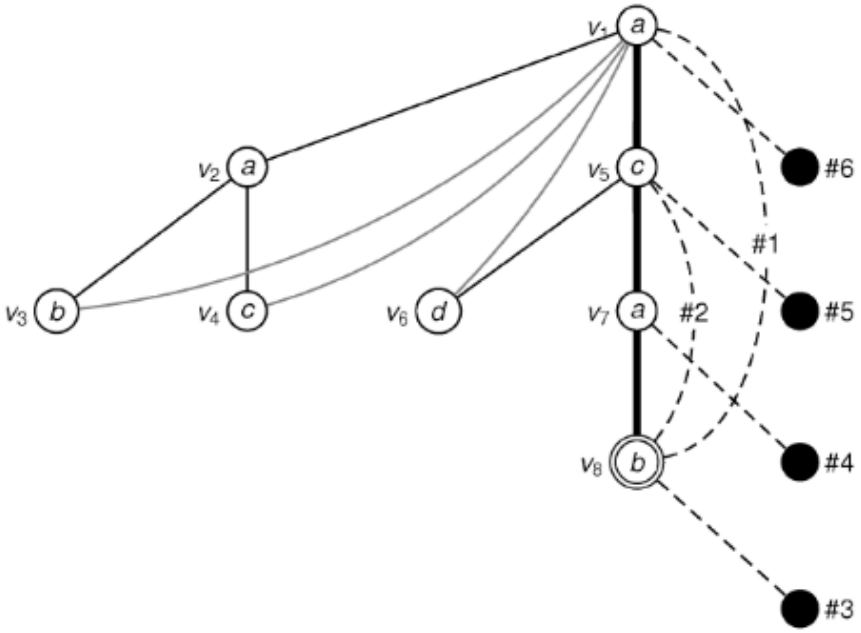
Mở rộng đường đi phải nhất: (Rightmost Path Extension) Chiến lược mở rộng của một đồ thị dựa trên các nodes ở đường đi phải nhất. Một mở rộng tiến (forward extension) là thêm một cạnh forward từ các nodes trên đường đi phải nhất đến một node mới. Một mở rộng lùi (backward extension) là thêm một cạnh backward từ node phải nhất lần lượt đến các nodes thuộc đường đi phải nhất. Trong quá trình mở rộng, luôn xem xét tất cả mở rộng backward trước, rồi mới đến mở rộng forward.

Đối với mở rộng backward, gọi u_r là node phải nhất, một cạnh (u_r, v_i) thì được xem xét mở rộng trước cạnh (u_r, v_j) với $i < j$. Hay nói cách khác, mở rộng backward gần node gốc hơn thì được xem xét trước các nodes khác xa node gốc thuộc đường đi phải nhất.

Đối với mở rộng forward, gọi v_x là đỉnh mới được thêm vào, cạnh (v_i, v_x) thì được xem xét mở rộng trước cạnh (v_j, v_x) nếu $i > j$. Hay nói cách khác, các node ở xa node gốc hơn thì được xem xét mở rộng trước các nodes ở gần node gốc. Node v_x sẽ được đánh số thứ tự là $x = r + 1$ và sẽ trở thành node phải nhất mới sau khi mở rộng.

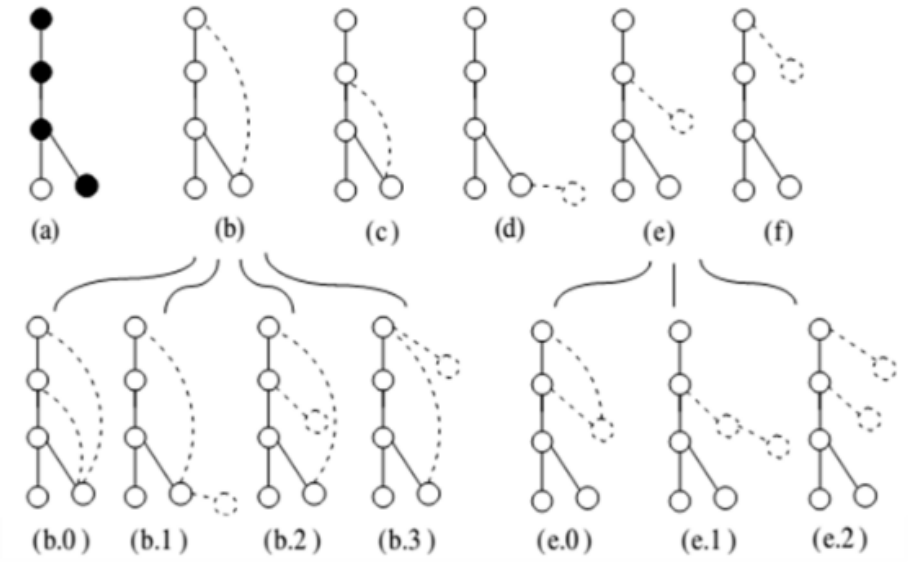
Ví dụ 4. Thứ tự mở rộng đường đi phải nhất được minh họa trong Hình 2.1 với các nodes màu đen là đỉnh mới được thêm vào ứng với chiến lược mở

rộng forward, #* là thứ tự mở rộng ứng với từng chiến lược. Cụ thể, node v_8 là node phải nhất, đường đi phải nhất bao gồm các nodes lần lượt từ gốc là: $\{v_0, v_5, v_7, v_8\}$. Theo chiến lược mở rộng backward, ta lần lượt thêm các cạnh $\{(v_8, v_0)(\#1), (v_8, v_5)(\#2)\}$ (*Lưu ý:* cạnh (v_8, v_7) đã tồn tại nên không được mở rộng). Theo chiến lược mở rộng forward, lần lượt mở rộng theo các cạnh sau: $\{(v_8, v_9)(\#3), (v_7, v_9)(\#4), (v_5, v_9)(\#5), (v_0, v_9)(\#6)\}$.



Hình 2.1: *Minh họa thứ tự mở rộng RightMost Path*

Ví dụ 5. Hình 2.2 bên dưới minh họa việc phát triển và mở rộng một DFS Code hợp lệ. Các nodes màu đen ở Hình 2.2 (a) biểu thị đường đi phải nhất. Hình 2.2 (b) - (c) mô tả chiến lược mở rộng backward, nếu mở rộng (b) hợp lệ, tiếp tục phát triển DFS Code theo các Hình 2.2 (b.0) - (b.3) và cứ tiếp tục như vậy. Nếu đã xét hết khả năng mở rộng backward, tiến hành mở rộng forward như Hình 2.2 (d) - (f) biểu diễn, tương tự nếu mở rộng (e) tồn tại, tiếp tục phát triển DFS Code theo các Hình 2.2 (e.0) - (e.2) và cũng tiếp tục như vậy cho đến khi hoàn thành.



Hình 2.2: *Xây dựng DFS Code theo chiến lược mở rộng RightMost Path*

Thứ tự cạnh: (Edge ordering) Dựa trên quá trình mở rộng DFS Tree của một đồ thị G, một quan hệ nhị phân $\prec_{E,T}$ được xây dựng trên tập cạnh E theo thứ tự mở rộng của DFS Tree, trong đó: tất cả các mở rộng backward của một node đều được xem là đến trước (ở bên trái của quan hệ) so với mọi mở rộng forward của node đó. Lúc này, thứ tự $\prec_{E,T}$ chính là thứ tự cạnh của T_G.

Mã DFS: (DFS code) Cho DFS tree T_G cùng với thứ tự $\prec_{E,T}$ của đồ thị G, DFS code của đồ thị G trên T_G, ký hiệu là $\alpha(T_G)$, là một chuỗi cạnh sao cho $e_i \prec_{E,T} e_{i+1}$ (với $i \in [0, |E| - 1]$).

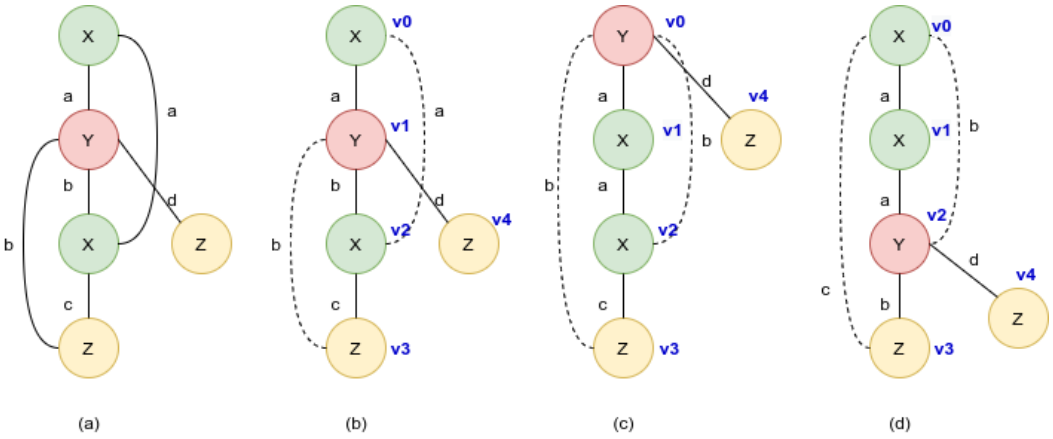
Thứ tự từ điển DFS: (DFS lexicographic order) Cho đồ thị G có F là tập những DFS tree của G. Biết rằng Z=$\{\alpha(T)|T \in F\}$, Z là tập bao gồm tất cả các DFS code của G. Biết $\prec_L$ là ký hiệu thứ tự tuyến tính của các phần tử trong L_A. Khi đó, kết hợp $\prec_{E,T}$ và $\prec_L$ là một thứ tự tuyến tính $\prec_e$ trên tập $E \times L_A \times L_A$. Thứ tự tuyến tính này là chính là thứ tự từ điển DFS, nó được định nghĩa như sau.

Nếu $\alpha(T_1) = (a_0, a_1, ..., a_m)$, $\alpha(T_2) = (b_0, b_1, ..., b_n)$ và $\alpha(T_1)$, $\alpha(T_2) \in Z$, khi đó $\alpha(T_1) \leq \alpha(T_2)$ khi và chỉ khi thỏa một trong các điều kiện sau đây:

1. $\exists t$, $0 \leq t \leq min(m, n)$, $a_k = b_k$, với $k < t$, $a_t \prec_e b_t$.
2. $a_k = b_k$, với $0 \leq k \leq m$ và $n \geq m$.

DFS Code Nhỏ nhất: (Minimum DFS Code - Canonical DFS Code) Xét đồ thị G cho trước, $Z(G) = \{\alpha(T) | T \in \digamma\}$, $Z(G)$ là tập hợp các DFS Code của đồ thị G. Dựa trên DFS Lexicographic Order, tồn tại DFS Code nhỏ nhất, kí hiệu min(Z(G)), hay còn gọi là đại diện (canonical) cho đồ thị G.

Định lý: Cho hai đồ thị G và G', G được gọi là đồ thị đẳng cấu của G' khi và chỉ khi $min(G) = min(G')$.



Hình 2.3: *Minh họa DFS Tree*

Ví dụ 6. Đồ thị ở Hình 2.3(a) cho 10 cách duyệt DFS khác nhau. Bảng 2.2 liệt kê 3 DFS Codes trong số đó tương ứng với Hình 2.3 (b), (c), (d). Theo DFS Lexicographic Order, $\gamma \prec \alpha \prec \beta$ và γ cũng chính là min DFSCode cần tìm.

Cây DFS code: (DFS code tree) Một DFS code tree là một cây có hướng:

1. Node gốc (root) là một node ảo.
2. Mỗi node không phải là node gốc, ký hiệu là v_α, là một DFS code α.

Edge	α**(b)**	β**(c)**	γ**(d)**
0	(0, 1, X, a, Y)	(0, 1, Y, a, X)	(0, 1, X, a, X)
1	(1, 2, Y, b, X)	(1, 2, X, a, X)	(1, 2, X, a, Y)
2	(2, 0, X, a, X)	(2, 0, X, b, Y)	(2, 0, Y, b, X)
3	(2, 3, Y, c, Z)	(2, 3, X, c, Z)	(2, 3, Y, b, Z)
4	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	(3, 0, Z, c, X)
5	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	(2, 4, Y, d, Z)

Bảng 2.2: *Minh họa DFS Code cho DFS Tree ở Hình 2.3*

3. Một node v_α có DFS code $(e_0, ..., e_k)$, node con của nó phải là một DFS code có dạng $(e_0, ..., e_k, e')$.
4. Thứ tự DFS code của node anh/em của v_α phải thỏa mãn thứ tự từ điển DFS.

Tương tự, một DFS code graph là một đồ thị có hướng với node gốc là một node ảo và những node không phải node gốc thỏa mãn điều kiện của DFS code.

2.3 Luật Kết hợp Mẫu đồ thị

Luật kết hợp của mẫu đồ thị: (Graph-pattern association rules - GPARs) Một luật kết hợp mẫu đồ thị R được định nghĩa là quan hệ $Q_l \Rightarrow Q_r$. Trong đó, Q_l và Q_r là các mẫu đồ thị liên thông có ít nhất một cạnh, chúng có các nodes chung nhưng không có cạnh chung. Trong trường hợp này, Q_l và Q_r lần được gọi là tiền đề (antecedent) và kết quả (consequent).

Một luật R trong đồ thị G thể hiện rằng: Có một phép ánh xạ đẳng cấu h_r từ Q_l đến một đồ thị con G_1 của G, mặt khác cũng tồn tại một ánh xạ h_r từ Q_r đến một thị con G_2 khác của G, sao cho:

- Mỗi đỉnh $u \in V_l \cap V_r$, nếu u được ánh xạ h_l đến đỉnh v trong G_1 thì u cũng được ánh xạ h_r đến đỉnh v giống như vậy trong G_2.
- Có một đỉnh trong G là một so khớp với mẫu đỉnh u của Q_1 và Q_r, cứ như vậy, thỏa mãn điều kiện này với tất cả các đỉnh u của Q_l và Q_r.

Một luật kết hợp R được xây dựng bằng cách từ mẫu đồ thị Q_R, sau đó mở rộng sang Q_l với tập cạnh của Q_r. Quá trình mở rộng và phát sinh luật thỏa mãn các điều kiện sau:

1. Q_R, Q_l và Q_r là các đồ thị liên thông.
2. Q_l và Q_r không rỗng, chúng phải có ít nhất một cạnh.
3. Q_l và Q_r không có cạnh chung.

Ví dụ 7. Như đã biểu diễn trong Hình 1.2 (b) và Ví dụ 2, một luật kết hợp được phát biểu như sau: $R : Q_l \Rightarrow Q_r$, với Q_l và Q_r lần lượt là tiền đề và kết quả tương ứng, R có để được xây dựng là một mẫu đồ thị Q_R bằng cách mở rộng Q_l với tập cạnh của Q_r.

2.4 Khai thác Luật Kết hợp Mẫu đồ thị

Trong phần này, chúng tôi sẽ bàn về các khái niệm thường gặp trong bài toán khai thác luật kết hợp như độ hỗ trợ, độ tin cậy và tìm hiểu tổng quan về bài toán khai thác luật kết hợp.

Độ hỗ trợ: (Support) Cho đồ thị G và mẫu đồ thị Q, độ hỗ trợ của Q trong G, ký hiệu là $supp(Q, G)$, thể hiện tần suất xuất hiện Q trong G.

Tương tự như tìm tập phổ biến cho tập hạng mục, phương pháp tính độ hỗ trợ của mẫu đồ thị nên có tính chất đơn điệu (*monotonic*) để giúp tỉa nhánh, nghĩa là cho hai mẫu đồ thị Q và Q', nếu $Q' \subseteq Q$ thì $supp(Q', G) >= supp(Q, G)$ với bất kì đồ thị G hay nói cách khác, nếu một mẫu đồ thị không phổ biến thì mọi đồ thị cha của nó cũng không phổ biến.

Có nhiều phương pháp tính độ hỗ trợ mà sử dụng được tính đơn điệu. Ở khóa luận này, chúng tôi sử dụng phương pháp tính độ hỗ trợ *minimum image* (MNI - số lượng ảnh ánh xạ nhỏ nhất ứng với từng đỉnh [4]) cho mẫu đồ thị, được định nghĩa như sau:

$$supp(Q, G) = min\{|img(u)||u \in V_Q\} \tag{2.1}$$

Trong đó, $img(u)$ là ảnh ánh xạ của node u thuộc mẫu đồ thị Q đang xét trong đồ thị G.

Tương tự, độ hỗ trợ của một GPAR R là:

$$supp(R, G) = supp(Q_R, G) \tag{2.2}$$

bằng cách xem luật R như là mẫu đồ thị Q_R.

Độ tin cậy: (Confidence) Để xác định tính đúng đắn của luật kết hợp được sinh ra với Q_l và Q_r, người ta dựa trên độ đo gọi là độ tin cậy. Độ tin cậy

của luật kết hợp R được xác định theo công thức sau:

$$conf(R, G) = \frac{supp(Q_R, G)}{supp(Q_l, G)} \tag{2.3}$$

Ví dụ 8. Xem lại Ví dụ 3, mặc dù có 4 đồ thị đẳng cấu ứng với mẫu đồ thị Q_l trong đồ thị G nhưng $supp(Q_l, G) = 3$ bởi vì $|img(u, G)| = 3$ với mỗi $u \in Q_l$. Tương tự, tập so khớp của Q_R trong G, $img(Q_R, G)$ là {(BA, {Jean, Nancy}), (PM, {Bob, Walt}), (DBA, {Mat, Fred}), (PRG, {Dan, Bill}), (ST, {Emmy, Rei})}. Vì vậy, $supp(Q_R, G) = 2$; $conf(R, G) = \frac{2}{3}$.

Bài toán Khai thác Luật Kết hợp: (Mining Association Rules Problem) Bài toán khai thác luật kết hợp được chia thành 2 bước chính:

1. Bước 1. Khai thác tập phổ biến từ đồ thị G dựa trên độ hỗ trợ θ đã cho trước. Kết quả thu được sau bước này là tập các mẫu đồ thị Q có độ hỗ trợ $supp(Q) \geq \theta$.
2. Bước 2. Sinh luật kết hợp từ các mẫu Q tìm được ở *Bước 1* dựa trên độ tin cậy η cho trước. Sau bước này, ta thu được các luật kết hợp R có $conf(R, G) \geq \eta$.

Algorithm 1: Miner

Input: Đồ thị G, độ hỗ trợ θ, độ tin cậy η.

Output: Tập các GPARs R thỏa $supp(R, G) \geq \theta, conf(R, G) \geq \eta$.

1 Khai thác mẫu đồ thị Q thỏa $supp(Q, G) \geq \theta$ từ đồ thị G;

2 Sinh tất cả các GPARs R thỏa $conf(R, G) \geq \eta$ từ mẫu đồ thị Q;

3 **return** GPARs.

CHƯƠNG 3. PHƯƠNG PHÁP TIẾN HÀNH

3.1 Khai thác Mẫu Đồ thị phổ biến

3.1.1 Tổng quan bài toán FPM

Bài toán: Để khai thác GPARs từ đồ thị G, việc đầu tiên là tìm tập mẫu đồ thị phổ biến từ G. Điều này dẫn đến bài toán Khai thác Mẫu đồ thị Phổ biến (*Frequent Pattern Mining - FPM*) được khái quát như sau:

- **Input:** Một đồ thị G và ngưỡng hỗ trợ $\theta \leq max\{sum(a)|a \in L_A\}$.
- **Output:** Một tập hợp S bao gồm các mẫu đồ thị phổ biến Q của G sao cho $supp(Q, G) \geq \theta$ với mọi $Q \in S$.

Lưu ý, $sum(a)$ là tổng số đỉnh xuất hiện trong đồ thị mà có $A_m = a$ và $max\{sum(a)|a \in L_A\}$ là giới hạn trên của độ hỗ trợ có thể có trong một đồ thị G bất kì. Điều kiện $\theta \leq max\{sum(a)|a \in L_A\}$ được thêm vào để tránh trường hợp tầm thường $S = \Phi$.

3.1.2 Thuật toán tiến hành

Thuật toán FPMiner

Algorithm 2: FPMiner

Input: Đồ thị G, ngưỡng hỗ trợ θ.

Output: Một code graph G_c.

1 Khởi tạo một code graph $G_c = \Phi$;

2 Tỉa nhánh các nodes trong đồ thị không thỏa mãn ngưỡng hỗ trợ;

3 Khai thác các Mẫu đồ thị phổ biến gồm một cạnh đơn ($Q2$);

4 Tỉa nhánh các cạnh và các nodes tương ứng không thỏa ngưỡng hỗ trợ;

5 Với mỗi code α trong Q2 thực hiện $PatExt(\alpha, G, \theta, G_c)$;

6 **return** G_c.

Chi tiết thuật toán FPMimer được mô tả như sau:

1. Đầu tiên, khởi tạo một code graph với node root là None.
2. Sau đó, gom nhóm các nodes theo label. Các nodes thuộc nhóm label có số lượng không thỏa ngưỡng hỗ trợ sẽ được xóa khỏi đồ thị và các nodes còn lại trong đồ thị sẽ được đánh lại chỉ số.
3. Tiếp theo, tiến hành khai thác các cạnh thỏa ngưỡng hỗ trợ (dòng 3, $Q2$). Bước này được tiến hành bằng cách duyệt qua các cạnh đồ thị và gom nhóm các cạnh theo label nodes và label cạnh. Trong khóa luận này, chúng tôi tạm thời chưa xét đến label cạnh và mặc định label cạnh là -1.
4. Sau khi đã có $Q2$, tiến hành tỉa nhánh các cạnh và nodes tương ứng không xuất hiện trong $Q2$. Tương tự, cũng tiến hành đánh lại chỉ số các nodes.
5. Với mỗi DFS Code trong $Q2$, gọi hàm $PatExt$ để mở rộng Mẫu đồ thị phổ

biến. Chi tiết hàm *PatExt* được mô tả cụ thể bên dưới.

6. Cuối cùng, trả về code graph G_c sau khi đã cập nhật.

Mã giả của hàm PatExt

Algorithm 3: PatExt

Input: DFS Code α, đồ thị G, ngưỡng hỗ trợ θ, code graph G_c.

Output: Code graph G_c được cập nhật.

```
1  Sinh các ứng viên ứng với DFS Code α - cSet;
2  foreach DFS Code α_c ∈ cSet do
3      if α_c là Min DFS Code then
4          MF = localMine(α_c, G, θ);
5          if supp(MF, G) ≥ θ then
6              Cập nhật G_c;
7              PatExt(α_c, G, θ, G_c);
8          end
9      end
10 end
11 return G_c.
```

Chi tiết hàm *PatExt* được thực hiện như sau:

1. Sinh các ứng viên ứng với DFS Code α dựa trên chiến lược Mở rộng Đường đi Phải nhất. Tức là, đầu tiên, đối với mở rộng backward, thêm một cạnh giữa node phải nhất với các nodes còn lại trên đường đi phải nhất theo thứ tự từ node gốc đến xa node gốc nhất. Sau đó, mở rộng forward, nghĩa là

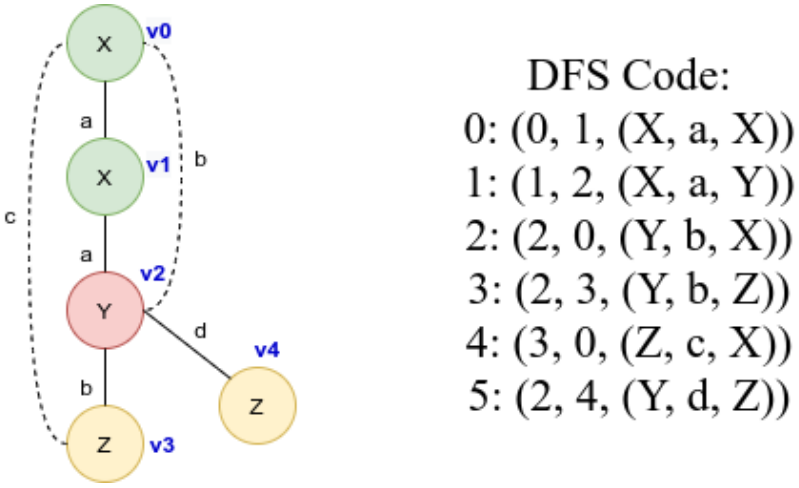
thêm một cạnh giữa các nodes trên đường đi phải nhất với một node mới, node mới này sẽ trở thành node phải nhất mới sau khi mở rộng. Lưu ý, đối với đồ thị có hướng thì từng chiến lược mở rộng backward và forward đều phải xét cả hai hướng. Nói cách khác, có bốn trường hợp mở rộng như sau: mở rộng backward với cạnh backward, mở rộng backward với cạnh forward, mở rộng forward với cạnh forward và mở rộng forward với cạnh backward. Đối với một node đang xét, chỉ mở rộng với các nodes có nhãn sao cho mẫu cạnh (nhãn node 1, nhãn cạnh, nhãn node 2) xuất hiện trong tập phổ biến Q2 tìm được ở trên, để tránh dư thừa ứng viên không cần thiết.

2. Ở bước này, cho vòng lặp duyệt qua tất cả ứng viên, ứng với mỗi ứng viên:
   - Đầu tiên ta kiểm tra đó có phải là Minimum DFS Code không, nếu không thì bỏ qua, xét tới ứng viên tiếp theo.
   - Chiến lược kiểm tra Minimum DFS Code dựa theo chiến lược trong thuật toán GSpan ([3]). Chiến lược này được tóm tắt như sau: Tiến hành lặp lại mở rộng code α' tuần tự từ tập rỗng, mỗi bước thêm một cạnh nhỏ nhất theo chiến lược Mở rộng Đường đi Phải nhất và so sánh với code α ở cạnh tương ứng, trả về $False$ nếu cạnh ở α lớn hơn α'. Kết thúc vòng lặp, trả về $True$.

     Ví dụ 9. Xét đồ thị ở Hình 2.3 (a), các bước tiến hành xây dựng và kiểm tra DFS Code ở Hình 2.3 (d) có phải là Minimum DFS Code hay không lần lượt như sau (DFS Code γ cần kiểm tra ở Hình 3.1):

     (a) Chuyển DFS Code γ ở Hình 2.3 (d) sang đồ thị g tương ứng để biết các cạnh và các nodes. Khởi tạo DFS Code $min = []$.

     (b) Từ đồ thị xây dựng được từ bước 1, ta xét các cạnh đơn của đồ thị g và chọn ra cạnh có thứ tự từ điển của mẫu (nhãn node 1,



Hình 3.1: *Kiểm tra Minimum DFS Code*

ID Node	Label	Nodes kề
0	X	[1, 2, 3]
1	X	[0, 2]
2	Y	[0, 1, 3, 4]
3	Z	[0, 2]
4	Z	[2]

Bảng 3.1: *Đồ thị g cho DFS Code γ*

nhãn cạnh, nhãn node 2) nhỏ nhất (được liệt kê ở Bảng 3.2). Trong trường hợp này, chính là $(0, 1, (X, a, X))$ và thêm nó vào min, lúc này $min = [(0, 1, (X, a, X))]$. So sánh ta được $min[0] = \gamma[0]$, do đó xét tiếp. Đối với Hình 2.3 (b), (c) thì ngay tại bước này, $min[0] < \alpha[0]$ và $min[0] < \beta[0]$ nên 2 DFS Code α và β trả về $False$ và thuật toán kiểm tra dừng ngay lập tức.

(c) Bước tiếp theo, xét đường đi phải nhất của min lúc này, kí hiệu $RMPath = [0, 1]$. Tìm tất cả đồ thị đẳng cấu của min trong đồ thị g. Đối với từng đồ thị đẳng cấu, tiến hành mở rộng backward, nếu

STT	Cạnh đơn mở rộng	Cạnh gốc
1	(0, 1, X, a, X)	(0, 1, X, a, X)
2	(0, 1, X, a, Y)	(1, 2, X, a, Y)
3	(0, 1, X, b, Y)	(2, 0, Y, b, X)
4	(0, 1, X, c, Z)	(3, 0, Z, c, X)
5	(0, 1, Y, b, Z)	(2, 3, Y, b, Z)
6	(0, 1, Y, d, Z)	(2, 4, Y, d, Z)

Bảng 3.2: *Các cạnh đơn của đồ thị g*

không có thì tiến hành mở rộng forward (các cạnh đơn mở rộng được liệt kê ở Bảng 3.3 rồi tìm cạnh mở rộng nhỏ nhất thêm vào min. Trường hợp này, chỉ có một đẳng cấu duy nhất là $(0, 1, (X, a, X))$, min được mở rộng như sau $min = [(0, 1, (X, a, X)), (1, 2, X, a, Y)]$. So sánh $min[1] = \gamma[1]$. Do đó, tiếp tục. Đối với các DFS Code khác nếu chúng không bằng nhau thì trả về $False$ và dừng thuật toán ngay lập tức.

STT	Cạnh mở rộng	Cạnh gốc	Loại mở rộng
1	(1, 2, X, a, Y)	(1, 2, X, a, Y)	forward
2	(0, 2, X, b, Y)	(2, 0, Y, b, X)	forward
3	(0, 2, X, c, Z)	(3, 0, Z, c, X)	forward

Bảng 3.3: *Mở rộng RightMost Path trong Minimum DFS Code*

(d) Tiếp tục như vậy cho đến khi trả về $False$ hoặc duyệt hết các cạnh của đồ thị g. Ở trường hợp này, ta duyệt hết thì thấy $min = \gamma$ nên γ chính là Minimum DFS Code của đồ thị ở Hình 2.3 (a). Chi tiết

các bước thực hiện kiểm tra Minimum DFS Code được trình bày ở Bảng 3.4.

Lần duyệt	*Min*	**RightMost Path**	**Cạnh mở rộng**
1		[]	**(0, 1, X, a, X)** (0, 1, X, a, Y) (0, 1, X, b, Y) (0, 1, X, c, Z) (0, 1, Y, b, Z) (0, 1, Y, d, Z)
2	**(0, 1, X, a, X)**	[0, 1]	**(1, 2, X, a, Y)** (0, 2, X, b, Y)) (0, 2, X, c, Z)
3	(0, 1, X, a, X) **(1, 2, X, a, Y)**	[0, 1, 2]	**(2, 0, Y, b, X)** (2, 3, Y, b, Z) (2, 3, Y, d, Z) (0, 3, X, c, Z)

4	(0, 1, X, a, X) (1, 2, X, a, Y) **(2, 0, Y, b, X)**	[0, 1, 2]	**(2, 3, Y, b, Z)** (2, 3, Y, d, Z) (0, 3, X, c, Z)
5	(0, 1, X, a, X) (1, 2, X, a, Y) (2, 0, Y, b, X) **(2, 3, Y, b, Z)**	[0, 1, 2, 3]	**(3, 0, Z, c, X)** (2, 4, Y, d, Z)
6	(0, 1, X, a, X) (1, 2, X, a, Y) (2, 0, Y, b, X) (2, 3, Y, b, Z) **(3, 0, Z, c, X)**	[0, 1, 2, 3]	**(2, 4, Y, d, Z)**
7	(0, 1, X, a, X) (1, 2, X, a, Y) (2, 0, Y, b, X) (2, 3, Y, b, Z) (3, 0, Z, c, X) **(2, 4, Y, d, Z)**	[0, 1, 2, 4]	

Bảng 3.4: *Các bước kiểm tra Minimum DFS Code*

- Đối với các ứng viên thỏa mãn Min DFS Code, ta tiến hành tìm các đồ thị đẳng cấu của nó; sau đó tính độ hỗ trợ tương ứng, nếu độ hỗ trợ của nó thỏa mãn ngưỡng hỗ trợ đã cho, thêm ứng viên đó vào code graph, tiến hành gọi đệ quy hàm $PatExt$ cho ứng viên đó. Việc sử dụng đệ quy dựa trên tính đơn điệu được mô tả ở phần 2.4, chỉ phát sinh ứng viên mới bằng cách mở rộng những mẫu đồ thị phổ biến tìm được. Ngoài ra, một chiến lược tỉa nhánh trong quá trình tìm các đồ thị đẳng cấu và tính độ hỗ trợ ứng với mẫu đồ thị ứng viên là dựa trên độ hỗ trợ s0 và bản so khớp của node có tần suất xuất hiện ít nhất của mẫu đồ thị con trước đó bằng cách lấy node đó làm gốc, tiến hành mở rộng, ứng với mỗi bản so khớp không mở rộng được thì trừ s0 đi một đơn vị cho đến khi s0 không thỏa ngưỡng hỗ trợ nữa thì dừng quá trình mở rộng.
- Tiếp tục như vậy cho đến khi không sinh thêm được ứng viên.

3. Kết quả cuối cùng trả về code graph đã cập nhật tất cả các DFS Code ứng với Mẫu đồ thị Phổ biến.

3.2 Khai thác Luật kết hợp

Algorithm 4: RuleGen

Input: Code graph G_c, ngưỡng tin cậy η.

Output: Một tập các GPARs.

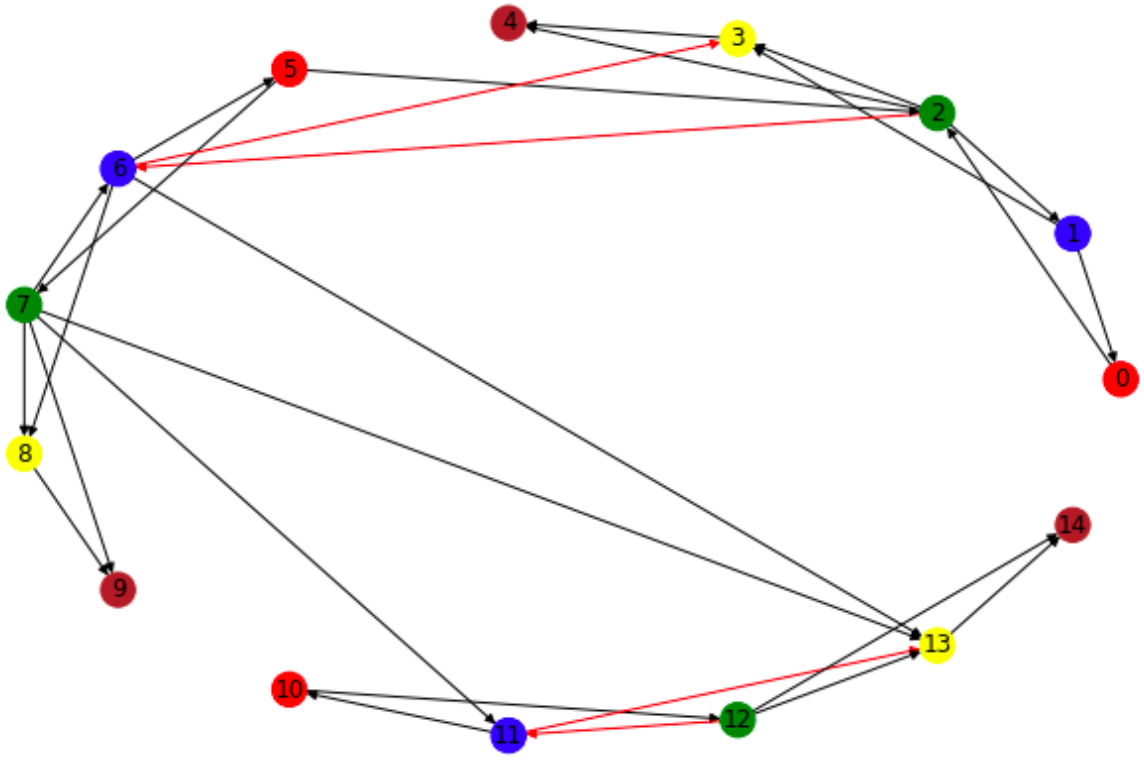
Khởi tạo tập $S = \Phi$, queue $q = \Phi$;
foreach *node* $v_\alpha \in G_c$ **do**
    $q = \Phi$;
    push v_α vào q;
    while $q \neq \Phi$ **do**
        node $q_R = q.pop()$;
        foreach $q_l \in ancestors(q_R)$ **do**
            push q_l vào q;
            Sinh q_r tương ứng;
            if q_r *liên thông và* $\frac{supp(q_R,G)}{supp(q_l,G)} \geq \eta$ **then**
                $S = S \cup \{(q_l \Rightarrow q_r)\}$;
            end
        end
    end
end
return S.

Chi tiết thuật toán RuleGen như sau:

1. Khởi tạo tập kết quả $S = \Phi$, hàng đợi $q = \Phi$.
2. Duyệt qua tất cả các nodes trong code graph G_c và ứng với mỗi node v_α tiến hành các bước sau:
   - Gán $q = \Phi$;
   - Thêm node v_α vào hàng đợi q.
   - Thực hiện vòng lặp sau cho đến khi $q = \Phi$:
     - Gán giá trị đầu tiên trong hàng đợi q cho node q_R.
     - Sinh tất cả các tổ tiên của q_R trong code graph G_c. Sau đó ứng với mỗi tổ tiên q_l, thêm q_l vào hàng đợi q và sinh ra kết quả q_r tương ứng của q_R và q_l (q_R, q_l, q_r lần lượt là Luật kết hợp, Tiền đề và Kết quả tương ứng trong GPAR). Chiến lược sinh tổ tiên của q_R được tiến hành bằng cách duyệt BFS của code graph G_c, ứng với mỗi DFS Code $dfsc$ có chiều dài bé hơn q_R, tiến hành kiểm tra có tồn tại đẳng cấu của $dfsc$ trong q_R hay không, nếu có thì $dfsc$ chính là một tổ tiên của q_R.
     - Kiểm tra q_r có liên thông hay không và độ tin cậy của luật q_R có thỏa mãn ngưỡng tin cậy (nói cách khác $\frac{supp(q_R,G)}{supp(q_l,G)} \geq$ ngưỡng tin cậy η). Nếu thỏa cả hai điều kiện trên, thêm luật $q_l \Rightarrow q_r$ vào tập S.
3. Cuối cùng trả về tập S chứa các GPARs thỏa mãn.

Ví dụ 10. Xem lại Hình 1.2. Cho ngưỡng tin cậy $\eta = 0.6$, xét mẫu đồ thị {(PM, DBA), (DBA, BA), (BA, PM), (BA, PRG), (DBA, PRG), (PRG, ST), (DBA, ST)} với $supp = 2$. Trong quá trình sinh luật, tìm thấy một tổ tiên q_l là {(PM, DBA), (DBA, PRG), (PRG, ST), (DBA, ST), (BA, PM)} với $supp = 3$, kết quả q_r tương ứng là {(DBA, BA), (BA, PRG)}. Như vậy, một GPAR có thể được sinh ra như sau: {(PM, DBA), (DBA, PRG), (PRG, ST), (DBA, ST), (BA, PM)} $\Rightarrow$ {(DBA, BA), (BA, PRG)} với độ tin cậy $conf = \frac{2}{3}$.

Hình 3.2 mô hình hóa lại đồ thị của Hình 1.2 và có thêm các cạnh có màu đỏ là cạnh mới đề xuất theo luật phát sinh ở Ví dụ 10. Ở đây, ta thấy có thể đề xuất {(Mat, Nancy), (Nancy, Dan)} hoặc {(Roy, Mary), (Mary, Tim)}.



Hình 3.2: *Ví dụ đề xuất cạnh mới của RuleGen*

CHƯƠNG 4. THỰC NGHIỆM

Toàn bộ mã nguồn được cài đặt bằng ngôn ngữ lập trình Python với Jupyter Notebook. Thực nghiệm được tiến hành trên Google Colab với RAM 25.51 GB, ổ đĩa 68.40 GB.

4.1 Dữ liệu thực nghiệm

Dữ liệu thực nghiệm gồm 5 tập:

1. Tập Cora: Dữ liệu thực tế được lấy từ ***networkrepository***[1].
2. Tập CL_10K_1d8_L5: Dữ liệu thực tế được lấy từ ***networkrepository***[2].
3. Tập SW_10000_6_0d3_L5: Dữ liệu thực tế được lấy từ ***networkrepository***[3].
4. Tập GenTest1: nhóm tự phát sinh bằng cách nhúng một số luật đồng thời thêm nhiễu để đồ thị mang tính tổng quát hơn.
5. Tập GenTest2: nhóm tự phát sinh bằng cách nhúng một số luật đồng thời thêm nhiễu để đồ thị mang tính tổng quát hơn.

Thông tin cơ bản của các tập dữ liệu trên được mô tả cụ thể ở Bảng 4.1. Các bộ dữ liệu này đều có thể xem là đồ thị vô hướng hoặc có hướng.

[1]NETWORK REPOSITORY http://networkrepository.com/cora.php
[2]NETWORK REPOSITORY http://networkrepository.com/CL-10K-1d8-L5.php
[3]NETWORK REPOSITORY http://networkrepository.com/SW-10000-6-0d3-L5.php

Tập dữ liệu	Số đỉnh	Số cạnh	Số label	Bậc
Cora	2708	5429	7	4
CL_10K_1d8_L5	10000	44896	5	8.98
SW_10000_6_0d3_L5	10000	30000	5	6
GenTest1	2000	2871	5	2.87
GenTest2	4000	5785	5	2.89

Bảng 4.1: *Các tập dữ liệu thực nghiệm*

4.2 Kết quả thực nghiệm FPM

Kết quả chạy thực nghiệm được tiến hành trên 5 bộ dữ liệu trên (xét đồ thị có hướng) với ngưỡng hỗ trợ và thời gian thực thi được trình bày ở Bảng 4.2 và Hình 4.1.

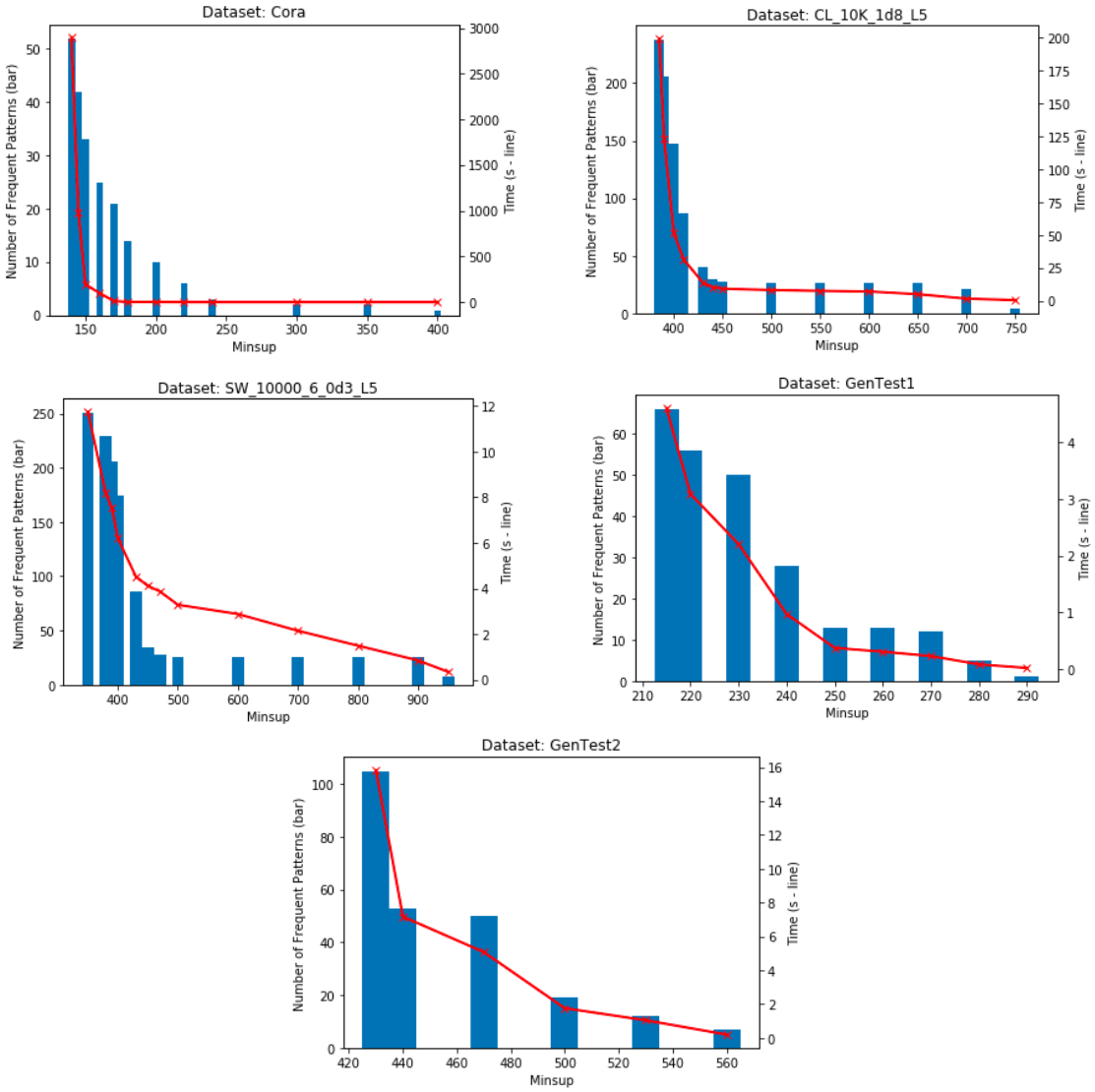
Tập dữ liệu	Minsup	Số lượng	Thời gian
Tập Cora	400	1	0.032819s
	300	2	0.103396s
	240	3	0.115851s
	220	6	0.171123s
	200	10	1.682202s
	180	14	1.749950s
	170	21	13.227389s
	160	25	98.895309s
	150	33	193.398976s
	145	42	986.224341s

Tập CL_10K_1d8_L5	750	4	0.487103s
	700	21	1.589396s
	450	27	7.221452s
	440	29	8.220977s
	430	40	10.423082s
	410	87	31.396106s
	400	147	51.373289s
	390	206	122.844527s
	385	238	199.051482s
Tập SW_10000_6_0d3_L5	950	8	0.348226s
	900	25	0.846386s
	470	28	3.872028s
	450	35	4.113262s
	430	86	4.522963s
	400	174	6.198317s
	390	206	7.538803s
	380	229	8.166873s
	350	251	11.744917s

Tập GenTest1	290	1	0.017429s
	280	5	0.080308s
	270	12	0.310278s
	240	28	0.968863s
	230	50	2.203916s
	220	56	3.085981s
	215	66	4.599453s
Tập GenTest2	560	7	0.196337s
	530	12	1.043546s
	500	19	1.754960s
	470	50	5.071292s
	440	53	7.168135s
	430	105	15.817664s

Bảng 4.2: *Kết quả thực nghiệm FPM - đồ thị có hướng*

Hình 4.1 minh họa trực quan lại các số liệu ở Bảng 4.2 với tên tập dữ liệu được ghi ở đầu mỗi biểu đồ, các cột (màu xanh) thể hiện số lượng tập phổ biến khai thác được, biểu đồ đường (màu đỏ) thể hiện thời gian thực thi ứng với mỗi ngưỡng hỗ trợ ở trục hoành.



Hình 4.1: *Tổng hợp Kết quả thực nghiệm FPM - đồ thị có hướng*

Đối với thuật toán Khai thác Mẫu đồ thị con Phổ biến trên đồ thị vô hướng, chúng tôi tiến hành thực nghiệm trên 5 tập dữ liệu thực Cora, CL_10K_1d8_L5, SW_1000_6_0d3_L5, GenTest1 và GenTest2. Kết quả chi tiết được thống kê trong Bảng 4.3 và Hình 4.2 (các biểu diễn tương tự như đồ thị có hướng được mô tả trước đó).

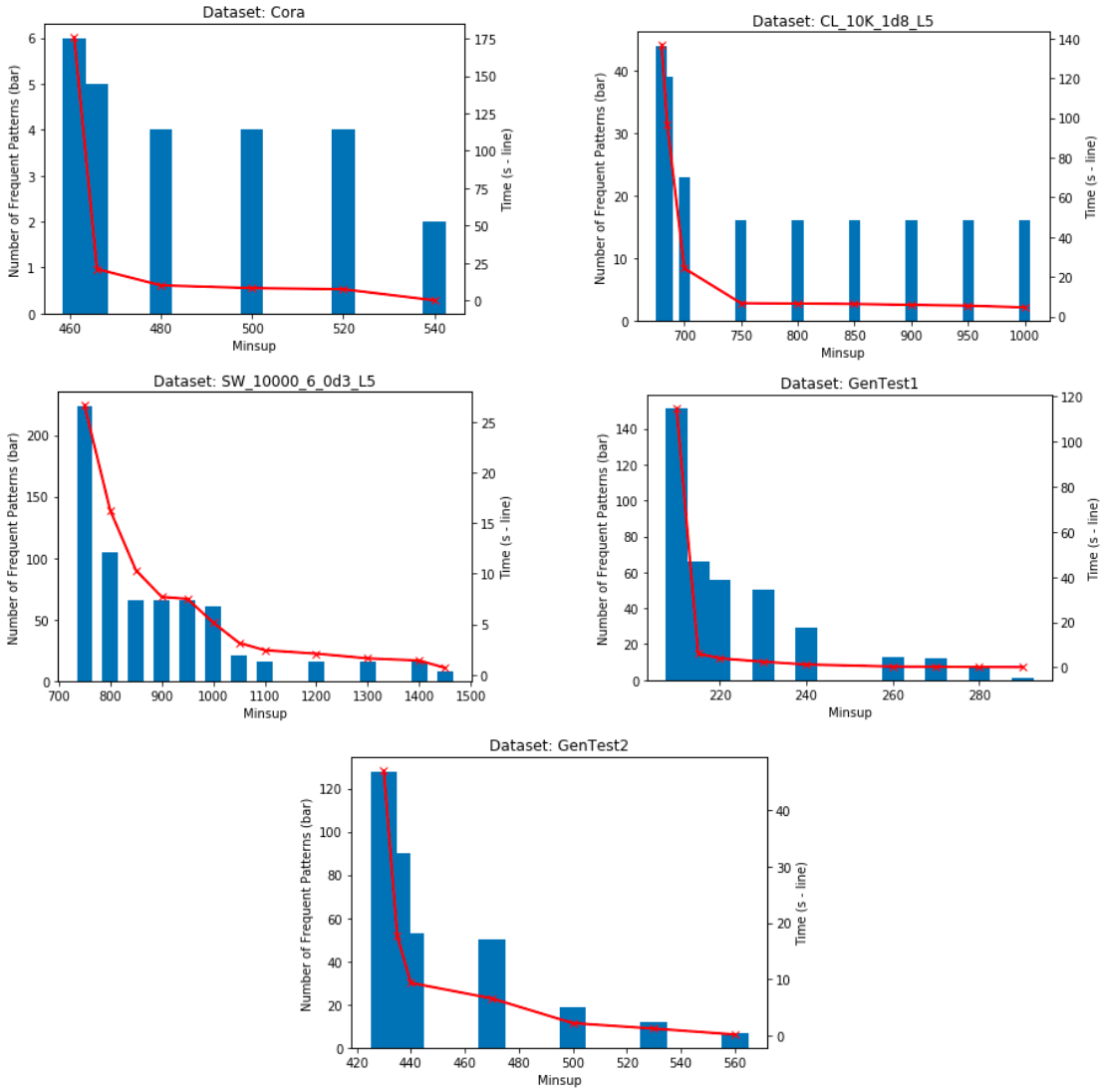
Tập dữ liệu	Minsup	Số lượng	Thời gian
Tập Cora	550	2	0.059198s
	530	4	3.482609s
	466	5	20.999120s
	461	6	175.861650s
Tập CL_10K_1d8_L5	1000	16	4.646169s
	700	23	24.327698s
	685	39	96.795707s
	680	44	136.677496s
Tập SW_10000_6_0d3_L5	1450	8	0.683120s
	1400	16	1.406610s
	1050	21	3.164529s
	1000	61	5.163074s
	950	66	7.511883s
	800	105	16.278127s
	750	224	26.705113s
Tập GenTest1	290	1	0.115696s
	280	7	0.189777s
	270	12	0.341727s
	240	29	1.283458s
	230	50	2.505852s
	220	56	4.025767s
	215	66	6.149690s
	210	151	114.88366s

Tập GenTest2	560	7	0.230510s
	530	12	1.273332s
	500	19	2.243390s
	470	50	6.618993s
	440	53	9.375655s
	435	90	17.664964s
	430	128	46.954458s

Bảng 4.3: *Kết quả thực nghiệm FPM - đồ thị vô hướng*

Nhận xét: Thông qua bảng thống kê kết quả trên thực nghiệm như trên, chúng ta có thể rút ra một số nhận xét như sau:

- Trên cùng một tập dữ liệu, giá trị *minsup* sẽ có một cận trên để bắt đầu cho ra tập phổ biến (trên ngưỡng này, sẽ không có tập phổ biến nào được tìm thấy). Tính từ ngưỡng giá trị này, giá trị *minsup* càng nhỏ, sẽ có xu hướng càng có nhiều tập phổ biến khai thác được, do có nhiều đẳng cấu phù hợp trong đồ thị đã cho. Cũng vì vậy, việc này làm thời gian khai thác khai thác cũng càng tăng dần khi *minsup* giảm đi.
- Các đồ thị có số lượng đỉnh và số lượng cạnh tương đương nhau, số lượng nhãn của đỉnh (không gian giá trị thuộc tính của đỉnh) càng lớn thì giá trị tối đa của *minsup* sao cho vẫn khai thác được tập phổ biến có xu hướng càng nhỏ. Nguyên nhân của vấn đề này là do, theo lý luận chung, khi số lượng nhãn càng nhiều, kích thước của các đẳng cấu tìm được có thể cũng lớn hơn. Trong khi đó, nếu số lượng nhãn nhỏ, các đẳng cấu tìm sẽ có thể có kích thước nhỏ hơn, dẫn đến so khớp trên đồ thị sẽ nhiều hơn (điều này



Hình 4.2: *Tổng hợp Kết quả thực nghiệm FPM - đồ thị vô hướng*

đồng nghĩa là giá trị *minsup* lớn nhất để tìm được tập phổ biến cao hơn).

- Xét các đồ thị cùng số lượng đỉnh, cùng số lượng nhãn, nếu số lượng cạnh càng lớn (đồ thị càng dày hay bậc trung bình của đỉnh trong đồ thị càng cao) thì *minsup* phù hợp cũng sẽ tăng theo.
- Trên cùng tập dữ liệu đồ thị, ngưỡng *minsup* tối đa thể khai thác ra tập phổ biến áp dụng cho giải thuật có hướng thường nhỏ hơn khi áp dụng giải thuật vô hướng. Nguyên nhân của vấn đề này là vì dữ liệu có hướng phân biệt 2 chiều của một cạnh, trong khi đó vô hướng không cần. Điều này dẫn

đến việc đẳng cấu sẽ không cần xét đến ràng buộc hướng nên số đẳng cấu sẽ nhiều hơn so với có hướng hay nói cách khác độ hỗ trợ của một mẫu thường sẽ cao hơn nếu xét vô hướng.

4.3 Kết quả thực nghiệm RuleGen

4.3.1 Kết quả thực nghiệm

Kết quả thực nghiệm thuật toán sinh luật kết hợp trên các tập dữ liệu (xét đồ thị có hướng) được thống kê trong Bảng 4.4.

Tập dữ liệu	**Minsup**	**Conf**	**Số lượng luật**	**Thời gian**
Cora	240	0.8	1	0.000726s
	200	0.9	3	0.001642s
		0.7	5	0.002304s
		0.5	9	0.001830s
	145	0.9	26	0.247965s
		0.7	57	0.259361s
		0.5	210	0.284210s
	140	0.9	36	0.480272s
		0.7	70	0.519609s
		0.5	89	0.548264s

CL_10K_1d8_L5	450	0.6	2	0.000965s
	440	0.6	6	0.001411s
	430	0.6	10	0.004283s
	400	0.6	12	0.032357s
		0.5	241	0.060180s
SW_10000_6_0d3_L5	450	0.5	4	0.004215s
	400	0.5	4	0.041981s
		0.45	168	0.053222s
		0.4	293	0.079396s
GenTest1	280	0.9	8	0.001079s
	240	0.9	74	0.045818s
		0.8	106	0.051006s
	215	0.9	254	0.655573s
		0.8	323	0.841221s
		0.7	333	0.784500s
GenTest2	560	0.9	11	0.000960s
	470	0.9	192	0.332807s
		0.8	150	0.390987s
	430	0.9	324	4.967680s
		0.8	525	5.668164s
		0.7	587	5.972687s

Bảng 4.4: *Kết quả thực nghiệm RuleGen - đồ thị có hướng*

Thực nghiệm thuật toán KTLKH trên đồ thị vô hướng được tiến hành trên 5 tập dữ liệu Kết quả chi tiết được thống kê trong Bảng 4.5.

Tập dữ liệu	**Minsup**	**Conf**	**Số lượng luật**	**Thời gian**
Cora	530	0.9	1	0.000329s
		0.7	2	0.000441s
		0.6	3	0.000441s
	466	0.9	1	0.00081s
		0.8	3	0.000618s
		0.6	6	0.000789s
	461	0.9	36	0.480272s
		0.7	70	0.519609s
		0.5	89	0.548264s
CL_10K_1d8_L5	700	0.6	14	0.002272s
	685	0.65	37	0.007314s
		0.6	46	0.007847s
	680	0.65	41	0.00838s
		0.6	56	0.00919s

SW_10000_6_0d3_L5	1050	0.7	10	0.001586s
	1000	0.7	64	0.014543s
		0.6	90	0.017369s
	800	0.7	136	0.071877s
		0.6	172	0.08269s
		0.5	232	0.091225s
	750	0.8	6	0.2068s
		0.7	362	0.27323s
		0.6	398	0.30781s
		0.5	651	0.37583s
GenTest1	280	0.9	8	0.001022s
	240	0.9	79	0.052428s
		0.8	115	0.060664s
	215	0.9	319	0.681755s
		0.8	418	0.835909s
		0.7	470	00.873516s
GenTest2	560	0.9	11	0.000960s
	470	0.9	224	0.334777s
		0.8	304	0.404478s
	430	0.9	460	15.689276s
		0.8	760	18.57670s
		0.7	865	20.525682s

Bảng 4.5: *Kết quả thực nghiệm RuleGen - đồ thị vô hướng*

4.3.2 Đánh giá kết quả

Do tính chất khá mới lạ của bài toán, đến thời điểm hiện tại vẫn chưa có độ đo chính xác cho luật kết hợp trên đồ thị, nên chúng tôi đánh giá kết quả của luật kết hợp dựa trên việc tham khảo các phương pháp đánh giá độ chính xác của các bài toán tương tự khác trên đồ thị như: gom cụm (clutering), dự đoán cạnh (link prediction), ...

Để đánh giá độ chính xác của các cạnh đề xuất được, chúng tôi dựa trên ý tưởng của các bài toán gom cụm và đề xuất quan hệ trong đồ thị không có nhãn. Chúng tôi đánh giá độ chính xác dựa trên các cạnh được đề xuất từ luật kết hợp đã khai thác bằng cách đối chiếu chúng với các cạnh được đề xuất theo phương pháp tính độ tương tự Jaccard và Simrank.

- Độ đo Jaccard định nghĩa độ tương đồng giữa 2 đỉnh u, v trong một đồ thị G theo công thức sau:

$$Sim_j(v_1, v_2) = \frac{N(v_1) \cap N(v_2)}{N(v_1) \cup N(v_2)} \tag{4.1}$$

  Trong đó, $N(v)$ là tập các đỉnh có quan hệ (liên kết) với đỉnh v.

- Simrank định nghĩa độ tương quan giữa 2 đỉnh u, v trong đồ thị như sau:

$$Sim_s(v_1, v_2) = C\frac{\Sigma_{v_1' \in N(v_1)}\Sigma_{v_2' \in N(v_2)} Sim(v_1', v_2')}{k_{v_1}.k_{v_2}} \tag{4.2}$$

  Trong đó, $C \in [0, 1]$ là hệ số phân rã, k_v là số bậc của đỉnh v trong đồ thị.

Việc đánh giá được thực hiện theo phương pháp **K-Fold Cross Validation**, ở mỗi Fold được tiến hành đánh giá theo các bước sau:

1. Trên phần dữ liệu đồ thị dùng để train: khai thác tập phổ biến và sinh luật từ các tập phổ biến này.
2. Trên dữ liệu đồ thị test: đề xuất các cạnh mới dựa trên các luật đã khai thác được từ tập train. Tính độ tương quan giữa các đỉnh theo độ đo Jaccard/Simrank và chọn ra top-L cặp đỉnh có độ tương quan cao nhất (không phải là cạnh đã tồn tại trong đồ thị) xem như là tập các đỉnh đề xuất đúng. Lúc này độ chính xác của các cạnh được đề xuất từ luật kết hợp khai thác được định nghĩa như sau:

$$Acc = \frac{L_r}{L} \tag{4.3}$$

   Trong đó, L_r là số lượng cạnh đề xuất đúng, với cạnh đề xuất đúng được định nghĩa là cạnh đề xuất từ luật kết hợp trùng với cạnh đề xuất bởi Jaccard/Simrank.
3. Độ chính xác *Acc* cuối cùng được tính bằng trung bình độ chính xác *Acc* của các Fold.

Phương pháp đánh giá này không thật sự công bằng cho việc đánh giá độ chính xác của bài toán hiện tại. Thậm chí, với bài toán hiện tại, nhãn của đỉnh là thành phần cốt lõi để khai thác mà phương pháp tính Jaccard/Simrank lại không quan tâm đến chúng. Tuy nhiên, phương pháp đánh giá này thể hiện phần nào tính đúng đắn của các cạnh được đề xuất bởi luật khai thác được.

Kết quả thực nghiệm đánh giá độ chính xác trên các tập dữ liệu (xét đồ thị có hướng) được thống kê trong Bảng 4.6.

Top-L		50	100	150	200	250	300	350
Cora	Jaccard	11	7.75	8.5	10.25	12.4	12.08	11.07
	Simrank	13	23.25	18.5	20.75	19.6	20.67	21.14
CL_10K _1d8_L5	Jaccard	26.43	28.94	30.87	31.05	31.32	31.2	31.17
	Simrank	20.83	20.42	22.27	22.25	22.24	22.8	23.29
SW_10000 _6_0d3_L5	Jaccard	18	19.57	19.81	19.21	20.46	20.48	20
	Simrank	14.29	13.29	13.72	13.5	13.43	13.91	13.83
GenTest1	Jaccard	21.5	25	24.5	23.75	23.1	19.92	17.07
	Simrank	19.5	17	17.83	18.75	18.3	18.83	18.57
GenTest2	Jaccard	22	23.5	24.67	24.5	24.5	24.58	24.14
	Simrank	12.5	12.5	13.17	13.75	13	13.67	14.36

Bảng 4.6: *Đánh giá kết quả theo Jaccard và Simrank - đồ thị có hướng (Đơn vị: %)*

Kiểm tra lại độ chính xác của các cạnh được đề xuất khi áp dụng giải thuật vô hướng trên các bộ dữ liệu trên với kết quả được thống kê chi tiết trong Bảng 4.7.

Top-L		50	100	150	200	250	300	350
Cora	Jaccard	16	22.5	21.33	19	21.0	20.33	20.29
	Simrank	17	18.5	19.5	20.88	20.4	19.83	18.71
CL_10K _1d8_L5	Jaccard	39	41.4	42.33	42.15	42.36	42.17	42.34
	Simrank	35.6	34.8	34.33	33.5	33.56	33.3	33.26
SW_10000 _6_0d3_L5	Jaccard	40.57	38	36.76	37.79	37.31	37.1	36.61
	Simrank	27.43	31	30.86	31.29	30.4	30.1	30.24
GenTest1	Jaccard	25.5	23.25	22.17	21.88	21.8	21.08	21.93
	Simrank	23	22.75	22.17	23.5	23.1	23.67	22.71
GenTest2	Jaccard	22	20.25	18.67	18	19.5	19.75	20.29
	Simrank	19	18.25	19	20.25	19.7	20.25	20.5

Bảng 4.7: *Đánh giá kết quả theo Jaccard và Simrank - đồ thị vô hướng (Đơn vị: %)*

Nhận xét chung: Qua phương pháp đánh giá các cạnh được đề xuất dựa trên hai độ đo Jaccard và Simrank, chúng ta có thể thấy rằng độ chính xác của các cạnh được đề xuất có xu hướng cao hơn khi áp dụng giải thuật vô hướng vào dữ liệu (so với có hướng). Đặc biệt, với dữ liệu kích thước càng lớn, sự chênh lệch này sẽ dễ nhận thấy hơn. Xu hướng này xảy ra vốn phần lớn là do độ đo Jaccard/Simrank sẽ phù hợp hơn với dữ liệu vô hướng trong bài toán này hơn.

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Bài toán KTLKH dựa trên Mẫu đồ thị con phổ biến là một bài toán quan trọng và có nhiều ứng dụng trong thực tế, đã và đang được các nhà nghiên cứu cũng như doanh nghiệp quan tâm nhiều hơn. Tuy nhiên, bài toán này vẫn còn rất nhiều thách thức về mặt hiệu năng, cách xây dựng cấu trúc dữ liệu đồ thị cũng như về mặt dữ liệu lớn.

Trong khóa luận này, chúng tôi đã tiến hành xây dựng được thuật toán Khai thác Mẫu đồ thị phổ biến **FPMiner** và Khai thác Luật kết hợp Mẫu đồ thị **ruleGen** dựa trên kết hợp ý tưởng của các bài báo khoa học liên quan đã đề xuất trước đó.

Thuật toán đã tiến hành cho kết quả với chi phí thời gian và không gian chấp nhận được. Ngoài ra, các Luật sinh ra (các cạnh mới được đề xuất) cũng có hiệu quả thực tế chấp nhận được. Hạn chế ở thuật toán này là về mặt tốc độ thực thi. Đối với các dữ liệu lớn và ngưỡng hỗ trợ càng nhỏ, thuật toán chạy khá lâu vì chỉ khai thác tuần tự trên một tập dữ liệu lớn và chiến lược tỉa nhánh chưa thực sự tối ưu.

5.2 Hướng phát triển

Trong tương lai với điều kiện cho phép, chúng tôi sẽ tiếp tục cải tiến và tối ưu thời gian truy xuất cũng như giảm bộ nhớ lưu trữ của thuật toán hiện tại với phương pháp lưu trữ dữ liệu mới.

Một hướng tiếp cận khác mà chúng tôi dự định tiến hành là sử dụng lập trình song song trên GPU để tối ưu thời gian thực thi.

Ngoài ra, chúng tôi dự định sẽ tiếp cận theo hướng Big Data sử dụng cơ sở dữ liệu phân tán để tăng tốc độ khai thác Mẫu đồ thị Phổ biến kết hợp với thuật toán khai thác cục bộ phát triển từ thuật toán chúng tôi đã tiến hành trong khóa luận này với hướng tối ưu đã trình bày ở trên.

Cuối cùng, chúng tôi sẽ áp dụng trên nhiều miền dữ liệu khác nhau và xây dựng một hệ thống đề xuất.

TÀI LIỆU THAM KHẢO

[1] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *European conference on principles of data mining and knowledge discovery*, pp. 13–23, Springer, 2000.

[2] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 313–320, IEEE, 2001.

[3] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 721–724, IEEE, 2002.

[4] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 517–528, 2014.

[5] X. Wang, Y. Xu, and H. Zhan, "Extending association rules with graph patterns," *Expert Systems with Applications*, vol. 141, p. 112897, 2020.

[6] GraMi. `https://github.com/ehab-abdelhamid/GraMi`, 2015.

[7] Zaki and M. Jr. `https://www.cs.rpi.edu/~zaki/DMML/slides/pdf/ychap11.pdf`, 2014.

[8] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1502–1513, 2015.

[9] X. Wang and Y. Xu, "Mining graph pattern association rules," in *International Conference on Database and Expert Systems Applications*, pp. 223–235, Springer, 2018.