Graph Mining

Team 07

Mid Term Report

Student ID	Fullname
20127038	Nguyễn Phước Gia Huy
20127088	Nguyễn Thiện Hoàng Trí
20127474	Đoàn Ánh Dương
20127524	Phan Tuấn Khải

fit@hcmus

Contents

1 Content **3**

2 Task 1 **4**

2.1 Graph Classification 4

2.1.1 Giới thiệu 4

2.1.2 Graph Kernels 4

2.1.3 Ví dụ tổng quát 7

2.2 Link Prediction 10

2.2.1 Giới thiệu 10

2.2.2 Social Network 10

2.2.3 Link Prediction 10

2.2.4 Phương pháp dựa trên đỉnh láng giềng 11

2.2.5 Phương pháp dựa trên tất cả đường đi 15

2.3 Graph Indexing 18

2.3.1 Giới thiệu 18

2.3.2 Chỉ mục đồ thị dựa trên đặc trưng (feature-based graph indexing . 18

2.4 Community Detection 23

2.4.1 Định nghĩa bài toán 23

2.4.2 Ứng dụng 23

3 ProtGNN: Towards Self-Explaining Graph Neural Networks **40**

3.1 Introduction 40

3.2 Problem Statement 41

3.2.1 Kiến trúc ProtGNN 41

3.2.2 Monte-Carlo Tree Search 43

3.2.3 Learning Object 43

3.2.4 Prototype Projection 44

3.2.5 ProtGNN+ (Lấy mẫu đồ thị con có điều kiện) 46

3.2.6 Theorem on Subgraph Sampling 47

3.2.7 Thủ tục huấn luyện 51

3.2.8 ProtGNN cho các nhiệm vụ đồ thị tổng quát 51

3.3 Related Work 52

3.4 Experimental Evaluation 54
3.4.1 Chuẩn bị dữ liệu 54
3.4.2 Kết quả 55
3.4.3 Giải thích quy trình 56
3.4.4 Trực quan hóa Prototype 57
3.4.5 Kết luận 57

1

Content

Bảng phân công công việc dự kiến:

	Nguyễn Phước Gia Huy	**Nguyễn Thiện Hoàng Trí**	**Đoàn Ánh Dương**	**Phan Tuấn Khải**
Week1 +Week2 26/6 – 8/7	Tìm hiểu về graph classification: Graph Kernals & Graph Boosting	Tìm hiểu về link prediction	Tìm hiểu về Graph Ranking	Tìm hiểu về Community Detection
Week3 10/7 – 15/7	Thực hiện ví dụ 1 về graph classification	Thực hiện ví dụ 1 về Link Prediction	Thực hiện ví dụ 1 về graph ranking	Thực hiện ví dụ 1 về community dectection
Week4 17/7 – 22/7	Thực hiện ví dụ 2 về graph classification	Thực hiện ví dụ 2 về link prediction	Thực hiện ví dụ 2 về graph ranking	Thực hiện ví dụ 2 về community detection
Week 5 23/7-30/7	Tổng hợp thông tin và ví dụ về graph classification	Tổng hợp thông tin và ví dụ về link prediction	Tổng hợp thông tin và ví dụ về graph ranking	Tổng hợp thông tin và ví dụ về community detection
Week6 31/7 – 5/8	Viết báo cáo tổng hợp cho task1, và sửa chữa bổ sung nếu còn thiếu xót	Sửa chữa bổ sung		

Table 1: Bảng phân công công việc dự kiến

2 Task 1

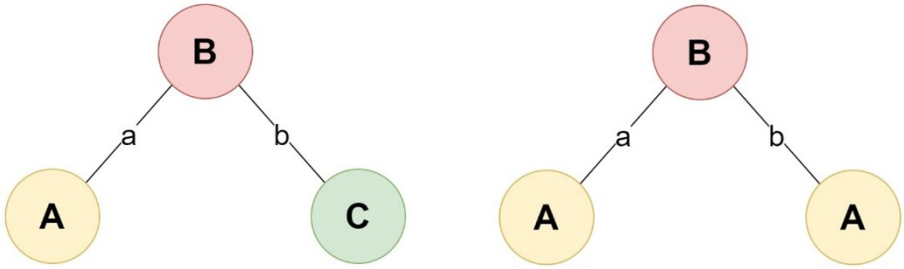
2.1 Graph Classification

2.1.1 Giới thiệu

Cụm từ graph classification có thể hiểu với 2 nhiệm vụ khác nhau. Nhiệm vụ một là phân loại nhãn dán cho toàn graph. Hoặc nhiệm vụ hai, là phân loại (gán) nhãn dán cho từng node trong graph. Nhưng ở đây chúng ta xem cụm từ đại diện cho nhiệm vụ đầu tiên, và nhiệm vụ 2 chúng ta sẽ dùng thuật ngữ "label propagation", thuộc công việc "Community Detection". "Graph classification có thể giải quyết bằng có giám sát hoặc không giám sát. Đối với phương pháp học không giám sát, ta sẽ phân loại graph thành một số loại nhất định. Còn với phương pháp học có giám sát thì sẽ có giá trị đích cụ thể và nhãn cụ thể. Phương pháp học có giám sát sẽ được coi là cơ sở hơn so với học không giám sát, vì chúng ta có thể giải quyết bài toán không giám sát bằng phương pháp có giám sát thông qua mô hình xác suất của nhãn tiềm ẩn. Những để có thể áp dụng thuật toán học có giám sát hay không giám sát, chúng ta đều cần phân tích các đặc điểm của đồ thị để tạo ra các đặc điểm để căn cứ phân lớp. Dựa vào [1] "Managing and Mining Graph Data" của Charu và Haixun, chúng ta có 2 phương pháp chính đó là Graph Kernels và Graph Boosting. Trong bài báo cáo này, chúng sẽ trình bày phương pháp Graph Kernels.

2.1.2 Graph Kernels

Kernel graph đề suất dựa trên ý tưởng của "radom walking". Bằng cách lặp lại "random walk" với điểm khởi tạo và đích ngẫu nhiên, việc có được xác suất của tất cả bước có thể là khả thi. Và ý tưởng cốt lỏi của graph kernel là thu độ tương đồng của 2 đồ thị bằng cách so sánh bảng xác suất. Cho 2 đồ thị G và G', bây giờ chúng ta sẽ đi tính "label sequence kernel" (graph kernel) của 2 đồ thị.



(a) G (b) G'

Random Walks

Lý thuyết

Chúng ta trích xuất các đặc trưng (chuỗi có nhãn) từ đồ thị G thực hiện: Đầu tiên, chúng ta lấy mẫu một đỉnh $x_1 \in \chi$, χ là tập các đỉnh, bằng phân phối xác suất $p_s(x_1)$. Tiếp tục, bước thứ i, đỉnh tiếp theo $x_i \in \chi$ được lấy tuân theo xác suất chuyển tiếp $p_t(x_i|x_{i-1})$ hoặc "random walk" kết thúc lại $x_{i-1}vixcsutp_q(x_{i-1})$. Như vậy chúng ta có tại bước i:

$$\sum_{k=1}^{|\chi|} p_t(x_k|\ x_{i-1}) + p_q(x_{i-1}) = 1$$

Với p_s, p_t: là phân phối đều; p_q: là xác suất nhỏ không đổi. Từ "random walk", chúng ta thu được chuỗi các đỉnh gọi là đường đi (path). $X = (x_1, x_2, \ldots,\ x_l)$, l: là độ dài của dãy X. Như vậy, ta có xác suất của đường đi X là:

$$p(X|\ G) = p_s(x_1) \prod_{i=2}^{l} p_t(x_i|\ x_{i-1})\, p_q(x_l)$$

Ta có chuỗi nhãn: $h = (h_1, h_2, \ldots, h_{2l-1}) \in (\Sigma_V \Sigma_E)^{l-1} \Sigma_V$, với đường đi X ta có chuỗi nhãn là: $h_X = (v_{x_1},\ e_{x_1x_2}, v_{x2}, e_{x_2x_3}, \ldots,\ v_{x_l})$.

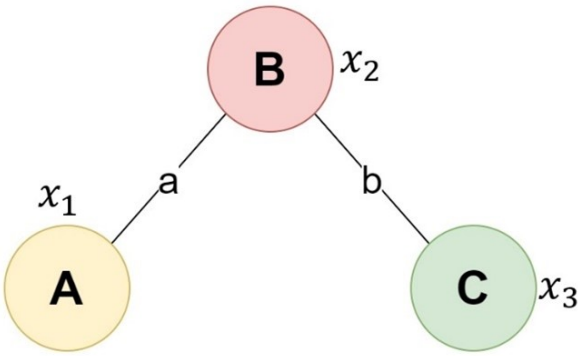
Thay đường đi bằng chuỗi nhãn, ta có hàm xác suất:

$$p(h|\ G) = \sum_X \delta(h = h_X) \bullet \left(p_s(x_1) \prod_{i=2}^{l} p_t(x_i|\ x_{i-1})\, p_q(x_l) \right)$$

Với δ: là hàm sẽ trả về 1 nếu thỏa, ngược lại là 0.

Áp dụng

Với đồ thị G, chúng ta random work dựa trên thuật toán DFS, và với điểm xuất phát là A. (Do sử dụng thuật toán DFS nên chúng ta sẽ không xét trường hợp quay lại – đi lùi) .



Khi duyệt đồ thị, ta tìm thấy 2 đường đi là $(v_{x_1}, e_{x1,x_2}, v_{x_2})$ và $(v_{x_1}, e_{x1,x_2}, v_{x_2}, e_{x_2,x_3}, v_{x_3})$. Trong lúc duyệt chúng ta sẽ tính $p(X|\ G)$ đồng thời, sau đó ta sẽ chuyển X thành chuỗi nhãn h , và tính tổng xác suất các h giống nhau, ta sẽ thu được $p(h|\ G)$ của h đó. Nhưng trong đồ thị này không xuất hiện các đường đi có nhãn chuỗi giống nhau nên $p(X|\ G)$ sẽ bằng $p(h|\ G)$. Như vậy ta sẽ được bảng sau:

Path- Đường đi	$p(X\|\ G)$	**Label Sequence - chuỗi nhãn**	$p(h\|\ G)$
$(v_{x_1}, e_{x1,x_2}, v_{x_2})$	$p_s(x_1)\, p_t(x_2\|\ x_1)\, p_q(x_2) = \frac{1}{3} * 1 * \frac{1}{2} = \frac{1}{6}$	(A, a, B)	$\frac{1}{6}$
$(v_{x_1}, e_{x1,x_2}, v_{x_2}, e_{x_2,x_3}, v_{x_3})$	$p_s(x_1)\, p_t(x_2\|\ x_1) \setminus p_t(x_3\|\ x_2)\, p_q(x_3) = \frac{1}{3} * 1 * \frac{1}{2} * 1 = \frac{1}{6}$	(A, a, B, b, C)	$\frac{1}{6}$

Label Sequence Kernel (Kernel chuỗi nhãn)

Lý thuyết

Bây giờ chúng ta sẽ xác định kernel k_z giữa 2 chuỗi nhãn h và h'.

Chúng ta giả sử rằng 2 hàm kernel nhãn, là kernel nhãn đỉnh $k_v(v, v')$ và kernel nhãn cạnh $k_e(e, e')$, đã được xác định và không âm. Ví dụ, $k_v(v, v')$, kernel sẽ trả về 1 nếu 2 nhãn giống nhau, và ngược lại là 0.

- Nếu nhãn rời rạc $k_v(v, v') = \delta(v = v')$.
- Nếu nhãn xác định trên $\mathbb{R}$, thì ta sẽ sử dụng Gaussian kernel:

$$k_v(v, v') = \exp\left(\frac{-||v - v'||^2}{2\sigma^2}\right)$$

Tương tự như vậy với kernel cạnh $k_e(e, e')$.

Dựa vào kernel nhãn đỉnh và cạnh, chúng ta có thể định nghĩa kernel cho chuỗi nhãn. Nếu 2 chuỗi h và h' cùng đồ dài, thì kernel chuỗi được định nghĩa như tích của các kernel nhãn.

$$k_z(h,\ h') = k_v(h_1, h'_1) \prod_{i=2}^{l} k_e\left(h_{2i-2}, h'_{2i-2}\right) k_v\left(h_{2i-1}, h'_{2i-1}\right)$$

Nếu 2 chuỗi có kích thước khác nhau thì kernel chuỗi sẽ trả về 0, $k_z(h,\ h') = 0$.

Cuối cùng, kernel chuỗi nhãn được định nghĩa như kì vọng của k_z của tất cả $h \in G$ và $h' \in G'$.

$$(G, G') = \sum_{h} \sum_{h'} k_z(h, h')\, p(h|G)\, p(h'|G')$$

Áp dụng

Tượng tự với cách tính $p(h|G)$, chúng ta tính các $p(h'|G')$ cho đồ thị G'.

Với đồ thị G', chúng ta có các nhãn chuỗi h' là: (A, a, B), (A, a, B, b, A).

Về mặt cơ bản việc tính $k_z(h, h')$, tức là kiểm tra xem các chuỗi nhãn có giống nhau không ($k_z(h, h')$ bằng 1 khi cặp chuỗi nhãn giống nhau và bằng 0 khi cặp chuỗi nhãn khác nhau). Vậy nên chúng ta chỉ phát hiện $h = h'$ duy nhất là (A, a, B).

Vậy ta có:

$$k(G, G') = k_z(h, h')\, p(h|G)\, p(h'|G') = 1 * \frac{1}{6} * \frac{1}{6} = \frac{1}{36}$$

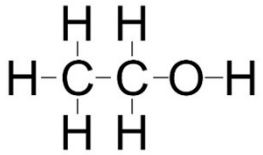
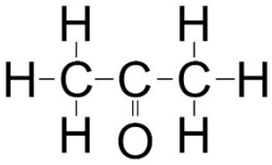
2.1.3 Ví dụ tổng quát

Cho một tập dữ liệu về hợp chất hóa học, đại diện cho đồ thị với node là nguyên tử và cạnh là liên kết hóa học với các chất như $C_2H_5OH, C_2H_3COCH_3, C_3H_6OH, \ldots$ Nhiệm vụ xây dựng mô hình phân loại ancol và xeton.

Nhóm sẽ trình bày trình tự cơ bản của graph classification với phương pháp graph kernel. Như đã nói bài báo cáo này chỉ tập trung vào graph kernel, nên nhóm chỉ trình bày chi tiết phần graph kernel, cụ thể là phần tính toán kernel của 2 hợp chất C_2H_5OH và $C_2H_3COCH_3$, và nói sơ lược các phần còn lại.

Chuyển thành dataset phù hợp

Từ 2 hợp chất trên, chuyển thành dữ liệu đồ thị:

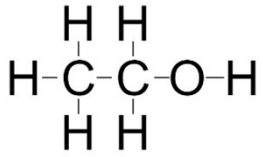
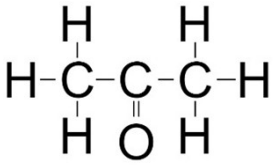
Hợp chất	Cấu tạo hóa học	Trực quan	Nhãn
1	C_2H_5OH		Nhãn đỉnh: H, C, H, H, C, H, H, O, H. Nhãn cạnh: Đơn, Đơn, Đơn, Đơn, Đơn, Đơn, Đơn, Đơn. Tên lớp huấn luyện: Ancol.
2	$C_2H_3COCH_3$		Nhãn đỉnh: H, C, H, H, C, O, C, H, H, H Nhãn cạnh: Đơn, Đơn, Đơn, Đơn, Đôi, Đơn, Đơn, Đơn, Đơn. Tên lớp huấn luyện: Xeton.

Tính Graph Kernel

Thực hiện tính kernel, ta sử dụng phối hợp thuật toán DFS với mỗi điểm, và để cho thực hiện nhanh chóng và nhỏ gọn, nhóm quyết định giới hạn số bước của random walk là 2. Chúng ta sẽ xét trường hợp chuỗi nhãn $(A, a, B) = (B, a, A)$. ($p_q(x_l)$ sẽ không tính trường hợp lùi lại, chỉ xét tiến lên hoặc dừng lại).

Chạy random walk tại mỗi node và tính xác xuất $p(X|G)$ của đường đi đó trong lúc bước. Chuyển đường đi X thành chuỗi nhãn, và tính tổng xác xuất của các đường đi có chuỗi nhãn giống nhau.

Dựa vào điều trên ta tìm được các chuỗi nhãn của đồ thị, với bảng như sau:

Hợp chất	Trực quan	Các chuỗi nhãn		Xác suất của các chuỗi nhãn
1		h	(C,đơn,H)	$\frac{7}{36} = 0.194$
			(C,đơn,O)	$\frac{1}{36} = 0.028$
			(C,đơn,C)	$\frac{1}{72} = 0.014$
			(O,đơn,H)	$\frac{5}{72} = 0.069$
2		h'	(C,đơn,H)	$\frac{1}{5} = 0.2$
			(C,đôi,O)	$\frac{1}{15} = 0.067$
			(C,đơn,C)	$\frac{1}{30} = 0.033$

Tính Kernel chuỗi nhãn (Label Sequence Kernel):

Tính $k_z(h, h')$ với các giá trị rạc, dễ hiểu hơn là kiểm tra độ tương đồng của các chuỗi nhãn nếu giống nhau thì trả về 1 (True), nếu khác nhau thì trả về 0 (False).

Ta phát hiện có 2 cặp chuỗi nhãn của G và G' giống nhau là cặp (C,đơn,H) và (C,đơn,C). Ta có Label Sequence Kernel:

$$k(G, G') = 0.194 * 0.2 + \ 0.014 * 0.033 = 0.039$$

Như vậy ta đã tính xong kernel chuỗi nhãn của của 1 cặp đồ thị, tương tự như vậy với các đồ thị còn lại, ta sẽ lập được ma trận kernel:

	Chất 1	**Chất 2**	**...**	**Chất n**
Chất 1	1	0,039	...	0,...
Chất 2	0,039	1		0,...
...	...		...	
Chất n	0,...	0,...	0,...	1

Áp dụng vào mô hình học máy

Sử dụng mô hình học có giám sát SVM, Random Forest,... để huấn luyện một classification bằng sử dụng nhãn dán. Kernel chuỗi nhãn sẽ được sử dụng để tính toán sự tương đồng giữa các chuỗi nhãn của các mẫu huấn luyện. Sử dụng: khi có hợp chất mới, chúng ta chuyển đổi cấu trúc phân tử của nó thành các chuỗi nhãn dựa trên mẫu dữ liệu và tính toán kernel và áp mô hình vào để phân loại.

2.2 Link Prediction

2.2.1 Giới thiệu

Hầu hết các nền tảng công nghệ xã hội hiện nay như Facebook, Instagram được hình thành như các đồ thị. Các users là những người như là các nodes được liên kết với nhau trong một cộng đồng mạng.Để làm việc với những đồ thị và mạng trên, vì vậy mà ta cần các cách tiếp cận, công cụ hay các thuật toán mới thay vì những phương pháp học máy truyền thống. Vì thế mà Link prediction chính là một trong các giải pháp cho vấn đề social network.

Vậy trong bài phân tích này, ta sẽ tìm hiểu chủ yếu về Link Prediction về cách hoạt động, các phương pháp và tiện lợi cũng như bất lợi của nó. Nhưng trước hết ta sẽ phải tìm hiểu sơ qua về Social Network là như thế nào trong phân tích và khai thác.

2.2.2 Social Network

Social Network là đại diện các mối quan hệ thiết yếu giữa các thực thể xã hội như con người, công ty hay các tổ chức chính phủ.

Công việc của một nhà khoa học dữ liệu chính là khai thác các hoạt động tương tác giữa các thực thể xã hội qua lượng lớn dữ liệu đến từ việc đăng bài, nhắn tin, comment và shares các tin tức, v.v. Dựa trên những dữ liệu đó mà ta phải thực hiện các tác vụ khai thác phân tích như thu thập dữ liệu từ các trang web mạng xã hội online và sử dụng những dữ liệu đó để ra quyết định trong doanh nghiệp.

Tóm lại tiện ích của việc phân tích mạng xã hội có thể kể đến là giúp ta có thể hiểu rõ về đối tượng khách hàng của mình, giúp ta phân khúc được các đối tượng khác hàng phù hợp v.v

2.2.3 Link Prediction

Khái niệm: Mục tiêu của Link Prediction là nhận diện/dự đoán một liên kết giữa một cặp nodes sẽ được hình thành hay không trong tương lai.

Cụ thể thì quá trình dự đoán liên kết là đi tính trọng số $score(u, v)$ cho mỗi cặp $< u, v >$ sau đó sắp xếp giảm dần để chọn ra v có độ đo cao nhất liên kết với u. (u, v là nodes trong đồ thị).

Hướng giải quyết tổng quát

- Bước 1: Tính toán khoảng cách đỉnh dựa trên các phương pháp học.
- Bước 2: Chọn một số lượng các cặp đỉnh có khoảng cách gần nhất
- Bước 3: Dự đoán cạnh mới từ các cặp đã chọn
- Bước 4: Đánh giá đồ thị được dự đoán so với đồ thị gốc.

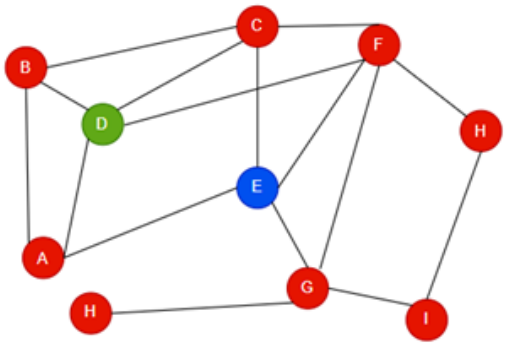
Máy học trong Link Prediction

- Supervised: Cung cấp tập các đỉnh để train mô hình.
- Unsupervised: Không cần tập huấn luyện mà:
  - Dựa trên độ tương tự (similarity-based): Các đỉnh có độ tương tự được kết nối với nhau.
  - Dựa trên gom nhóm (cluster-based): Các đỉnh từ cùng một nhóm chứng tỏ các mẫu kết nối tương tự.

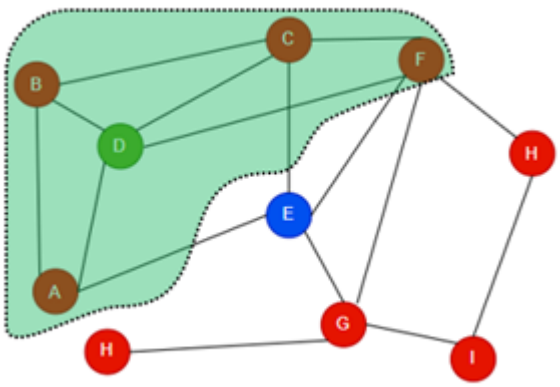
2.2.4 Phương pháp dựa trên đỉnh láng giềng

Common Neighbors: Là độ đo được sử dụng để tìm láng giềng chung của 2 đỉnh, đô đo tương đồng giữa hai đỉnh chính là số phần tử trong tập này $score\,(u, v) = |N(u) \cap N(v)|$.

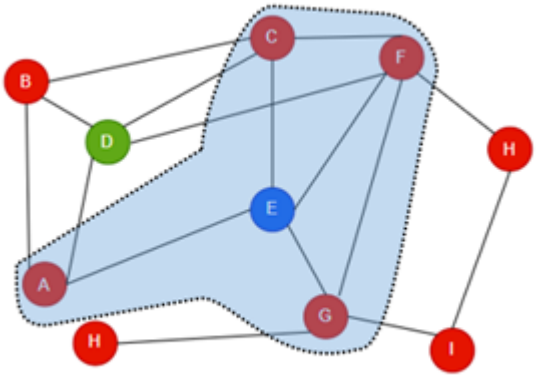
Ví dụ có đồ thị sau: (Tính độ đo giữa đỉnh D và đỉnh E)



Ta có $u = D$ và $v = E \Rightarrow N(u)$ là láng giềng của đỉnh D và $N(v)$ là láng giềng của đỉnh E



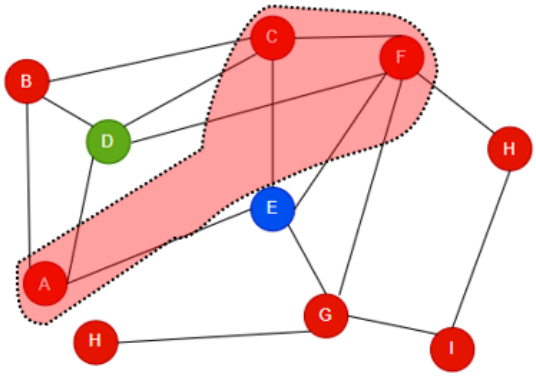
N(u)



N(v)

Áp dụng độ đo cho hai cặp đỉnh D và E có:

$$score\,(u,v) = |N(u) \cap N(v)| = |\{A,B,C,F\} \cap \{A,C,F,G\}| = 3$$



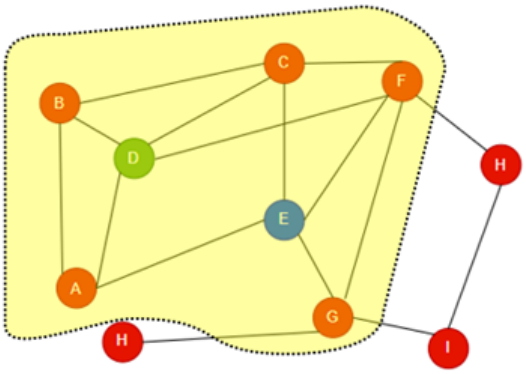
$N(u) \cap N(v)$

Jaccard Coefficient

- Hệ số Jaccard chuẩn hoá số láng giềng chung bằng tổng số láng giềng.

$$score\,(u,v) = \left|\frac{N(u) \cap N(v)}{N(u) \cup N(v)}\right|$$

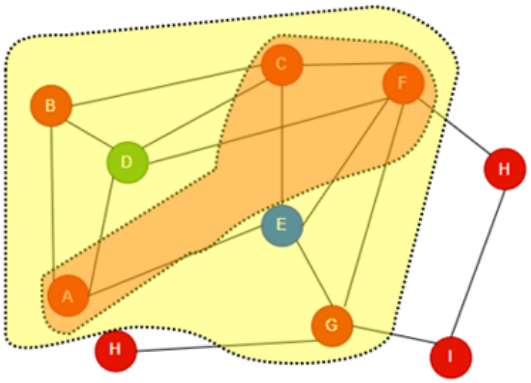
- Cùng với ví dụ ở trên ta dự đoán liên kết của đỉnh D và E theo hệ số Jaccard.
- Tổng số láng giềng của 2 đỉnh u và v đang xét được thể hiện qua hình



$N(u) \cup N(v)$

- Áp dụng hệ số Jaccard cho dự đoán liên kết D và E:

$$score\,(u,v) = \left|\frac{N(u) \cap N(v)}{N(u) \cup N(v)}\right| = \left|\frac{\{A,B,C,F\} \cap \{A,C,F,G\}}{\{A,B,C,F\} \cup \{A,C,F,G\}}\right| = \frac{3}{5}$$



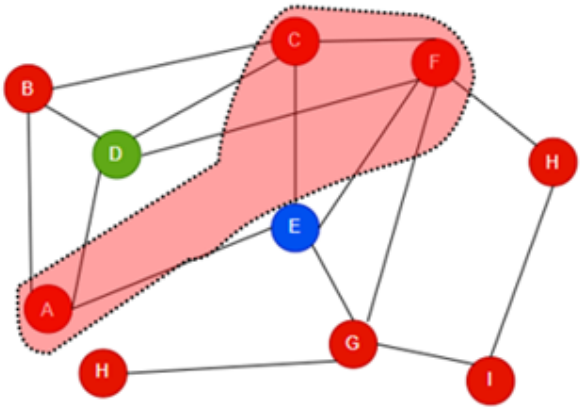
Jaccard coefficient

Adamic-Adar

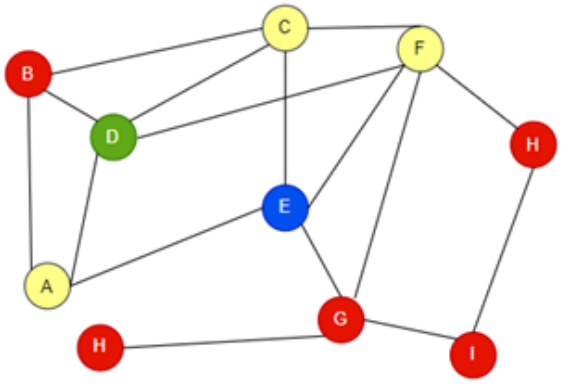
- Độ đo Adamic-Adar không chỉ đếm số láng giềng chung mà còn tính tổng các logarithm nghịch đảo bậc trung tâm của các láng giềng của 2 nút.

$$score\,(u,v) = \sum_{z \in\ N(u) \cap N(v)} \frac{1}{\log\,(N\,(z))}$$

- Dùng ví dụ trên để dự đoán liên kết D và E theo độ đo Adamic-Adar.
- Ở câu trên ta tìm được láng giềng chung của 2 đỉnh theo đồ thị sau:



- Từ các láng giềng đó ta tính bậc của từng nodes láng giềng
  - $N(C) = \{B, D, E, F\} = 3$
  - $N(F) = \{C, D, E, G, H\} = 5$
  - $N(A) = \{B, D, E\} = 3$

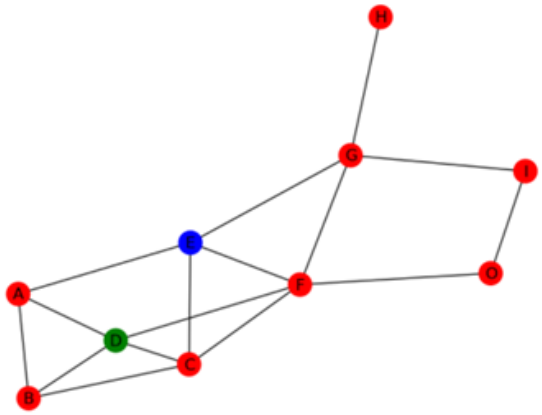


- Áp dụng độ đo Adamic-Ajar dự đoán liên kết của D và E

$$\begin{aligned} score\,(u,v) &= \sum_{z \in\ N(D) \cap N(E)} \frac{1}{\log\left(N\left(z\right)\right)} \\ &= \frac{1}{\log\left(N\left(C\right)\right)} + \frac{1}{\log\left(N\left(F\right)\right)} + \frac{1}{\log\left(N\left(A\right)\right)} \\ &= \frac{1}{\log\left(4\right)} + \frac{1}{\log\left(5\right)} + \frac{1}{\log\left(3\right)} = 2.253 \end{aligned}$$

Cài đặt thử nghiệm

- Cài đặt phép dự đoán liên kết bằng 3 độ đo trên cho biểu đồ ví dụ tương tự và so sánh.
- Biểu diễn biểu đồ của ví dụ trên:



- Sau khi cài phép dự đoán liên kết D và E theo 3 độ đo trên và compile ta có kết quả như sau:

```
Common neighbors:  3
Jaccard cofficient:  0.6
Adamic adar:  2.252921681630931
```

[-] So với kết quả của 3 thuật toán ta làm ví dụ ở trên hoàn toàn trùng khớp.

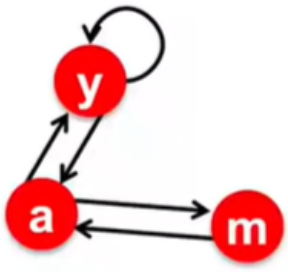
2.2.5 Phương pháp dựa trên tất cả đường đi

PageRank

- PageRank là một thuật toán đo lường tính quan trọng của mỗi nút trên đồ thị dựa vào số lượng liên kết đến và mức độ quan trọng của các nút nguồn tương ứng.
- PageRank làm việc trên các đồ thị có hướng trong trường hợp là một đồ thị vô hướng thì ta có thể chuyển đổi thành có hướng bằng cách cách cạnh trong đồ thị vô hướng sẽ trở thành cạnh hai chiều.
- Thuật ngữ Rank của 1 node nghĩa là tính quan trọng của 1 node dựa số lượng liên kết đến node đó ta tính tổng mức quan trọng các node nguồn. Ký hiệu r_{node} . (d_i là các liên kết ngoài của nút i)

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

- Ví dụ:



- Dựa vào đồ thị trên ta có:

$$r_a = \frac{r_y}{2} + \frac{r_m}{2} \; ; \; r_y = \frac{r_y}{2} + \frac{r_a}{2} \; ; r_m = \frac{r_a}{2}$$

- Ma trận kề ngẫu nhiên M dùng để thể hiển mối liên kết giữa các đỉnh.
  - Giả sử ta có nút j và d_j liên kết ngoài nếu nút j hướng đến nút i thì $M_{ij} = \frac{1}{d_j}$
  - Cứ thể chuẩn hoá được ma trận M sao cho tổng các phần tử trong cùng 1 cột $= 1$.
  - Ta có phương trình như sau $r = M * r$.
  - Ví dụ ta có thể dùng biểu đồ trên có:

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0 & 1 \\ 0 & 0.5 & 0 \end{bmatrix} * \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

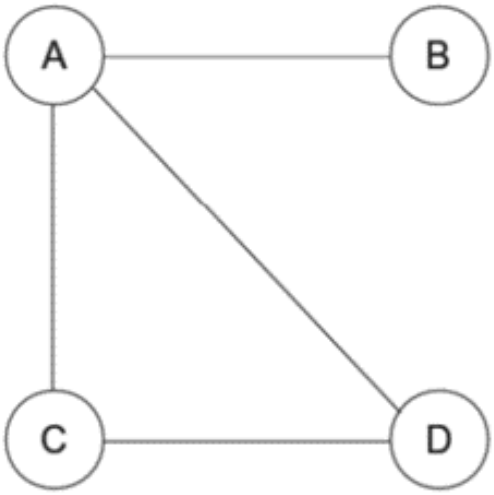
- Kết hợp cùng với Random Walk để có thể tránh những vấn đề về "dead ends" (các đỉnh không có liên kết ra) và "spider traps" (tập hợp các nút không là dead ends nhưng không có liên kết ra) bằng cách sử dụng xác suất khởi đầu β để tránh dead ends và thêm vào tính toán xác xuất nhỏ $1 - \beta$ cho việc "random suffer" nhảy đến một đỉnh ngẫu nhiên trong đồ thị để tránh "spider traps". Vậy nên ta có công thức sau:

$$r_{k+1} = \beta M r_k + (1 - \beta) \, r_0$$

- Các bước để dự đoán bằng PageRank như sau:
  - Bước 1: Tạo bản sau của đồ thị hiện tại và thêm cạnh (i,j) mà ta quan tâm đến.
  - Bước 2: Khởi tạo vector trạng thái ban đầu r_0 ($r_0[i] = 1$ nếu I là node ban đầu còn không thì bằng 0).
  - Bước 3: Mỗi lần lặp thì update PageRank cho các node trong graph
    - Nếu vẫn tiếp tục mà không cần phải quay lại node ban đầu thì công thức là: $r_{k+1} = M r_k$

- Khi bị buộc phải quay lại thì công thức sẽ thành: $r_{k+1} = \beta M r_k + (1-\beta)\, r_0$
- Bước 4: Vòng lặp dừng khi thoả điều kiện hội tự là khi $r_{k+1} = r_k$ thì lúc đó công thức sẽ là: $r = \beta M r + \ (1-\beta)\, r_0$

- Ví dụ có biểu đồ sau:



- Đầu tiên khởi tạo vector ban đầu là node B vì thế:

$$r_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \ M = \begin{bmatrix} 0 & 1 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \end{bmatrix}$$

- Update pageRank lặp liên tục:

$$r_1 = M * r_0 = \begin{bmatrix} 0 & 1 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \end{bmatrix} * \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$r_2 = M * r_1 = \begin{bmatrix} 0 & 1 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$r_3 = M * r_2 = \begin{bmatrix} 0 & 1 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \end{bmatrix} * \begin{bmatrix} 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 0 \\ 1/6 \\ 1/6 \end{bmatrix}$$

...

$$r_{42} = M * r_{41} = \begin{bmatrix} 0 & 1 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \end{bmatrix} * \begin{bmatrix} 0.37500091 \\ 0.12499958 \\ 0.24999975 \\ 0.24999975 \end{bmatrix} = \begin{bmatrix} 0.37499875 \\ 0.12500057 \\ 0.25000034 \\ 0.25000034 \end{bmatrix}$$

- Tại đây ta thấy được $r_{42} \approx r_{41}$. Lúc này đạt điều kiện hội tụ vòng lặp sẽ dừng và ta có được các chỉ số quan trọng cho từng nút và ta thấy rằng ta có thể tạo liên kết từ B đến C hay D vì cả hai đều có tính quan trọng như nhau.

2.3 GRAPH INDEXING

2.3.1 Giới thiệu

Khi internet phát triển thì dữ liệu và mạng lưới (đồ thị) ngày một gia tăng. Duyệt tuần tự rất tốn kém vì người ta không chỉ phải truy cập vào toàn bộ cơ sở dữ liệu đồ thị mà còn phải kiểm tra đồ thị đẳng cấu. Chúng ta biết rằng đẳng cấu đồ thị con là một bài toán NP-complete. Do đó, việc lập chỉ mục biểu đồ một cách hiệu quả (Graph Indexing) là cần thiết để tỉa các đồ thị không thỏa mãn yêu cầu truy vấn. Do đó việc phân tích các tập dữ liệu đồ thị lớn đã đặt ra nhiều thách thức và tập trung vào giải quyết vấn đề: làm thế nào để duyệt và tìm kiếm đồ thị một cách hiệu quả.

Vấn đề tìm kiếm đồ thị đã được xử lý trong nhiều lĩnh vực. Tùy theo cơ sở dữ liệu của đồ thị mà có các phương pháp tìm kiếm đồ thị khác nhau:

- Truy xuất ảnh dựa trên nội dung: lập chỉ mục đồ thị (index Graph) trên không gian nhiều chiều bằng sử dụng R-tree.
- Đối với cơ sở dữ liệu bán cấu trúc/XML: 1-index, A(k)-index,...

2.3.2 Chỉ mục đồ thị dựa trên đặc trưng (feature-based graph indexing

Định nghĩa bài toán: Cho trước 1 cơ sở dữ liệu đồ thị $D = \{G_1, G_2, ...G_n\}$và một truy vấn đồ thị Q, tìm kiếm tất cả các đồ thị chứa Q.

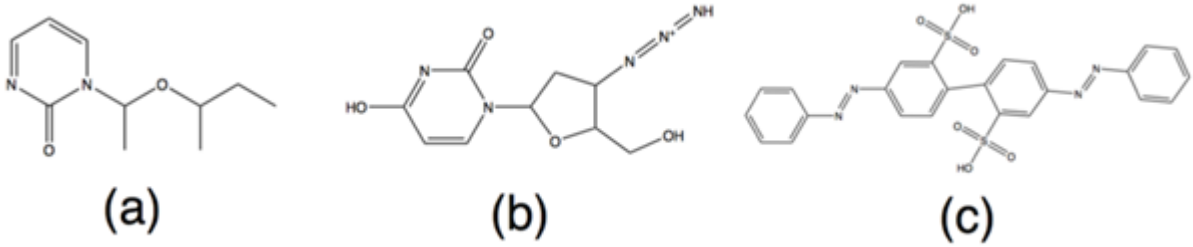
Tìm kiếm đồ thị con là một loại truy vấn đồ thị cơ bản, được quan sát thấy trong nhiều ứng dụng liên quan đến đồ thị. Feature-based graph indexing được thiết kế để trả lời các truy vấn tìm kiếm cấu trúc con bao gồm 2 bước:

- Index Constructor: Tính toán trước các đặc trưng trong cơ sở dữ liệu và xây dựng index trên các đặc trưng này. Một số phương pháp được sử dụng kể đến như gán nhãn canh/đỉnh (node/edge label), đường đi (path), cây (tree),...
    - Cho F là tập các thuộc tính (mà mỗi node là 1 thuộc tính) của cơ sở dữ liệu đồ thị D.
    - Với mọi $f \in F$, D_f là tập đồ thị chứa F, $D_f = \{G | f \subseteq G, G \in D\}$.
    - Chỉ mục đảo ngược được xây dựng giữa F và D:D_f có thể là mã Id của đồ thị chứa f.

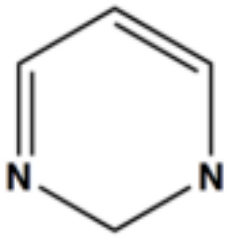
- Query Processing: Bao gồm 3 bước nhỏ:
  - Search: liệt kê tất cả thuộc tính của đồ thị Q, để tìm tập truy vấn ứng viên (candidate query answer set - C_Q).
  - Fetching: truy xuất đồ thị trong tập truy vấn ứng viên từ nơi lưu trữ.
  - Verification: kiểm tra đồ thị trong tập ứng viên có thỏa mãn truy vấn ban đầu hay không.

Path-based approach (dựa trên đường đi) Ý tưởng của phương pháp lập chỉ mục dưa trên đường đi, được thực hiện theo ý tưởng từ sách [1], coi các đường đi trên CSDL đồ thị như một đặc trưng để lập chỉ mục. Các bước thực hiện giống như trên, ta liệt kê tất cả đường đi có trong một CSDL (độ dài đường đi tối đa là $maxL$). Với đường đi là chuỗi các đỉnh $v_1, v_2, ..., v_k$,$\forall 1 \leq i \leq k-1, (v_i, v_{i+j})$ là một cạnh. Từ đó, ta sử dụng các chỉ mục này để xác định đồ thị mà chứa đường đi có trong đồ thị truy vấn Q (chọn đường đi dài nhất).

Ví dụ, Giả sử ta có tập đồ thị D sau:



yêu cầu truy vấn đồ thị Q:



Thực hiện truy vấn tuần tự theo các bước:

- Index Construction (Xây dựng chỉ mục):
  - Bước 1: Liệt kê các đường đi với độ dài cụ thể trong CSDL đồ thị (mỗi node là một nguyên tử):

- 0-length: C, O, N, S.
- 1-length: C-C, C-O, C-N, O-S, N-N.
- 2-length: C-C-C, C-C-C, C-N-C, N-C-O, C-C-N,...
- 3-length: C-C-C-N, C-C-C-O, O-C-C-O,...
- ...

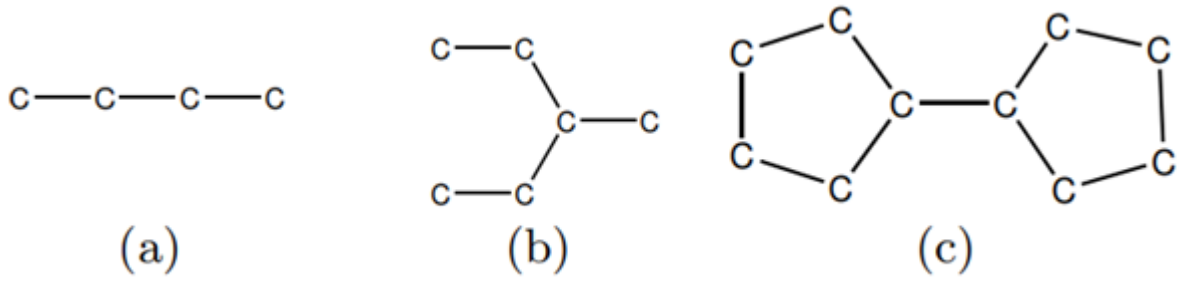
- Bước 2: Xây dựng chỉ mục đảo ngược (inverted index):
  - $D_C = \{a, b, c\}, D_O = \{a, b, c\}, D_N = \{a, b, c\}, D_S = \{b, c\}.$
  - $D_{C-C} = \{a, b, c\}, D_{C-O} = \{a, b, c\}, ..., D_{N-N} = \{b, c\}.$
  - $D_{C-C-C} = \{a, b, c\}, D_{C-N-C} = \{a, b\}$,...

- Query Processing:
  - Bước 3 (Search): Đối với truy vấn đồ thị Q, ta phân tách thành các đường đi để tính toán các đặc trưng:
    - $D_C = \{a, b, c\}, D_N = \{a, b, c\}.$
    - $D_{C-C} = \{a, b, c\}, D_{C-N} = \{a, b, c\}.$
    - $D_{C-N-C} = \{a, b\}, D_{C-C-C} = \{a, b, c\}.$
  - Bước 4 (Search): Từ việc liệt kê đặc trưng của đồ thị Q, ta tìm sự giao nhau giữa đồ thị Q và D theo công thức sau: $C_Q = \cap_f D_f = \{a, b\}$
  - Bước 5 (Fetching Verification): Truy xuất đồ thị từ CSDL và kiểm tra thỏa mãn yêu cầu truy vấn Q có a,b.

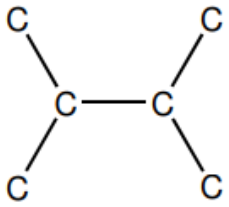
Một số nhược điểm:

- Không thực sự tối ưu cho các đồ thị phức tạp và lớn, nhưng phù hợp cho đồ thị có dạng cây.
- Việc phá vỡ cấu trúc của truy vấn có thể dẫn đến mất đi một số thông tin của đồ thị, bởi đó có thể trả ra kết quả đúng giả (false positive).

Ta có thể xét ví dụ sau cho thấy hướng tiếp cận dựa trên đường đi là không phù hợp: Xét đồ thị D sau



Có yêu cầu truy vấn đồ thị:



Ta thực hiện theo các bước trên:

- Bước 1: Index Construction, ta lập được chỉ mục đảo ngược (inverted index) cho D như sau:
  - $D_C = \{a, b, c\}$.
  - $D_{C-C} = \{a, b, c\}$.
  - $D_{C-C-C} = \{a, b, c\}$.
  - $D_{C-C-C-C} = \{a, b, c\}$.
- Bước 2: Tính toán đặc trưng của Q:
  - $D_C = \{a, b, c\}$.
  - $D_{C-C} = \{a, b, c\}$.
  - $D_{C-C-C} = \{a, b, c\}$.
  - $D_{C-C-C-C} = \{a, b, c\}$.
- Bước 3: Tìm tập giao nhau với độ dài maxL=4, ta có $C_Q = \cap f D_f = \{a, b, c\}$.
- Ta nhận thấy rằng cả a,b,c đều thỏa điều kiện truy vấn Q, nhưng trên thực tế đáp án chỉ có (c), nên (a) và (b) là kết quả khớp giả.

⇒ Đề xuất hướng tiếp cận mới để cải thiện: cấu trúc phổ biến và cấu trúc phân biệt.

Graph-based Graph Indexing (gIndex) Được thực hiện và công bố trong bài báo [7]. Các bước thực hiện giống như trên, nhưng thay vì ta liệt kê đường đi như đặc trưng thì ta sử liệt kê các cấu trúc đồ thị con. Vì số lượng cấu trúc các đồ thị con lớn hơn nhiều so với dựa trên đường đi, nên chỉ mục của các cấu trúc có thể tăng theo cấp lũy thừa. Do đó ta tìm các mẫu phổ biến thỏa mãn lớn hơn một ngưỡng nhất định.

Về các bước thực thiện hướng tiếp cận này như sau:

Cho CSDL đồ thị $D = \{G_1, G_2, ..., G_n\}$ và một cấu trúc f, ta có độ hỗ trợ $sup(f) = |D_f|$, cấu trúc f phổ biến khi độ hỗ trợ của nó lớn hơn ngưỡng $\min_s upchotrc$:sup(f) $\geq min_s up$.

- Index Construction: Có thể chuyển đồ thị thành nhãn chính tắc để cải thiện kiểm tra đẳng cấu, vì kiểm tra đẳng cấu là một bài toán NP-Khó.
- Query Processing:
  - Tìm các cấu trúc phổ biến trong cơ sở dữ liệu (Searching): Có thể sử dụng 1 số thuật toán tìm cấu trúc phổ biến như Apriori, gSpan. Nếu Q phổ biến, ta có thể truy xuất trực tiếp khi lập chỉ mục xong. Ngược lại, thực hiện sắp xếp cấu trúc con của Q theo thứ tự tăng dần độ hỗ trợ $f_1, f_2, \ldots, f_n$. Lập chỉ mục cho các cấu trúc phổ biến, khi đó tập truy vấn $C_Q = \cap_{1 \le j \le i} D_{f_j}$có kích thước tối đa là $sup\,(f_i)$. Ngoài ra, thay vì sử dụng min_sup với giá trị cố định, ta có thể sử dụng size-increasing support threshold, sử dụng một hàm $\psi\,(size\,(f))$ là một hàm số không giảm đơn điệu. Bằng cách này, với những cấu trúc con kích thước lớn sẽ cần ngưỡng độ trợ lớn hơn, và cấu trúc con có kích thước nhỏ sẽ cần ngưỡng độ trợ nhỏ hơn, khi đó f phổ biến khi và chỉ khi $sup\,(f) \ge \psi\,(size\,(f))$.
  - Tỉa những tập cấu trúc dư thừa để giữ tập chỉ gồm cấu trúc phân biệt(discriminative structure) và giảm kích thước của tập ứng viên.
    - Cho x là một substructure, $x \in F$, x là một cấu trúc dư thừa hoặc phân biệt. Để đo độ phân biệt của x, ta tính toán như sau:

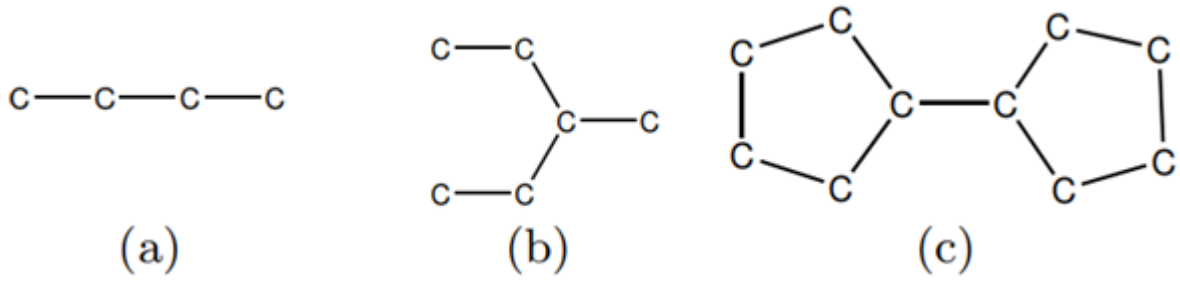
$$Pr\,(x|f_{\varphi_1}, \ldots, f_{\varphi_m})\,, f_{\varphi_i} \subset x, 1 \le \varphi_i \le n$$

    - Tỷ lệ phân biệt:

$$\gamma = \frac{1}{Pr\,(x|f_{\varphi_1}, \ldots, f_{\varphi_m})} = \frac{\left|\cap_i D_{f_{\varphi_i}}\right|}{|D_x|}$$

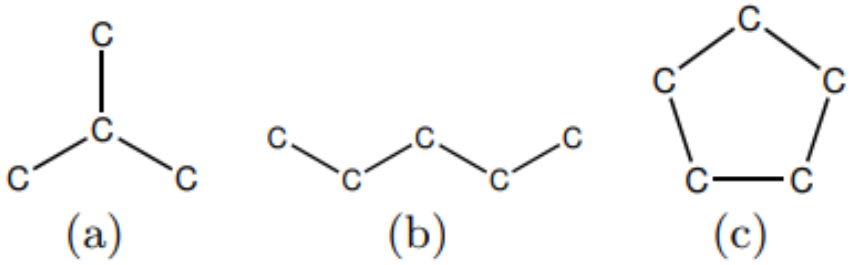
      - $\gamma = 1$: x là cấu trúc dư thừa.
      - $\gamma \gg 1$:x càng phân biệt. khi đó, x là ứng viên phù hợp cho chỉ mục.
    - Một tập x được gọi là phân biệt khi và chỉ khi $\gamma_x \ge \gamma_{min}$

Ví dụ: Xét lại ví dụ trên, ta đặt $\gamma_{min} = 1.5$, hàm độ trợ $\psi = 1$.



Ta tìm được các cấu trúc con phân biệt:

- Đối với các cấu trúc như C, C-C, C-C-C đều tồn tại trong (a), (b) và (c). Giống như đều tồn tại trong database, nên ta không cần tìm cấu trúc phân biệt (vì nó dư thừa).
- Khi $maxL = 4$, ta có một số cấu trúc sau:



- Ta tính được độ phân biệt của các cấu trúc này:

$$\gamma_a = \frac{\left|\cap_i D_{f_{\varphi_i}}\right|}{|D_x|} = \frac{3}{2} = 1.5, \gamma_b = \frac{\left|\cap_i D_{f_{\varphi_i}}\right|}{|D_x|} = \frac{3}{2} = 1.5, \gamma_c = \frac{\left|\cap_i D_{f_{\varphi_i}}\right|}{|D_x|} = \frac{2}{1} = 2.$$

- Xây dựng chỉ mục nghịch đảo (inverted index).
- Tính quan hệ giao giữa truy vấn Q và CSDL đồ thị: $C_Q = C_Q \cap D_x$
  - Ta cần làm giảm số lần thực hiện bước trên: Giả sử $F(q)$ là tập cấu trúc đồ thị con phân biệt (hay feature index). $F_Q = \{f_x | f_x \subset Q \wedge f_x \in F\}$. Ta có $F_m(Q)$ là tập cấu trúc con chứa đồ thị con khác. $F_m(Q) = \{f_x | f_x \in F(Q),\ \nexists f_y, s.t..,\ f_x \subset f_y \wedge f_y \in F(Q)\}$ Khi đó F_m là đồ thị phân biệt tối đại. Để tính C_Q, chúng ta chỉ cần tính tập giao trên danh sách ids của đồ thị phân biệt tối đại này.

2.4 COMMUNITY DETECTION

2.4.1 Định nghĩa bài toán

Phát hiện cộng đồng là quá trình phân tích đồ thị/mạng để xác định các nhóm đỉnh (group) được kết nối với nhau dày đặt hơn so với phần còn lại của đồ thị/mạng. Các nhóm như thế sẽ được gọi là cộng đồng hoặc cụm, đại diện cấu trúc con có ý nghĩa trong mạng, nơi các đỉnh thể hiện các đặc điểm tương tự hoặc có vai trò tương tự.

2.4.2 Ứng dụng

- Phân vùng mạng thành các nhóm để hiểu được cấu trúc

- Giúp xác định các nhóm gen có liên quan chức năng hoặc tham gia vào cùng một quá trình sinh học
- Phát hiện cộng đồng trong mạng xã hội giúp xác định nhóm người dùng có các quan hệ mạnh với nhau
- Giúp phát hiện các gian lận hoặc hành vi không bình thường trong các cộng đồng

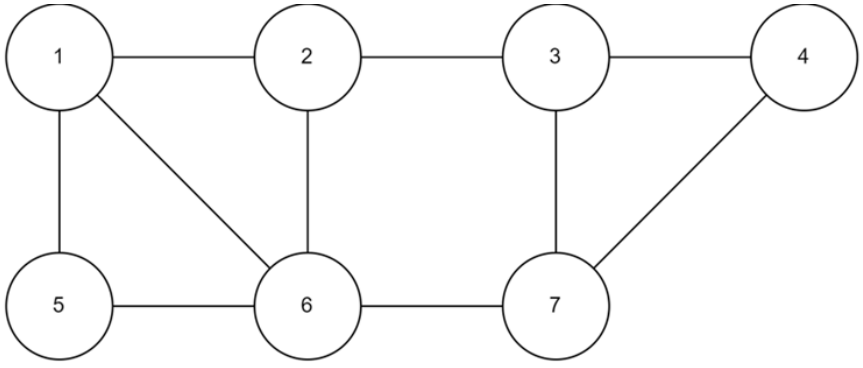
Một số thuật toán trong phát hiện cộng đồng:

- Girvan-Newman algorith
- Fluid Communities algorithm
- Label Propagation algorithm
- Clique Percolation algorithm
- Keinighan-Lin algorithm

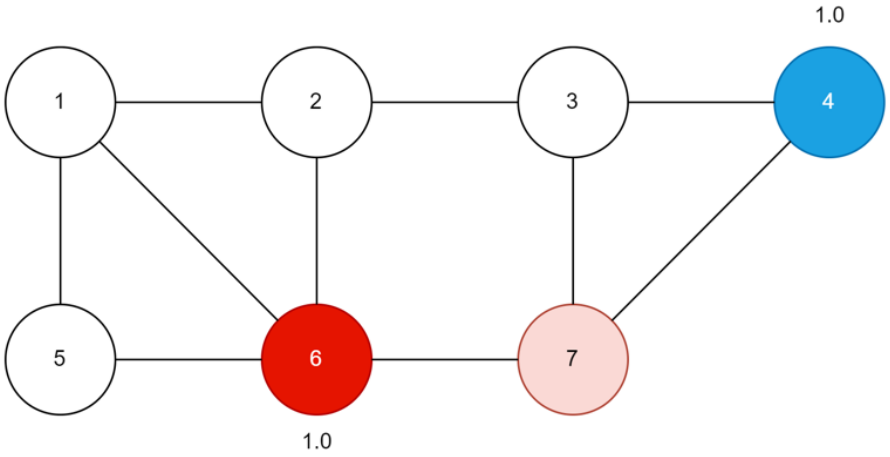
Trong chủ đề này, nhóm xin được phép trình bày ví dụ minh hoạ cho thuật toán Fluid Communities.

- Ý tưởng: Ý tưởng chính của thuật toán là xem xét các thông tin dòng chảy (fluid) trên mạng và sử dụng dòng chảy này để xác định các cộng đồng, các luồng sẽ có xu hướng đẩy và hút các đỉnh gần kề để đạt được trạng thái ổn định của toàn mạng. Do đó sau 1 số các bước mạng sẽ đạt được trạng thái ổn định.
- Pseudo code: Cho đồ thị G, có số lượng cộng đồng k, và số vòng lặp i:
  - Khởi tạo k cộng đồng (random).
  - Khởi tạo mật độ cho mỗi cộng đồng.
  - Với mọi đỉnh $v \in G$:
    - Gán v vào trong cộng đồng có mật độ cao nhất.
    - Cập nhật cộng đồng và mật độ.
  - Kiểm tra hội tụ hay không, nếu không quay lại bước 3.
  - Trả kết quả cộng đồng.

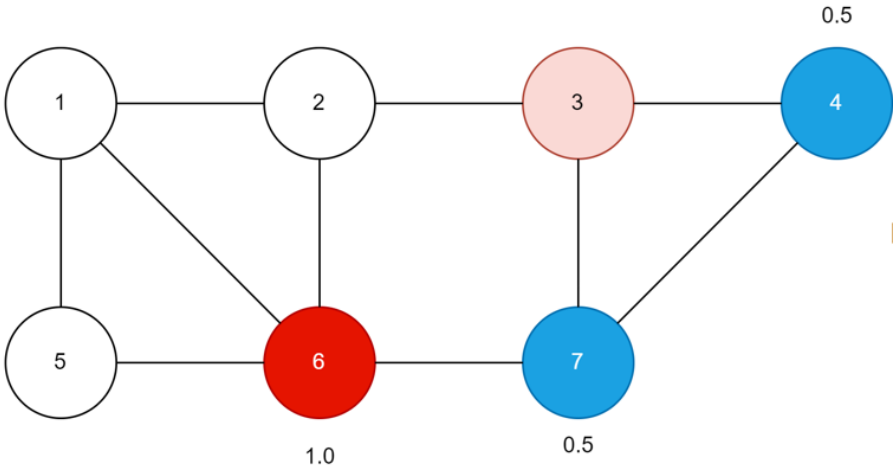
Ví dụ minh họa cho thuật toán: Xét đồ thị G có 7 đỉnh như hình:



Bước 1: : Khởi tạo $k = 2$ cộng đồng, giả sử cộng đồng đầu tiên sẽ chứa đỉnh 4 và cộng đồng thứ 2 chứa đỉnh 6.

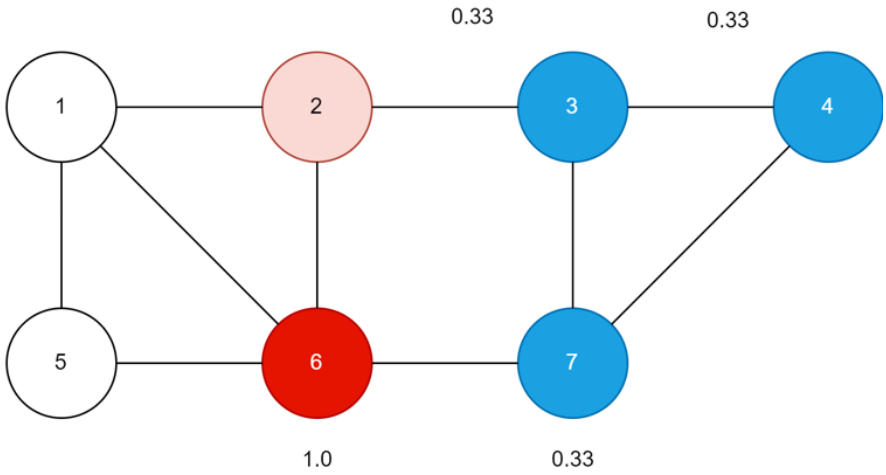


Bước 2: Ta xét đỉnh 7, lần lượt xét các đỉnh kề của 7 ta có 2 đỉnh kề của 7 là 4 và 6 lần lượt nằm trong cộng đồng 1 và cộng đồng 2 và cả 2 đều có density bằng nhau do đó ta sẽ gán đỉnh 7 ngẫu nhiên vào cộng đồng 1.



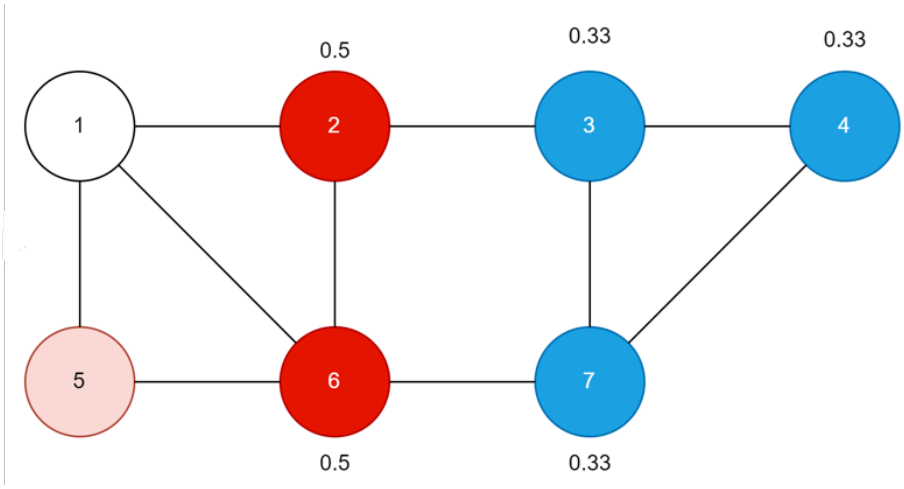
Sau đó tính toán lại giá trị density cho từng cộng đồng.

Bước 3: Tiếp theo xét 3, với các đỉnh kề của 3 lần lượt là 2, 4, và 7 nhưng 2 chưa nào trong cộng đồng nào do đó 3 sẽ được gán vào cộng đồng 1 do density của cộng đồng 1 chứa 4 và 7 là 0.5.

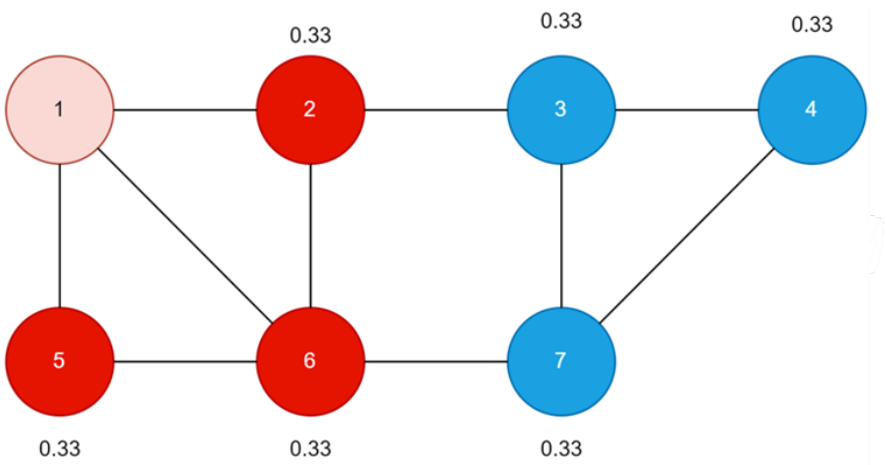


Sau đó tính lại giá trị density lại cho các cộng đồng. Cộng đồng 1 có 3 đỉnh do đó density sẽ là $1/3 = 0.33$ và cộng đồng 2 chỉ có 1 đỉnh 6 nên density là 1.0.

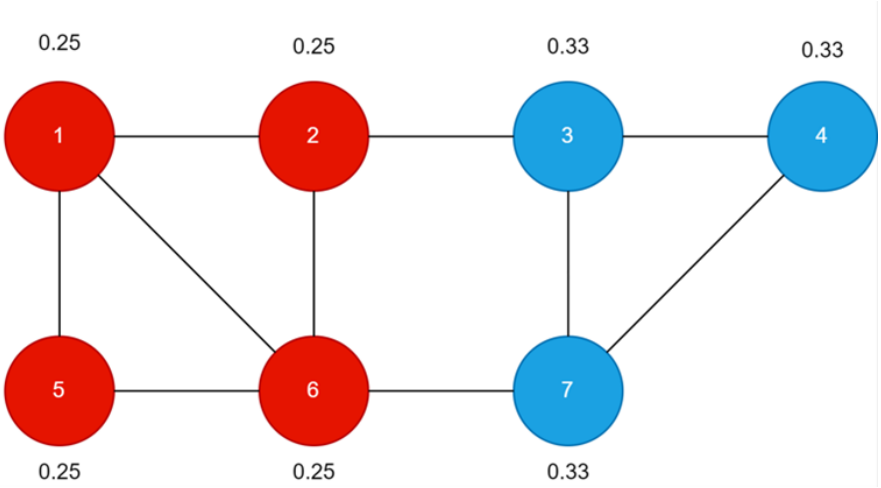
Bước 4: Tiếp theo ta xét đỉnh 2, lần lượt các đỉnh kề của 2 là 1, 3, 6. Do 1 không nằm trong cộng đồng nào nên ta bỏ qua, xét đỉnh 3 và 6 ta thấy đỉnh 3 nằm trong cộng đồng 1 với density là 0.33 nhưng đỉnh 6 nằm trong cộng đồng 2 với density là 1.0 do đó đỉnh 2 sẽ được gán vào cộng đồng 2. Tiếp theo ta tính lại density cho các cộng đồng, với cộng đồng 1 hiện đang có 3 đỉnh do đó $density = \frac{1}{3} = 0.3$, cộng đồng 2 có 2 đỉnh nên $density = \frac{1}{2} = 0.5$



Bước 5: Tiếp tục xét đỉnh 5, các đỉnh kề của 5 lần lượt là 1 và 6, nhưng chỉ có đỉnh 6 thuộc cộng đồng 2 nên do đó đỉnh 5 sẽ được gán vào cộng đồng với density là 0.5. Tiếp tục tính lại density cho cộng đồng, cộng động 1 không có gì thay đổi so với ban đầu nên density vẫn là 0.33, cộng đồng 2 đã thêm vào đỉnh 5 do đó density sẽ là $\frac{1}{3} = 0.33$.



Bước 6: Xét đỉnh 1, dễ thấy các đỉnh kề của 1 đều thuộc vào cộng đồng 2 do đó 1 cũng sẽ được gán vào cộng đồng 2 với density là 0.33, như bước 5 cộng đồng 1 vẫn không có gì thay đổi do đó density của cộng đồng 1 vẫn là 0.33, cộng động 2 được thêm vào đỉnh 1 do đó density của cộng đồng 1 là $\frac{1}{4} = 0.25$

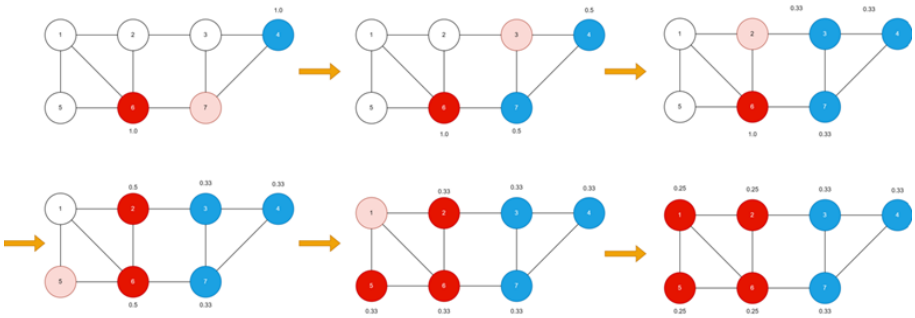


Bước 7, 8: Ta vẫn sẽ xét lại đỉnh 4 và 6 do có thể sau các bước cập nhật thì đỉnh 4 và 6 sẽ thuộc vào cộng đồng khác với density của cộng đồng khác lớn hơn density của cộng đồng cũ. Nhưng với ví dụ này các đỉnh 4 và 6 vẫn giữ nguyên cộng đồng và không có gì thay đổi vì với đỉnh 6 lần lượt các đỉnh kề là 1, 2, 5, 7 trong đó các đỉnh 1, 2, và 5 thuộc vào cộng đồng 2 do đó tổng các density sẽ là 1 trong khi đỉnh 7 thuộc vào cộng đồng 1 với tổng density là 0.33 nên đỉnh 6 vẫn sẽ giữ nguyên cộng đồng là 2, tương tự tính toán cho đỉnh 4.

Bước 9: Ta sẽ tiến hành thực hiện duyệt qua lại tất cả các đỉnh để tiến hành lan truyền như các bước đề cập ở trên, nếu các cộng đồng vẫn còn thay đổi thì sẽ tiếp tục thực hiện lại các bước 2 đến 8 với thứ tự các đỉnh được chọn ngẫu nhiên. Nếu không có

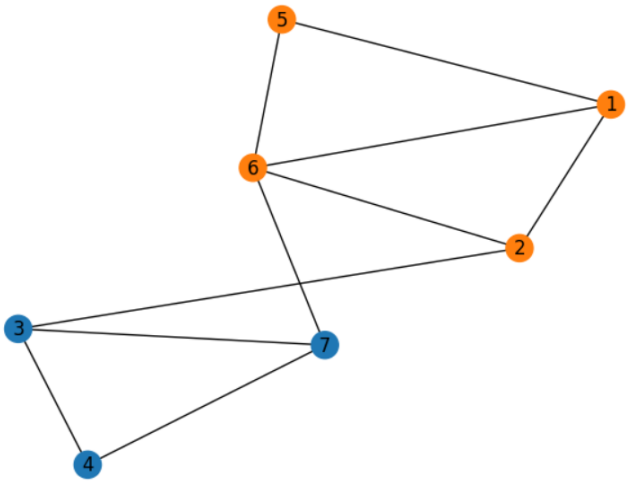
thay đổi giữa các cộng đồng tức mạng đã hội tụ và tiến hành trả về kết quả các cộng đồng.

Tổng quan quá trình thực hiện:



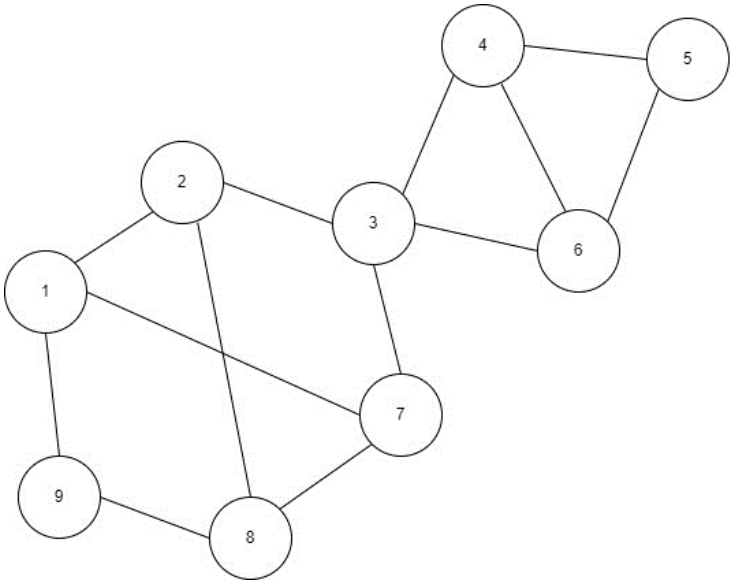
Kiểm tra lại thuật toán:

Tiếp theo nhóm sử dụng code để chạy thử ví dụ trên:

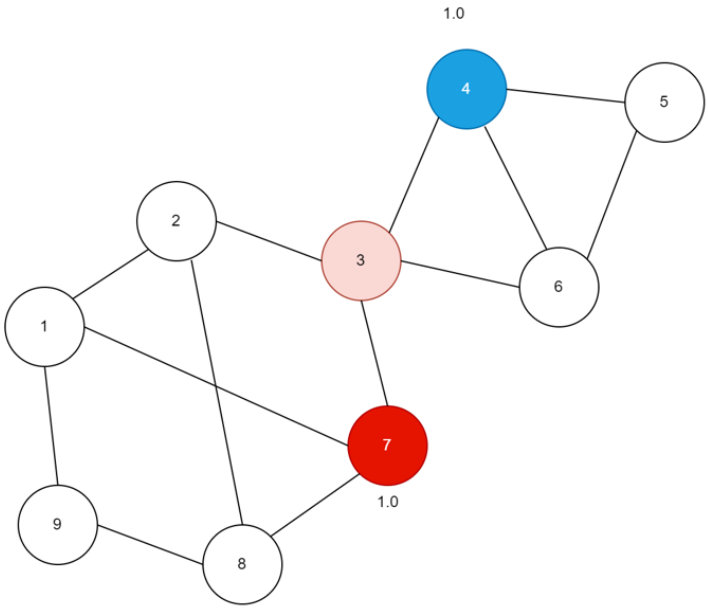


Ta thấy kết quả sau khi chạy bằng code cũng tương tự như tính toán của nhóm.

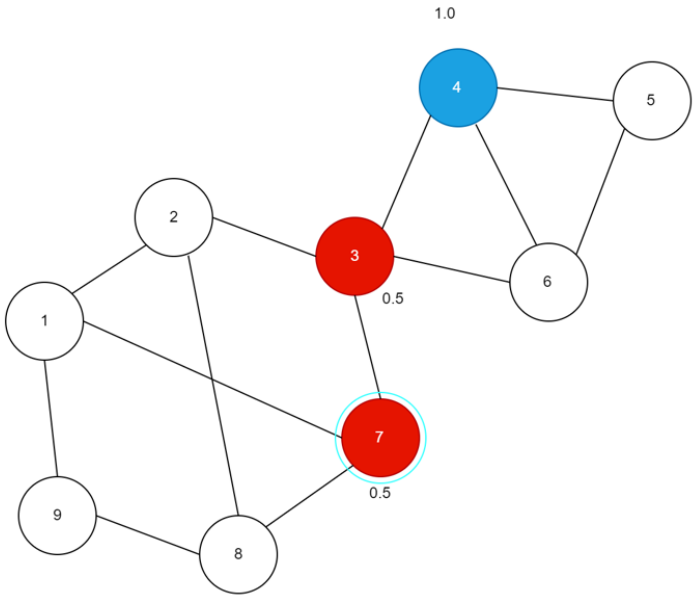
Ví dụ 2: Ta sẽ xét 1 ví dụ khó hơn với đồ thị G gồm 9 đỉnh như sau:



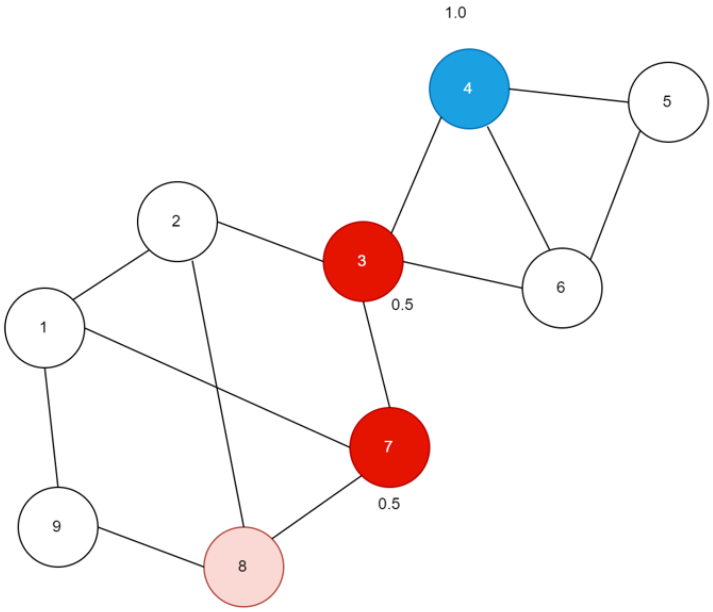
Bước 1: Như ví dụ trên ta vẫn chọn số lượng communities $k = 2$ với cộng đồng 1 chứa đỉnh 4 và cộng đồng 2 chứa đỉnh 7.



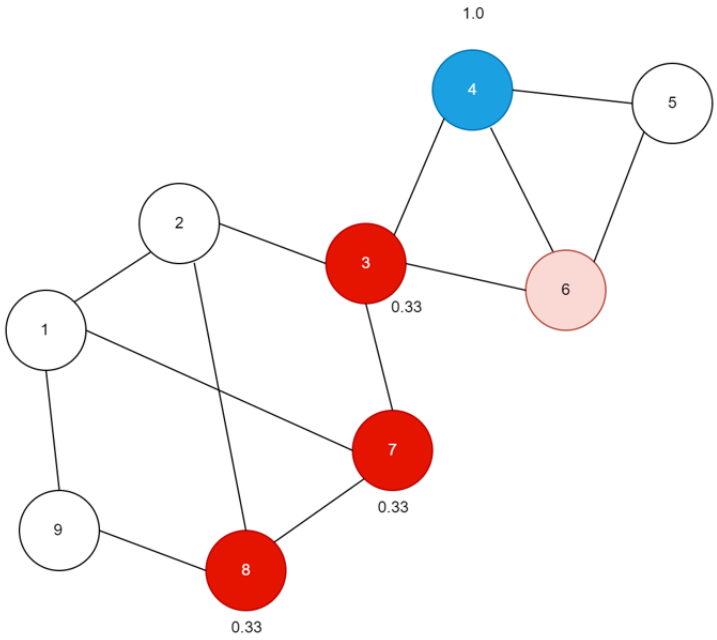
Bước 2: Ta xét đỉnh 3, ở đây ta có các đỉnh kề của 3 là 4 và 7 nằm ở 2 cộng đồng khác nhau và có cùng mật độ do đó ta sẽ gán 3 vào cộng đồng 2 ngẫu nhiên. Sau đó ta tính toán lại density cho cộng đồng 2, do có 2 đỉnh nên density là $\frac{1}{2} = 0.5$.



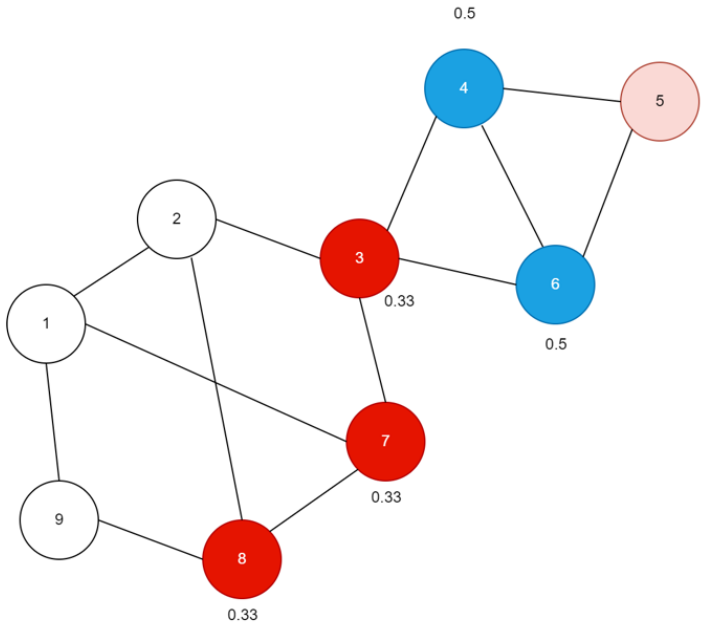
Bước 3: Tiếp theo ta xét đến đỉnh 7 (được tô vòng màu xanh), ta có đỉnh 7 hiện đang ở cộng đồng 2 và các đỉnh kề của 7 chỉ có 3 đã có cộng đồng do đó đỉnh 7 vẫn sẽ giữ nguyên ở cộng đồng 2.



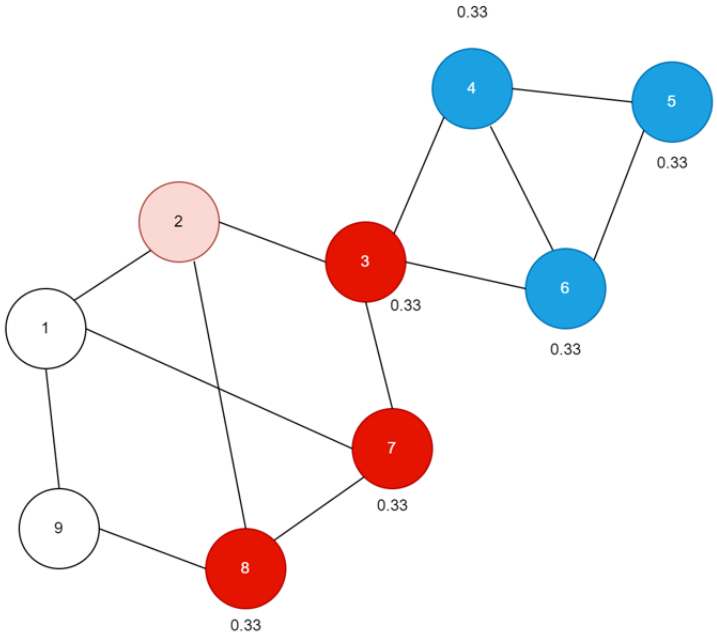
Bước 4: Tiếp theo ta xét đỉnh 8 với các đỉnh kề là 2, 7, và 9. Do các đỉnh kề của 8 chỉ có 7 đã thuộc vào cộng đồng 2 với density là 0.5 do đó 8 sẽ được gán vào cộng đồng 2, sau đó ta cập nhật lại density cho cộng đồng 2 bây giờ sẽ là $\frac{1}{3} = 0.33$.



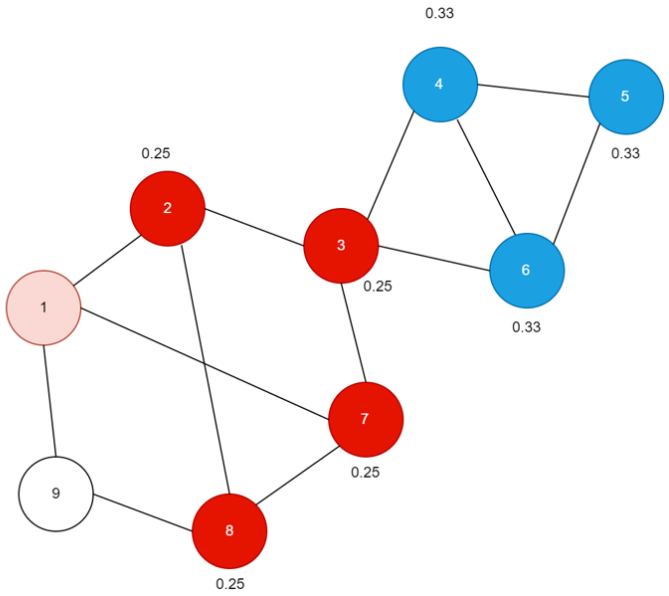
Bước 5: Tiếp theo xét đỉnh 6, với các đỉnh kề là 3, 4, và 5 nhưng chỉ có đỉnh 3 và 4 đã thuộc vào cộng đồng do đó ta chỉ xét các đỉnh 3 và 4. Do đỉnh 3 thuộc vào cộng đồng 2 với density là 0.33 nên đỉnh 6 sẽ thuộc vào cộng đồng 1 với density là 1.0. Sau đó ta cập nhật lại density cho cộng đồng 1, $density = \frac{1}{2} = 0.5$



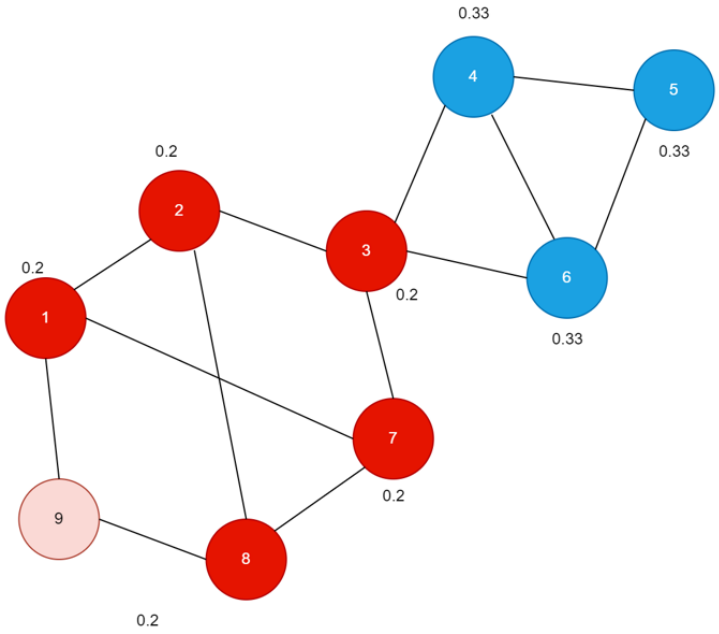
Bước 6: Tiếp theo ta xét đỉnh 5, đỉnh này hiển nhiên sẽ thuộc vào cộng đồng 1 do các đỉnh kề của nó đều thuộc vào cộng đồng 1 tổng density là 1.0. Ta cập nhật lại denstiy cho cộng đồng 1, density sẽ là $\frac{1}{3} = 0.33$ do có 3 đỉnh trong cộng đồng.



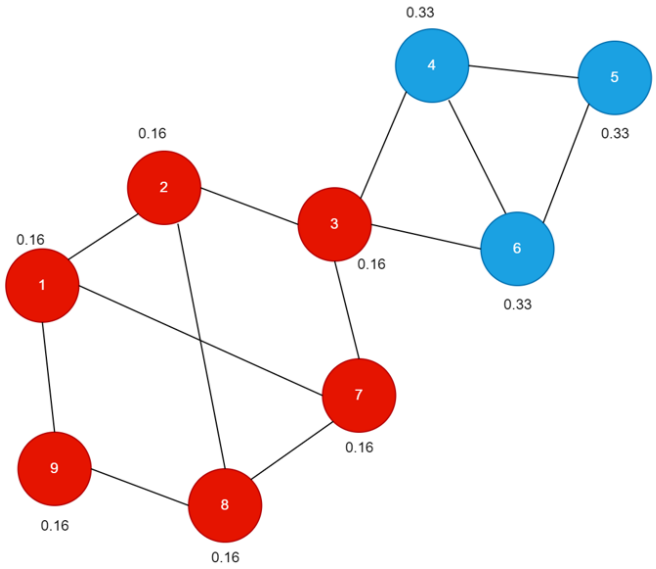
Bước 7: Bây giờ ta xét đỉnh 2, với các đỉnh kề là 1, 3, và 8 trong đó 3 và 8 thuộc vào cộng đồng 2 và không có đỉnh kề nào thuộc vào cộng đồng 1 do đó đỉnh 2 được thêm vào cộng đồng 2 (với tổng density là 0.66), tương tự cập nhật lại density cho cộng đồng 2, hiện đang có 4 đỉnh do đó density sẽ là $\frac{1}{4} = 0.25$.



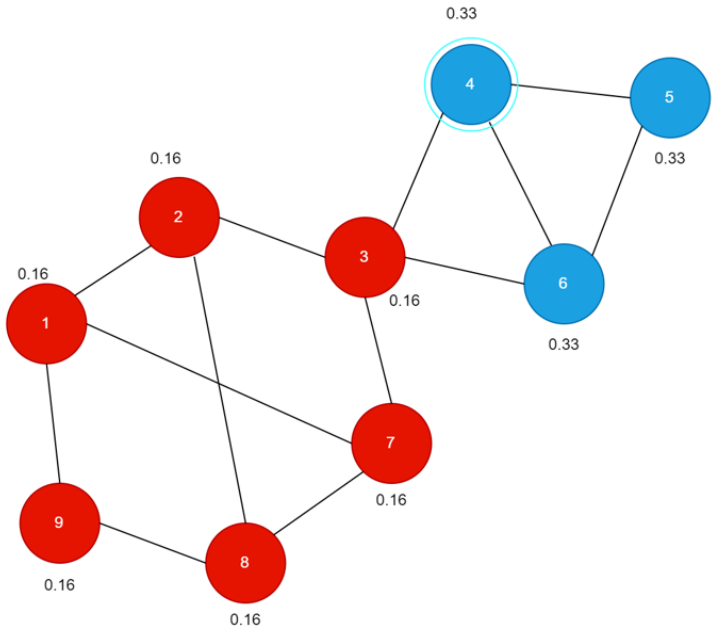
Bước 8: Ta xét đỉnh 1, với các đỉnh kề là 2, 7, và 9 trong đó 2 đỉnh 2 và 7 thuộc vào cộng đồng 2 và không có đỉnh nào thuộc vào cộng đồng 1 do đó đỉnh 1 được thêm vào cộng đồng 1 với tổng density là 0.5, tiếp tục cập nhật lại density của cộng đồng 2 sẽ là $\frac{1}{5} = 0.2$



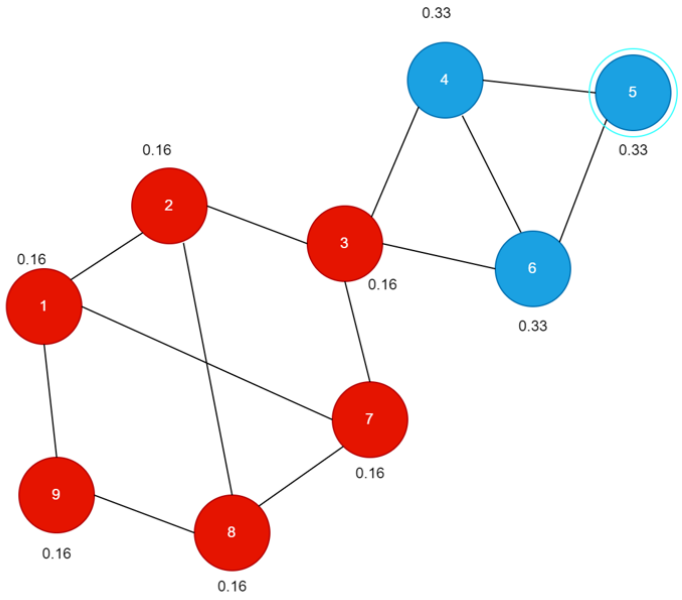
Bước 9:Tiếp theo ta xét đỉnh 9 (đỉnh cuối cùng chưa được thêm vào bất cứ cộng đồng nào), hiển nhiên đỉnh 9 sẽ phải thuộc vào cộng đồng 2 vì tất cả đỉnh kề của đỉnh 9 đều thuộc vào cộng đồng 2 với tổng density là 0.4, bây giờ density của cộng đồng 2 sẽ là $\frac{1}{6} = 0.16$ (cộng đồng 2 có 6 đỉnh).



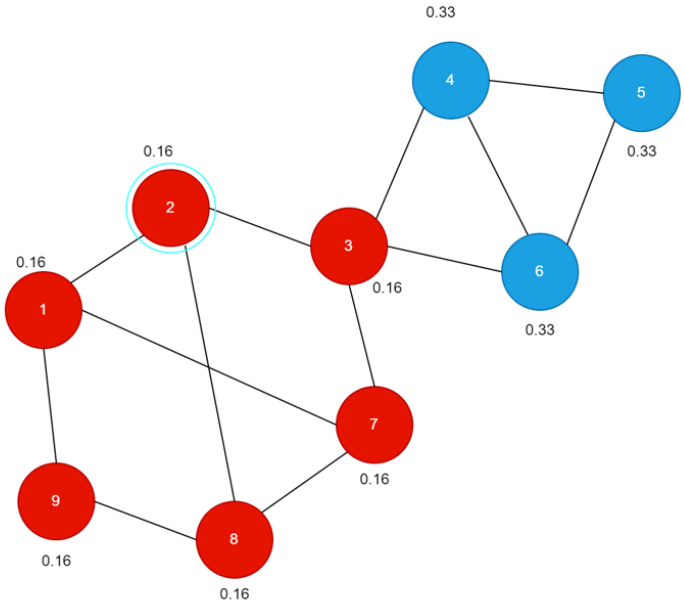
Bước 10: Cuối cùng ta xét đỉnh cuối cùng là đỉnh 4, kết quả không có gì thay đổi đỉnh 4 vẫn sẽ nằm trong cộng đồng 1 với tổng density của cộng đồng 1 là 0.66 lớn hơn tổng density của cộng đồng 2 là 0.16.



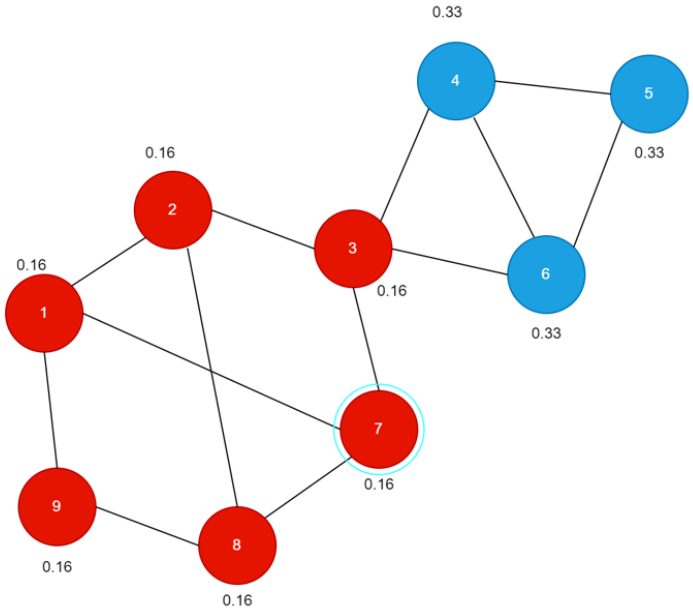
Bước 11: Sau khi đã duyệt qua tất cả các đỉnh, tiếp theo ta sẽ phải duyệt qua các đỉnh lần nữa để kiểm tra xem mạng đã ổn định hay chưa. Bắt đầu với đỉnh 5, ta thấy mọi thứ không có gì thay đổi khi tất cả các đỉnh kề của 5 đều thuộc vào cộng đồng 1.



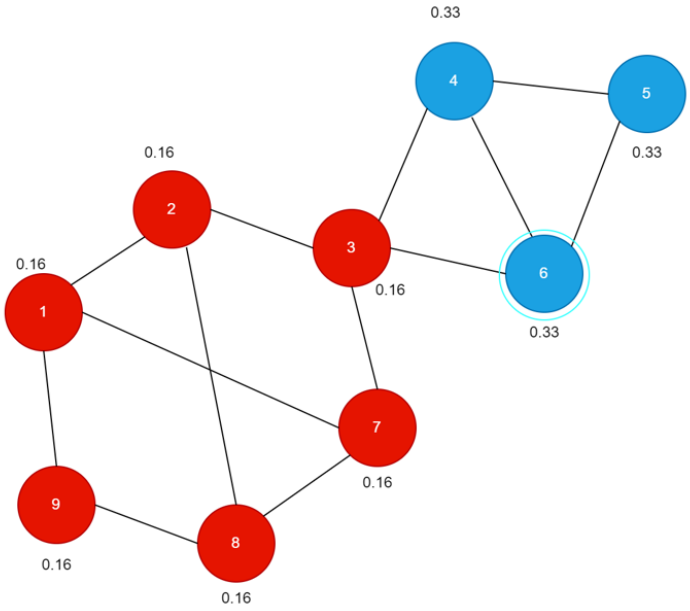
Bước 12: Ta xét tiếp đỉnh 2, tương tự đỉnh 5, tất cả đỉnh kề của đỉnh 2 đều thuộc cộng đồng 2 do đó không có gì thay đổi tại bước này.



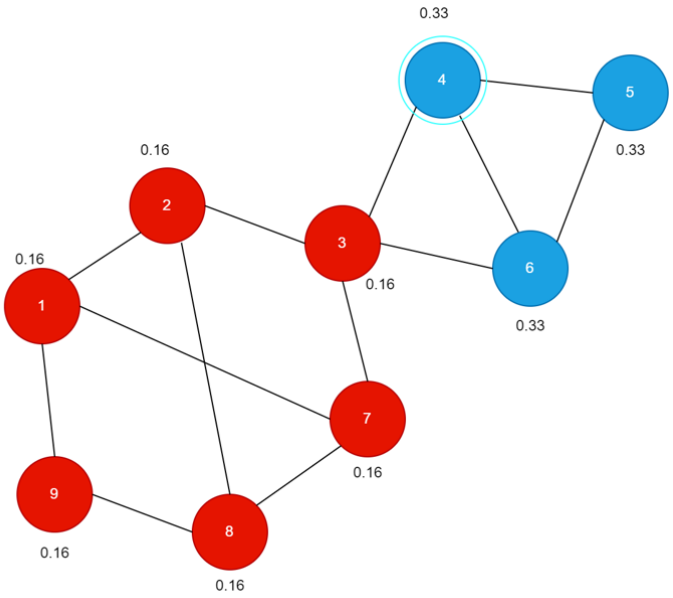
Bước 13: Tiếp theo ta xét đỉnh 7, tương tự đỉnh 2, với tổng density là 0.48 cho cộng đồng 2 và 0 cho cộng đồng 1 do đó đỉnh 7 vẫn sẽ nằm ở cộng đồng 2.



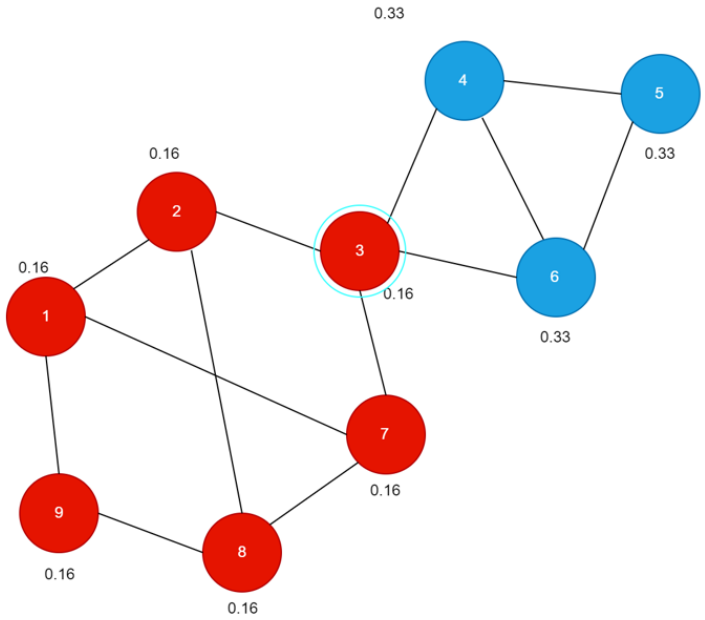
Bước 14: Tiếp theo ta xét đỉnh 6, với các đỉnh kề là 3, 4, và 5. Ta có tổng density của cộng đồng 1 (các đỉnh 4, và 5) là 0.66 nhưng tổng density của cộng đồng 2 (đỉnh 3) là 0.16 do đó đỉnh 6 vẫn sẽ thuộc vào cộng đồng 1.



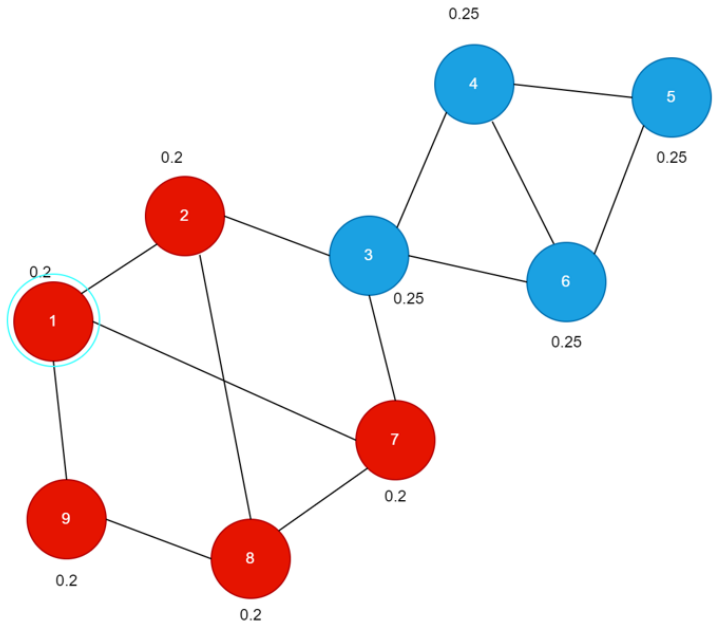
Bước 15: Ta xét đỉnh 4, tương tự tính toán cho đỉnh 6 ta vẫn sẽ có đỉnh 4 thuộc vào cộng đồng 1 với tổng density là 0.66.



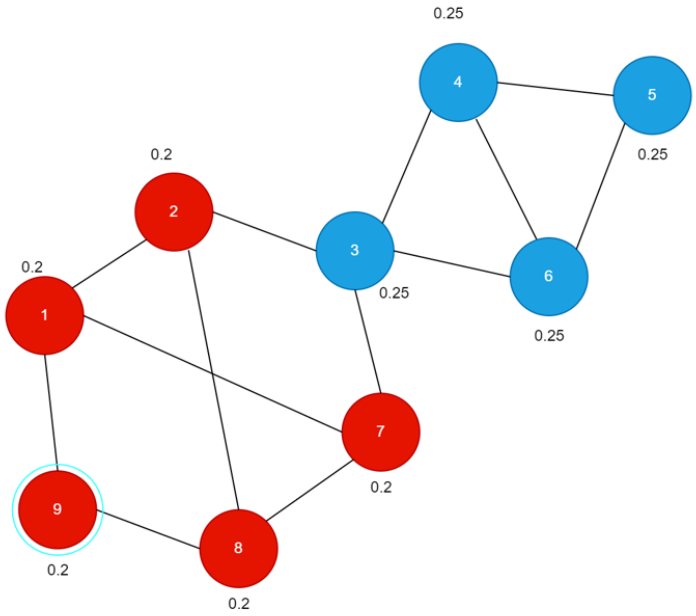
Bước 16: Bây giờ ta xét đỉnh 3, tại đây ta thấy đỉnh 3 đang thuộc vào cộng đồng 2 với tổng density hiện tại là 0.32, nhưng tổng density của cộng đồng 1 (chứa đỉnh kề 4 và 6) là 0.66 do đó đỉnh 3 bây giờ sẽ không còn thuộc vào cộng đồng 2 nữa do liên kết yếu hơn so với cộng đồng 1. Sau đó ta tính lại density của cộng đồng 1, giờ ta có cộng đồng 1 hiện đang có 4 đỉnh do đó density của cộng đồng sẽ là $\frac{1}{4} = 0.25$, và cộng đồng 2 mất đi 1 đỉnh do đó density sẽ là $\frac{1}{5} = 0.2$.



Bước 17: Tiếp theo ta sẽ xét đỉnh 2, ta có đỉnh 2 vẫn sẽ thuộc cộng đồng 2 vì tất cả các đỉnh kề của 2 đều thuộc cộng đồng 2.

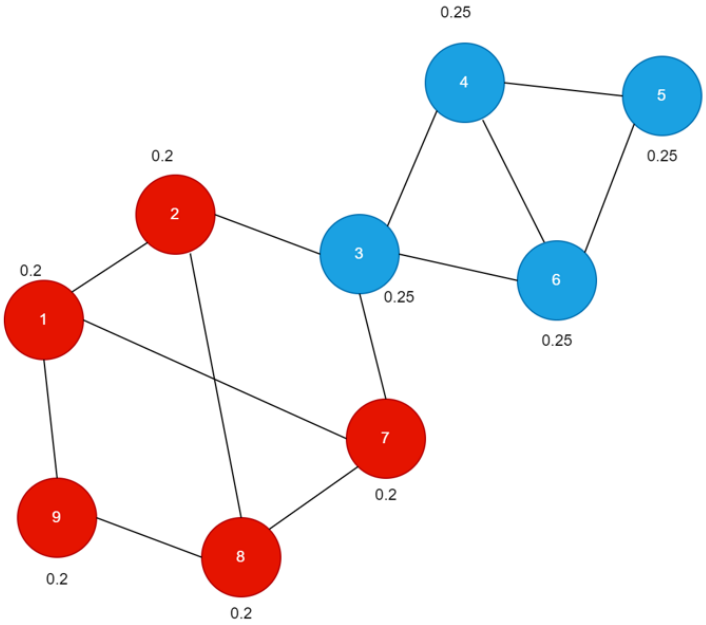


Bước 18: Tiếp theo ta xét đỉnh 9, tương tự ta vẫn sẽ có đỉnh 9 thuộc vào cộng 2, kết cấu mạng vẫn giữ nguyên.

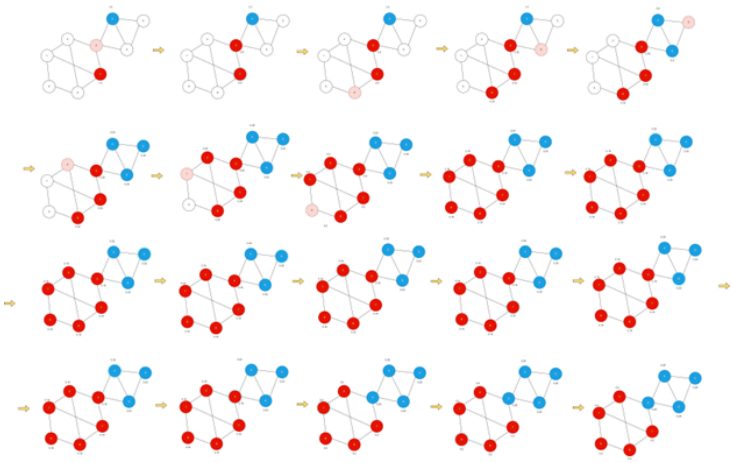


Bước 19: Sau khi đã duyệt qua xong tất cả đỉnh, ta sẽ tiếp tục thực hiện lại các bước duyệt do ta đã thay đổi cộng đồng của đỉnh 3 do đó có thể mạng sẽ còn thay đổi tiếp.

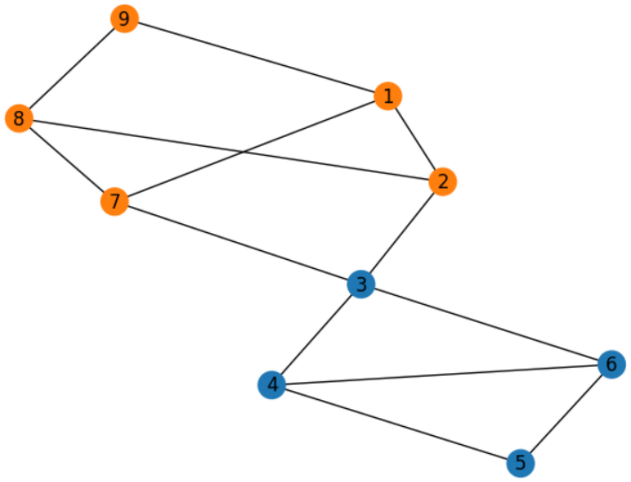
Bước 20: Do sau khi thay đổi đỉnh 3, mạng đã ổn định nên ta có kết quả sau khi phân chia cộng đồng



Cuối cùng ta có tổng quan các bước làm:



Kiểm tra lại thuật toán: Tương tự ở trên nhóm tiếp tục sử dụng code để kiểm tra lại các bước tính toán của nhóm



Kết quả cho thấy tính toán của nhóm giống như chạy bằng code.

3 ProtGNN: Towards Self-Explaining Graph Neural Networks

3.1 Introduction

Ngày nay với sự phát triển của dữ liệu đặc biệt là dữ liệu dạng đồ thị (Social networks, financial data, . . .) đã làm cho Graph Neural Networks (GNNs) trở nên ngày càng phổ biến và được ứng dụng rộng rãi trong giới nghiên cứu khoa học máy tính.

GNNs dựa trên ý tưởng của mô hình (paradigm) message passing [Ý tưởng giống Attention mechanism] để biểu diễn các nodes.

Message Passing Paradigm: $h_v^{(t)} = f_{upd}(h_v^{(t)}, m_v^{(t)})$, trong đó $m_v^{(t)} = f_{agg}(h_v^{(t-1)}, h_u^{(t-1)})$ với $h_u^{(t-1)}$ là biểu diễn của các neighbor nodes.

Nhờ vào khả năng biểu diễn các nodes tốt, GNNs đã đạt được nhiều SOTA trên nhiều bài toán ở dạng đồ thị như: node classification, graph classification, link prediction, ...

Mặc dù đạt được kết quả tốt nhưng GNNs thuộc dạng black box model, chúng ta không thể biết quá trình biểu diễn và đưa ra quyết định của các mô hình GNNs do đó với một số lĩnh vực như y sinh GNNs vẫn chưa thể được áp dụng.

Để giải quyết vấn đề black box, đã có nhiều công trình nghiên cứu để giải thích cho các quyết định dự đoán (prediction) của GNNs dựa trên phương pháp post-hoc (sử dụng một mô hình khác để giải thích cho prediction của GNNs). Các phương pháp dựa trên post-hoc có thể sẽ đưa ra các giải thích không chính xác hoặc không đầy đủ, do đó nhóm tác giả đã đề xuất phương pháp dựa trên prototype.

Prototype là phương pháp giải thích cho GNNs dựa trên các ví dụ đã được học (đã giải thích được). Khác với post-hoc mô hình nhóm tác giả đề xuất có thể giải thích cho các prediction chỉ dựa vào 1 mô hình đưa ra prediction. Bên cạnh đó mô hình của nhóm tác giả đề xuất có khả năng giải quyết 2 vấn đề khi sử dụng prototype learning:

- Sự rời rạc của các cạnh dẫn đến khó khăn cho việc visualization và projection
- Tính chất tổ hợp của đồ thị (Combinatorial nature of graph) là cản trở cho việc xây dựng mô hình tự giải thích (self-explaning) về khía cạnh hiệu quả và độ chính xác. structure

Đề xuất chính: ProtGNN sử dụng các GNNs làm encoder cho ProtGNN, các prediction cho các inputs mới được dựa trên sự tương đồng giữa các prototype của các prototype layers. Thêm vào đó là sử dụng Monte-Carlos Search Tree để giải quyết hiệu quả projection và visualization.

Tương tự ProtGNN, ProtGNN+ được thêm vào một module để sampling subgraph để so sánh tương đồng giữa các prototype giúp giải thích hiệu quả hơn và tốt hơn.

Một số thách thức và hạn chế:

- Với module lấy mẫu đồ thị con có điều kiện, một hạn chế được đề cập là training bổ sung dữ liệu. Cũng là hạn chế của ProtGNN, nên MCTS vẫn được sử dụng trong bước Prototype Projection để ổn định tối ưu.
- Khi thực hiện đánh giá ProtGNN và ProtGNN*, tác giả bài báo đã so sánh với các biến thể của GNN bao gồm: Graph Convolutional Network (GCN), Graph Attention Network (GAT), Graph Isomorphism Networks (GIN). Có thể thấy kết quả của ProtGNN và ProtGNN+ có kết quả rất tốt, nhưng điểm hạn chế về chi phí thời gian train trên bộ dữ liệu. Thì ProtGNN và ProtGNN+ đều kém hiệu quả hơn so với GCN, đặc biệt ProtGNN+* chiếm tới hơn 2 giờ cho việc train dữ liệu. Nhưng dù sao, chi phí thời gian của ProtGNN và ProtGNN+ có thể chấp nhận được.

3.2 PROBLEM STATEMENT

Mục tiêu của bài toán: học các mẫu đồ thị tiêu biểu đại diện có thể được sử dụng để phân loại và giải thích tương tự.

3.2.1 Kiến trúc ProtGNN

$(\{x_i, y_i\})_{i=1}^n$ là tập huấn luyện được gán nhãn với x_i là đồ thị đầu vào và $y_i \in \{1, \ldots, C\}$ là tập nhãn của đồ thị.

Với mục đích để học các mẫu đồ thị tiêu biểu (prototype) đại diện có thể được sử dụng để tham chiếu phân loại và giải thích tương tự.

- Tham chiếu phân loại (Classification references): Là những mẫu đồ thị hoặc đại diện mà mô hình đã học và phân loại các đối tượng trong đồ thị. Khi một đối tượng mới cần được phân loại, mô hình sẽ so sánh đối tượng đó với các tham chiếu phân loại để xác định lớp hoặc nhãn của nó.
- Giải thích tương tự (Analogical explanations): Khi giải thích một dự đoán của mô hình, các giải thích dựa trên Analogic được sử dụng để làm rõ quyết định của mô hình.

Với một đồ thị đầu vào mới, độ tương đồng của nó với mỗi đại diện được tính trong latent space. Latent space Là một không gian trừu tượng trong đó các đối tượng hoặc dữ liệu được biểu diễn dưới dạng các vector có số chiều nhỏ hơn so với không gian ban đầu (PCA, t-SNE để tạo latent space).

Dự đoán của mẫu mới có thể được suy ra và giải thích bằng cách sử dụng các mẫu đặc trưng đồ thị tiêu biểu tương đồng của nó.

Kiến trúc của ProtGNN gồm 3 giai đoạn chính: GNN Encoder f, Prototype Layer g_p và Fully Connected Layer c [10]. Cụ thể được tổng quan như sau:

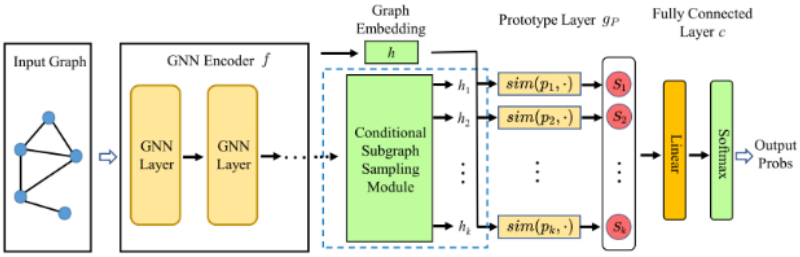


Figure 3: ProtGNN: The architecture [10]

- **Input**: Đầu vào là 1 đồ thị x_i cho trước.
- **GNN Encoder** f: Ánh xạ toàn bộ đồ thị thành nhúng đồ thị đơn (graph embedding) h với chiều dài cố định. Encoder có thể là bất kỳ mạng GNN nào chẳng hạn như GCN, GAT hoặc GIN. Có thể thu được bằng cách tính tổng hay max pooling của các đặc trưng từ lớp cuối cùng của mạng GNN.
- **Graph Embedding** h: Có thể thu được bằng cách tính tổng hoặc lấy giá trị lớn nhất của lớp GNN cuối cùng.
- **Prototype Layer** g_p : cấp lượng prototypes đã xác định trước m cho mỗi lớp. Trong đợt cuối ProtGNN đã huấn luyện, mỗi lớp đại diện cho một tập các prototypes)đã học. Prototypes này nên nắm bắt được những mẫu đồ thị quan trọng nhất để xác định đồ thị thuộc từng lớp. Prototype layer tính toán các điểm tương đồng (similarity scores)

$$sim\left(p_k, h\right) = \log\left(\frac{||p_k - h||_2^2 + 1}{||p_k - h||_2^2 + \epsilon}\right) \tag{3.1}$$

  Trong đó:

  - p_k: là prototype thứ k có cùng kích thước với embedding h.
  - Hàm tương đương được thiết kế sao cho giảm một cách đơn điệu với $(||p_k - h||)_2 > 0$.
  - ϵ là giá trị rất nhỏ.
- **The fully Connected Layer** c : với giá trị tương đồng (similarity scores) mà fully connected layer với softmax tính được xác suất đầu ra cho mỗi lớp.

Tóm lại, ProtGNN tính điểm tương đồng $sim(p_k, h)$ giữa graph embedding và prototypes đã học trong prototype layer. Để tăng tính tường minh, module lấy mẫu có điều kiện của đồ thị con (Conditional Subgraph Sampling Module) được tính hợp vào ProtGNN+ để đầu ra các đồ thị con giống nhất với mỗi prototypes đã học.

3.2.2 Monte-Carlo Tree Search

Monte Carlo tree search (MCTS) [2] là phương pháp tìm kiếm được ứng dụng trong nhiều lĩnh vực khác nhau. Phương pháp này tạo giả lập lặp lại để sinh sub-cây để lấy được kết quả đầu ra tối ưu. Thuật toán: - Selection: bắt đầu từ nút gốc R và duyệt xuống cây bằng cách chọn nút con dựa trên một quy ước chọn. - Expansion: Khi đi đến node lá, mở rộng nó bằng cách thêm nút con biểu diễn đường đi hoặc hành động có khả năng. - Simulation (Rollout): Với mỗi nút con, thực hiện mô phỏng từ nút đó tới trạng thái cuối. Chiến lược mô phỏng có thể là ngẫu nhiên hoặc heuristic. - Backpropagation:Sau khi quá trình triển khai hoàn tất, hãy cập nhật số liệu thống kê của tất cả các nút đã truy cập trong giai đoạn lựa chọn và mở rộng. Điều này bao gồm cập nhật số lượt duyệt qua và phần thưởng tích lũy. Thuật toán sẽ chạy cho đến khi đạt đến giới hạn thời gian cụ thể. Do tính hiệu quả của phương pháp MCTS, nhóm tác giả đã áp dụng phương pháp này để lấy mẫu đồ thị con trên pha chiếu (projection).

3.2.3 Learning Object

Mục tiêu là học mô hình ProtGNN đều đạt được độ chính xác và độ tường minh ổn định.

- Về tính chính xác, ta tối thiểu hoá hàm mất mát crossentropy trên tập dữ liệu huấn luyện:

$$L_(crossentropy) = \frac{1}{n}\sum_{i=1}^{n} CrsEnt\left(c * g_p * f(x_i)(, y)_i\right) \tag{3.2}$$

- Về tính tường minh, ta sẽ xem xét một số ràng buộc trong việc xây dựng các prototypes cho lớp tương ứng:
  + Cluster cost: Khuyến khích mỗi nhúng đồ thị nằm gần ít nhất một prototype của lớp tương ứng. Điều này giúp cho các prototypes thể hiện tính chất chung của lớp và đại diện cho các đồ thị thuộc cùng lớp.
  + Seperation cost: Khuyến khích mỗi nhúng đồ thị nằm cách xa các prototypes của các lớp không tương ứng. Điều này đảm bảo các prototypes không bị nhầm lẫn và tăng tính khác biệt giữa các lớp.

+ Diversity cost: Khuyến khích tính đa dạng của các prototypes đã học bằng cách "phạt" các prototypes quá gần nhau. Điều này nhằm đảm bảo các prototypes không bị trùng lắp và đa dạng về thông tin.

- Tóm lại, hàm mục tiêu mà ta nhằm tối thiểu hoá là:

$$L_{total} = L_{crossentropy} + \lambda_1 * L_{Cluster} + \lambda_2 * L_{Seperation} + \lambda_3 * L_{Diversity} \tag{3.3}$$

$$L_{Cluster} = \frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \in P_{y_i}} ||f(x_i) - p_j||_2^2 \tag{3.4}$$

$$L_{Separation} = -\frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \notin P_{y_i}} ||f(x_i) - p_j||_2^2 \tag{3.5}$$

$$L_{Diversity} = \sum_{k=1}^{C} \sum_{\substack{i \neq j \\ p_i, p_j \in P_k}} max\left(0, \cos\left(p_i, p_j\right) - s_{max}\right) \tag{3.6}$$

+ Với $\lambda_1, \lambda_2, \lambda_3$ là những siêu tham số để điều chỉnh trọng số của các hàm mất mát.
+ P_{y_i} là tập hợp các prototypes thuộc lớp y_i.
+ Hàm mất mát Div sử dụng ngưỡng s_{max} để đo lường độ tương đồng cosine.

3.2.4 Prototype Projection

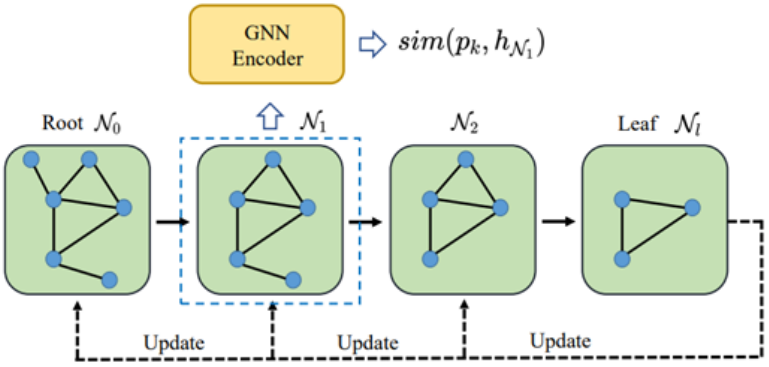
Prototypes đã học là những nhúng vector mà không có khả năng giải thích trực tiếp. Với mục đích cải thiện tính giải thích và trực quan, ta thiết kế một quy trình tham chiếu thực hiện trong giai đoạn huấn luyện.

Cụ thể, ta tham chiếu mỗi prototype p_j ($p_j \in P_k$) vào đồ thị con huấn luyện tiềm năng gần nhất từ cùng lớp với p_j. Như vậy, mỗi prototype tương đương với một đồ thị con, điều này dễ hiểu và có thể giải thích bởi con người. Để giảm thiểu chi phí tính toán, bước tham chiếu chỉ được thực hiện sau mỗi vài epoch huấn luyện.

$$p_j \leftarrow \arg\min_{\widetilde{h} \in \varkappa_j} \left|\left|\widetilde{h} - p_j\right|\right|_2 \tag{3.7}$$

$$\varkappa_j = \left\{\widetilde{h} : f\left(\widetilde{x}\right), \widetilde{x} \in Subgraph\left(x_i\right), \forall i \ s.t. \ y_i = k\right\}.$$

Khác với dữ liệu lưới như hình ảnh, đặc tính tổ hợp của đồ thị làm cho việc tìm đồ thị con gần nhất thông qua liệt kê trở nên bất khả thi. Trong phép chiếu prototype của đồ thị, ta sẽ sử dụng thuật toán Monte Carlo tree search (MCTS – được đề cập ở phần trên) như là một thuật toán tiềm kiếm nhằm khám phá các đồ thị con.



- Ta xây dựng một cây tìm kiếm trong đó nút gốc (Root) liên kết với đồ thị đầu vào còn các nút khác tương ứng với từng đồ thị con được khám phá. Cụ thể, ta định nghĩa mỗi nút trong cây tìm kiếm T là N_i và N_0 là nút gốc.
- Các cạnh được biểu thị cho việc cắt tỉa. Trong cây tìm kiếm, đồ thị liên quan với node con có thể thu được bằng tỉa các node từ đồ thị tương ứng của nút cha. Giới hạn không gian tìm kiếm, ta thêm hai ràng buộc bổ sung: N_i phải là đồ thị con kết nối và kích thước đồ thị con được tham chiếu phải nhỏ.

Trong quá trình tìm kiếm, MCTS ghi nhận thống kê về số lần truy cập và phần thưởng để hướng dẫn việc khám phá, giảm không gian tìm kiếm. Đặc biệt, với cặp node và việc tỉa nhánh (N_i, a_j), ta giả định rằng đồ thị con N_j thu được bởi hành động a_j từ N_i. Thuật toán MCTS có 4 biến cho cặp (N_i, a_j):

- $C(N_i, a_j)$ được định nghĩa là số lần chọn hành động a_j cho node N_i.
- $W(N_i, a_j)$ là tổng phần thưởng cho số tất cả lần truy cập (N_i, a_j).
- $Q(N_i, a_j)$ là trung bình phần thưởng cho nhiều lần truy cập.
- $R(N_i, a_j)$ là phần thưởng tức thời khi chọn hành động a_j trên N_i, được đo bằng sự tương đồng giữa prototypes và nhúng đồ thị con. Nhúng đồ thị con thu được bằng mã hoá đồ thị con với GNN encoder f.

MCTS tìm kiếm các đồ thị con gần nhất qua nhiều lần lặp mà mỗi lần lặp có 2 giai đoạn:

- **Giai đoạn đầu**: MCTS chọn một đường đi bắt đầu từ gốc N_0 đến một nút lá N_l. Để giữ cho các đồ thị con kết nối, ta chọn cắt tỉa các node ngoại biên có bậc nhỏ nhất. Node lá được xác định dựa trên số lượng node trong đồ thị con sao cho $|N_l| \leq N_{min}$. Tiêu chí hành động cho node N_i là:

$$a^* = \text{argmax}a_j Q(N_i, a_j) + U(N_i, a_j) \tag{3.8}$$

$$U\left(N_i, a_j\right) = \lambda R\left(N_i, a_j\right) \frac{\sqrt{\sum_k C\left(N_i, a_k\right)}}{1 + C\left(N_i, a_j\right)}$$

+ λ là một siêu tham số (hyper-parameter) nhằm điều chỉnh sự cân đối giữa khám phá và khai thác.
+ Mục đích nhằm ưu tiên chọn các node con có số lần truy cập thấp để khám phá các hành động cắt tỉa khác nhau, nhưng khi tiến đến vô cực thì nó ưu tiên các hành động dẫn đến điểm tương đồng cao hơn.

- **Giai đoạn sau**: thống kê của tất cả cặp node và action được chọn cho đường đi này được cập nhật như sau:

$$C\left(N_i, a_j\right) = C\left(N_i, a_j\right) + 1 \tag{3.9}$$

$$W\left(N_i, a_j\right) = W\left(N_i, a_j\right) + sim\left(p_k, h_{N_l}\right) \tag{3.10}$$

3.2.5 ProtGNN+ (Lấy mẫu đồ thị con có điều kiện)

Giới thiệu Để cải tiến, tác giả đề suất thêm ProtGNN+ với module lấy mẫu đồ thị con có điều kiện mới.

Trong ProtGNN+, chúng ta không chỉ cho thấy độ tương đồng mà còn xác định phần nào của đồ thị đầu vào là tương đồng nhất với mỗi prototype, điều này được xem như một phần của quá trình lập luận.

Như hình trực quan ProtGNN phía trên, Module này xuất ra nhiều nhúng đồ thị con khác nhau. Điều này chúng ta có thể thực hiện bằng MCTS, nhưng:

- Độ phức tạp sẽ tăng cấp số nhân theo kích thước của đồ thị.
- Khó trong việc song song hóa và tổng quát hóa.

→ Sử dụng phương pháp tham số hóa để chọn đồ thị con tương đồng có điều kiện từ prototype đã đưa.

Ý tưởng Thông thường, ta sẽ để $e_{ij} \in 0, 1$ là biến nhị phân để xác định xem cạnh giữa node i và j có được chọn không. Ma trận của e_{ij} gọi là ε. Tối ưu mục tiêu của module lấy mẫu đồ thị con có điều kiện:

$$\max sim\left(p_k, f(\mathcal{G}_s)\right) \;\; s.t. |\mathcal{G}_s| \leq B \tag{3.11}$$

Trong đó $\mathcal{G}_s$ đồ thị con được chọn với ma trận kề là ε. B là kích thước lớn nhất của đồ thị con.

Thuật toán Nhưng tính tổ hợp và tính rời rạc của đồ thị khiến việc tối ưu hóa trực tiếp hàm mục tiêu trờ nên khó khăn. Tác giả đơn giản hóa bài toán bằng cách giả định đồ thị giải thích là một đồ thị ngẫu nhiên Gilbert (Gilbert 1959) trong đó trạng thái mỗi cạnh độc lập nhau. Hơn nữa, để dễ tính độ dốc (gradient) và cập nhật, chúng ta chuyển $\varepsilon \in \{0,1\}^{N\times N}$ thành không gian lồi $\varepsilon \in [0,1]^{N\times N}$. N là số lượng node trong đồ thị đầu vào. Để hiệu quả và tổng quát hóa, tác giả sử dụng deep neural network để học e_{ij}:

$$e_{ij} = \sigma\left(MLP_\theta\left([z_i; z_j; p_k]\right)\right) \tag{3.12}$$

Trong đó $\sigma(\cdot)$ là hàm sigmoid. MLP là neural network nhiều lớp tham số hóa với θ và $[\cdot;\cdot;\cdot]$ là phép nối. z_i và z_j là nhúng node thu được từ GNN Encoder, nó mã hóa đặc trưng và cấu trúc thông tin lân cận của các node. Và mục tiêu của biểu thức (12) trở thành:

$$\max \theta sim\left(p_k, f\left(\mathcal{G}_s\right)\right) - \lambda_b R_b \tag{3.13}$$
$$R_b = ReLU\left(\sum_{eij\in\varepsilon} e_{ij} - B\right)$$

Trong đó λ_b là trọng số cho việc điều cỉnh ngân sách R_b (budget regularization). Trong thí nghiệm, tác giả sử dụng stochastic gradient descent để tối ưu hóa hàm mục tiêu.

So sánh với MCTS Module lấy mẫu đồ thị con có điều kiện mà tác giả thiết kế hiệu quả hơn nhiều so với MCTS và dễ dàng để tính toán song song. Tham số của module được cố định và độc lập với kích thước đồ thị. Để lấy mẫu từ đồ thị với số lượng $|\varepsilon|$ cạnh, độ phức tạp thời gian là $\mathcal{O}(|\varepsilon|)$.

Một điều giới hạn của module này là nó yêu cầu huấn luyện bổ sung. Do đó, MCTS vẫn được sử dụng trong bước chiếu prototype của ProtGNN+ để đảm bảo tính ổn định của quá trình tối ưu hóa.

3.2.6 Theorem on Subgraph Sampling

Giả sử số lượng prototype là như nhau ở mỗi lớp, ký hiệu là m. Cứ mỗi lớp k, trọng số liên kết trong layer cuối c giữa một prototype của lớp k và đầu ra logit của lớp k là 1, ngược lại trọng số liên kết trong layer cuối giữa một prototype không thuộc lớp k và đầu ra logit thuộc lớp k là 0. Ta ký hiệu p_l^k là prototype thứ l trong lớp k và h_l^k hành động nhúng của đồ thị con đã tỉa. ProtGNN và ProtGNN+ đều có cùng graph encoder f. Tại đây ta giả định tồn tại một δ với $0 < \delta < 1$:

- Đối với lớp đúng, chúng ta có:

$$\left|\left|h - h_l^k\right|\right|_2 \leq \left(\sqrt{1+\delta} - 1\right) \left|\left|h - p_l^k\right|\right|_2 \ và \ \left|\left|h - p_l^k\right|\right|_2 \leq \sqrt{1+\delta} \tag{3.14}$$

- Đối với lớp sai, chúng ta có:

$$\left|\left|h - h_l^k\right|\right|_2 \leq \theta \left|\left|h - p_l^k\right|\right|_2 - \sqrt{\varepsilon} \ và \ \theta = \min\left(\sqrt{1+\delta} - 1, 1 - \frac{1}{\sqrt{2-\delta}}\right) \tag{3.15}$$

- Đối với đồ thị đầu vào được phân loại đúng trong ProtGNN, nếu các đầu ra logits giữa hai lớp hàng đầu ít nhất là $2m\ log\left((1+\delta)(2-\delta)\right)$ thì ProtGNN+ cũng có thể phân loại đúng đồ thị đầu vào như vậy.

Mục đích chính của định lý này là chứng minh việc lấy mẫu đồ thị con không làm thay đổi quá nhiều nhúng đồ thị thì ProtGNN+ sẽ có các dự đoán đúng tương tự ProtGNN.

Chứng minh cụ thể Với mọi lớp k , có $L_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\}$ ký hiệu là tổng điểm đóng góp của đồ thị x thuộc lớp k trong ProtGNN và $L\prime_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\}$ ký hiệu là tổng điểm đóng góp của đồ thị trong ProtGNN+, theo như công thức (1) ta có:

$$L_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\} = \sum_{l=1}^{m} \log\left(\frac{\left|\left|h - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}\right) \ (ProtGNN)$$

$$L\prime_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\} = \sum_{l=1}^{m} \log\left(\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon}\right) \ (ProtGNN+)$$

Tiếp theo, ta tính khoảng cách giữa các điểm đóng góp tổng hợp, được ký hiệu là Δ_k là:

$$\begin{aligned}
\Delta_k &= L'_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\} - L_k\left\{x, \left\{p^k\right\}_{l=1}^m\right\} \\
&= \sum_{l=1}^{m} \log\left(\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon}\right) - \sum_{l=1}^{m} \log\left(\frac{\left|\left|h - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}\right) \\
&= \sum_{l=1}^{m} \log\left(\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} * \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon}\right)
\end{aligned} \tag{3.16}$$

Trường hợp đúng lớp: Ta có bất đẳng thức sau theo như công thức (14) – (Khai thác giới hạn dưới).

$$\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \geq \frac{1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \geq \frac{1}{2-\delta} \tag{3.17}$$

- Tiếp theo ta có bất đẳng thức tam giác sau: $\left|\left|h_l^k - p_l^k\right|\right| \leq \left|\left|h - p_l^k\right|\right| + \left|\left|h - h_l^k\right|\right|$. $(*)$ Để giải thích cho bất đẳng thức trên tại sao tuân theo bất đẳng thức tam giác thì ta có lần lượt các ý nghĩa của các trị tuyệt đối như sau:

+ $h_l^k - p_l^k$: Là khoảng cách từ nhúng đồ thị con tới nhúng prototype ban đầu.
+ $h - p_l^k$: Là khoảng cách từ toàn nhúng đồ thị tới nhúng prototype ban đầu
+ $h - h_l^k$: Là khoảng cách từ toàn nhúng đồ thị tới nhúng đồ thị con.

- Vì thế ta có bất đẳng thức sau:

$$\frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon} \geq \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left(\left|\left|h - p_l^k\right|\right|_2 + \left|\left|h - h_l^k\right|\right|_2\right)^2 + \epsilon}$$

- Theo công thức (14) ta có $\left|\left|h - h_l^k\right|\right|_2 \leq \left(\sqrt{1+\delta} - 1\right)\left|\left|h - p_l^k\right|\right|_2$ vậy bất đẳng thức trên thành:

$$\begin{aligned}
\frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon} &\geq \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left(\left|\left|h - p_l^k\right|\right|_2 + \left|\left|h - h_l^k\right|\right|_2\right)^2 + \epsilon} \\
&\geq \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left(\left|\left|h - p_l^k\right|\right|_2 + \left(\sqrt{1+\delta} - 1\right)\left|\left|h - p_l^k\right|\right|_2\right)^2 + \epsilon} \\
&\geq \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{(1+\delta)\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon} \\
&\geq \frac{1}{(1+\delta)}
\end{aligned} \tag{3.18}$$

- Từ bất đẳng thức (17) và (18) ta có Δ_k cho trường hợp đúng lớp là:

$$\begin{aligned}
\Delta_{k-correct} &= m\ log\left(\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} * \frac{\left|\left|h - p_l^k\right|\right|_2^2 + \epsilon}{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + \epsilon}\right) \geq mlog\left(\frac{1}{(1+\delta)(2-\delta)}\right) \\
&= -m\ log\left((1+\delta)(2-\delta)\right)
\end{aligned}$$

Trường hợp sai lớp: ta có bất đẳng thức sau theo bất đẳng thức tam giác (*) – (Khai thác giới hạn trên)

$$\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \leq \frac{\left(\left|\left|h - p_l^k\right|\right|_2 + \left|\left|h - h_l^k\right|\right|_2\right)^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1}$$

- Theo công thức (15) ta có:

$$\left|\left|h - h_l^k\right|\right|_2 \leq \left(\sqrt{1+\delta} - 1\right)\left|\left|h - p_l^k\right|\right|_2 - \sqrt{\epsilon} \leq \left(\sqrt{1+\delta} - 1\right)\left|\left|h - p_l^k\right|\right|_2$$

+ Bất đẳng thức thành:

$$\frac{\left|\left|h_l^k - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \leq \frac{\left(\left|\left|h - p_l^k\right|\right|_2 + \left|\left|h - h_l^k\right|\right|_2\right)^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \leq \frac{(1+\delta)\left|\left|h - p_l^k\right|\right|_2^2 + 1}{\left|\left|h - p_l^k\right|\right|_2^2 + 1} \leq 1 + \delta \tag{3.19}$$

- Tiếp theo ta có bất đẳng thức sau: $\left|\left|h-p_l^k\right|\right|_2 \leq \left|\left|h-h_l^k\right|\right|_2 + \left|\left|h_l^k-p_l^k\right|\right|_2$

$$\frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon} \leq \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left(\left|\left|h-p_l^k\right|\right|_2-\left|\left|h-h_l^k\right|\right|_2\right)^2+\epsilon}$$

+ Theo công thức (15) ta có:

$$\left|\left|h-h_l^k\right|\right|_2 \leq \left(1-\frac{1}{\sqrt{2-\delta}}\right)\left|\left|h-p_l^k\right|\right|_2 - \sqrt{\epsilon} \leq \left(1-\frac{1}{\sqrt{2-\delta}}\right)\left|\left|h-p_l^k\right|\right|_2$$

$$\Rightarrow \frac{1}{\sqrt{2-\delta}}\left|\left|h-p_l^k\right|\right|_2 \leq \left|\left|h-p_l^k\right|\right|_2 - \left|\left|h-h_l^k\right|\right|_2$$

+ Bất đẳng thức thành:

$$\frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon} \leq \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left(\left|\left|h-p_l^k\right|\right|_2-\left|\left|h-h_l^k\right|\right|_2\right)^2+\epsilon} \leq \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left(\frac{1}{\sqrt{2-\delta}}\left|\left|h-p_l^k\right|\right|_2\right)^2+\epsilon}$$

Mà $\frac{1}{2} < \frac{1}{2-\delta} < 1$ do $0 < \delta < 1$ nên $\frac{1}{2-\delta}\left|\left|h-p_l^k\right|\right|_2^2+\epsilon \geq \frac{1}{2-\delta}\left|\left|h-p_l^k\right|\right|_2^2 + \frac{1}{2-\delta}\epsilon$.

+ Vậy bất đẳng thức trên thành:

$$\frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon} \leq \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left(\left|\left|h-p_l^k\right|\right|_2-\left|\left|h-h_l^k\right|\right|_2\right)^2+\epsilon} \leq \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left(\frac{1}{\sqrt{2-\delta}}\left|\left|h-p_l^k\right|\right|_2\right)^2+\epsilon} \leq 2-\delta \tag{3.20}$$

- Từ (19) và (20) ta có Δ_k cho lớp sai là:

$$\Delta_{k-wrong} = mlog\left(\frac{\left|\left|h_l^k-p_l^k\right|\right|_2^2+1}{\left|\left|h-p_l^k\right|\right|_2^2+1} * \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon}\right) \leq mlog\left((1+\delta)(2-\delta)\right)$$

Vậy ta có:

$$\Delta_{k-correct} = mlog\left(\frac{\left|\left|h_l^k-p_l^k\right|\right|_2^2+1}{\left|\left|h-p_l^k\right|\right|_2^2+1} * \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon}\right) \geq -mlog\left((1+\delta)(2-\delta)\right)$$

$$\Delta_{k-wrong} = mlog\left(\frac{\left|\left|h_l^k-p_l^k\right|\right|_2^2+1}{\left|\left|h-p_l^k\right|\right|_2^2+1} * \frac{\left|\left|h-p_l^k\right|\right|_2^2+\epsilon}{\left|\left|h_l^k-p_l^k\right|\right|_2^2+\epsilon}\right) \leq mlog\left((1+\delta)(2-\delta)\right)$$

$$\Delta_{k-correct} - \Delta_{k-wrong} \geq -2mlog\left((1+\delta)(2-\delta)\right)$$

3.2.7 Thủ tục huấn luyện

Input: Training dataset $\{x_i, y_i\}_{i=1}^n$
Parameter: Training epochs T, Warm-up epoch T_w, Projection epoch T_p, Prototype projection period τ, ProtGNN+

```
Initialize model parameters.
for training epochs t = 1, 2, ··· , T do
    Optimizing objective function in Eq. (3)
    if t>T_p and t%τ = 0 then
        Performing prototype projection with MCTS
    end if
    if ProtGNN+ enabled and t>T_w then
        Optimizing the objective function in Eq. (14).
    end if
end for
Output: Trained model, prototype visualization
```

Trong giải thuật trên, chúng ta thấy được thủ tục huấn luyện của ProtGNN/ProtGNN+:

- Trước khi bắt đầu huấn luyện, chúng ta khởi tạo ngẫu nhiên các tham số mô hình. Chúng ta để w_c là trọng số ma trận của layer hoàn toàn kết nối c và $w_c^{(k,j)}$ là trọng số kết nối giữa output của prototype thứ j và logit của class k. Cụ thể, đối với class k, chúng ta gán $w_c^{(k,j)} = 1$ cho tất cả j với $p_j \in P_k v w_c^{(k,j)} = 0$ cho tất cả j với $p_j \notin P_k$. Hiểu đơn giản, việc khởi tạo w_h như vậy khiếu khích các prototype thuộc về class k học khái niệm ngữ nghĩa đặc trưng của class k.
- Sau khi bắt đầu huấn luyện, chúng ta cài đặt gradient descents để tối ưa hàm mục tiêu trong biểu thức (3).
- Nếu epoch huấn luyện lớn hơn epoch chiếu T_p, chúng ta thực hiện chiếu prototype sau mỗi vài epoch huấn luyện.
- Nếu chúng ta huấn luyện ProtGNN+, sau epoch khởi động T_w, module lấy mẫu đồ thị con có điều kiện và ProtGNN sẽ được tối ưu lặp khi tối ưu hóa của GNN Encoder và prototype đã ổn định.

3.2.8 ProtGNN cho các nhiệm vụ đồ thị tổng quát

Trong các phần trên, tác giả mô tả ProtGNN/ProtGNN+ thông qua sử dụng chúng để làm nhiệm vụ graph classification. Nhưng tác giả cũng đã đề cập, chúng cũng có thể sử dụng cho các nhiệm vụ khác như: node classification hay link prediction, một cách dễ dàng.

Ví dụ trong nhiệm vụ node classification, mục tiêu giải thích là cung cấp tiến trình

lập luận đằng sau dự đoán của node v_i. Giả sử GNN encoder có L layer, dự đoán của node v_i chỉ phụ thuộc vào đồ thị tính toán L-hop của nó. Do đó, chiếu prototype và lấy mẫu đồ thị con có điều kiện đều được thực hiện trong đồ thị tính toán L-hop.

3.3 Related Work

Khi ứng dụng của GNNs ngày càng phát triển, dẫn đến các dự đoán của GNNs ngày càng quan trọng. Để có thể áp dụng vào các vấn đề quan trọng đó thì khả năng giải thích đã được nghiên cứu và phát triển nhanh chóng. Dựa vào bài *A Taxonomic Survey* [9] của Yuan năm 2020, thì các thuật toán giải thích GNNs có thể chia thành các loại:

- Phương pháp dựa trên gradient/đặc trưng (Gradients/features-based methods).
- Phương pháp dựa trên sự thay đổi (Perturbation-based methods).
- Phương pháp phân rã (Decomposition methods).
- Phương pháp đại diện (Surrogate methods).

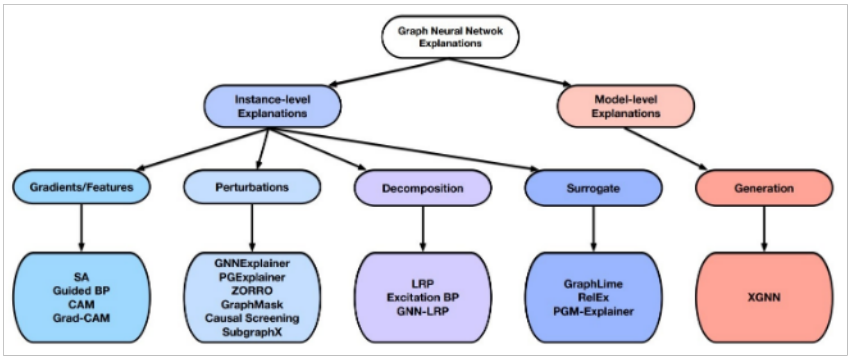


Figure 4: Sơ đồ phân loại các thuật toán giải thích GNN[9]

(Trong phần này, tác giả không đề cập đến giải thích cấp độ mô hình - XGNN)

Giải thích sơ lược về 4 phương pháp:

- **Phương pháp dựa trên gradient/đặc trưng**: Các thuật toán sử dụng gradient hoặc giá trị đặc trưng để chỉ ra mức độ quan trọng của các đặc trưng đầu vào, sử dụng các kỹ thuật giải thích trong hình ảnh cho đồ thị mà không cần tích hợp các thuộc tính của dữ liệu đồ thị.
- **Phương pháp dựa trên sự thay đổi**: Các thuật toán xem xét các thay đổi trong dự đoán trong khi thay đổi các đặc trưng đầu vào để xác định đặc trưng có tác động quan trọng.
- **Phương pháp phân rã**: Giải thích bằng cách phân rã các dự đoán ban đầu của mô hình thành nhiều thành phần và liên kết các thành phần này với đỉnh hoặc cạnh của đồ thị.

- **Phương pháp đại diện**: Dưa ra giải thích cục bộ cho một trường hợp xác định, dựa vào việc tạo ra các mẫu lân cận và khớp mô hình đơn giản có khả năng giải thích vào đó, ví dụ mô hình tuyến tính, cây quyết định,...

Dựa vào khái niệm bài báo đề cập, thì cả 4 phương pháp trên đều được gọi là các thuật toán "post-hoc explanation" tức là các thuật toán tạo ra giải thích ra sau khi đã có mô hình cần giải thích (GNNs). Khi so với đó thì ProtGNN, thuật toán "built-in interpretability", thuật toán có khả năng tự ra giải thích trong quá trình huấn luyện mô hình, có sức hấp dẫn hơn. Vì trong khi "post-hoc" không thể khớp hoàn toàn với mô hình gốc, thì nó có khả năng giải thích tự nhiên với độ chính xác.

3.4 EXPERIMENTAL EVALUATION

3.4.1 Chuẩn bị dữ liệu

Trong bài báo này, nhóm em tiến hành chạy lại thử nghiệm trên 5 bộ dữ liệu đề xuất:

- MUTAG dataset [5]: Bộ dữ liệu được sử dụng cho phân loại đồ thị. Nó chứa các biểu đồ biểu thị các hợp chất hóa học và nhiệm vụ cần làm là phân loại các hợp chất này thành 2 loại: gây đột biến và không gây đột biến.
  + Kích thước: bao gồm 177 Graphs.
  + Mỗi đồ thị đại diện cho một hợp chất hóa học, trong đó các nút biểu thị các nguyên tử và các cạnh biểu thị các liên kết hóa học giữa các nguyên tử.
  + Mỗi nút trong đồ thị được dán nhãn bằng các ký hiệu nguyên tử tương ứng (C, H, O,...).
  + Nhãn đồ thị: mỗi đồ thị được phân loại thành đột biến hoặc không gây đột biến.
- BBBP dataset (Blood-brain barrier penetration) [6]: được sử dụng để dự đoán liệu một hợp chất hóa học có thể đi qua hàng rào máu-não hay không.
  + Kích thước: Bộ dữ liệu chứa tổng số 2053 hợp chất hóa học.
  + Mỗi hợp chất hóa học được coi như 1 đồ thị phân tử, mà mỗi node là 1 nguyên tử, các cạnh biểu thị số liên kết hóa học giống như MUTAG dataset.
  + Trường SMILES đại diện cho cấu trúc phân tử.
  + Nhãn của đồ thị là các giá trị nhị phân.
- Graph-SST2 dataset [4]: là bộ dữ liệu chuẩn được sử dụng cho các tác vụ phân tích cảm xúc. Trong bộ dữ liệu Graph-SST2, mỗi câu được chuyển thành cấu trúc biểu đồ để nắm bắt các mối quan hệ cú pháp và ngữ nghĩa giữa các từ và cụm từ. Các nút trong tập dữ liệu đại diện cho các từ hoặc cụm từ riêng lẻ, các cạnh thể hiện mối quan hệ giữa chúng.
  + Kích thước: Bao gồm 700000 cạnh và hơn 700000 node.
- Graph-Twitter dataset [3]: là bộ dữ liêu được cào từ Twitter và đã được gán nhãn, quá trình gán nhãn được thực hiện thủ công.
  + Kích thước: Gồm 6248 tweets, bộ dữ liệu test có 692 tweets.
- BA-Shape dataset [8]: là bộ dữ liệu phân loại nút tổng hợp, bộ dữ liệu được tạo lần đầu tiên với đồ thị Barabasi-Albert (BA) có 300 nút và một bộ gồm 80 mạng có cấu trúc "ngôi nhà" năm nút điển hình, sau đó chọn ngẫu nhiên các nút trong đồ thị và đính kèm cấu trúc "ngôi nhà" vào đó. Nhiệm vụ là phân loại các nút xem nó

ở đầu, giữa hay cuối của ngôi nhà.

Khi nhóm em thử nghiệm mô hình trên các bộ dữ liệu này, Graph-Twitter không được tìm thầy và có một số lỗi khi tải bộ dữ liệu BA-Shapes. Vì vậy, nhóm em đã tiến hành thử nghiệm trên BA-2Motifs thay vì bộ dữ liệu BA-Shapes.

Nhìn chung, để đánh giá mô hình ta chia chia các bộ dữ liệu thành 3 nhóm. Nhóm đầu tiên là tập dữ liệu phân tử để phân loại đồ thị. Nhóm thứ hai là bộ dữ liệu đồ thị cảm xúc để phân loại đồ thị như đã nói ở trên và nhóm cuối cùng là bộ dữ liệu phân loại nút tổng hợp.

Đối với nhóm thứ hai, các câu trong các bộ dữ liệu này đã được phân tích cú pháp và nhúng để xây dựng đồ thị.

3.4.2 Kết quả

Để thử nghiệm mô hình, tác giả đã triển khai ba biến thể của GNN là GCN, GAT và GIN. Chỉ số được sử dụng để đánh giá là accuracy, bởi vì các tác vụ này được phân loại bằng nhãn. Ngoài ra, các tác giả đã sử dụng GNN làm cơ sở để so sánh mô hình ProtGNN/ProtGNN+ mà tác giả đề xuất.

Datasets	GCN			GIN			GAT		
	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+
MUTAG	73.3±5.8	**76.7±6.4**	73.3±2.9	**93.3±2.9**	90.7±3.2	91.7±2.9	75.0±5.0	78.3±4.2	**81.7±2.9**
BBBP	84.6±3.4	**89.4±4.1**	88.0±4.6	86.2±1.1	**86.5±1.6**	85.9±4.0	83.0±2.6	**85.9±2.5**	85.5±0.8
Graph-SST2	89.7±0.5	**89.9±2.4**	89.0±3.0	92.2±0.3	92.0±0.2	**92.3±0.4**	88.1±0.8	**89.1±1.2**	88.7±0.9
Graph-Twitter	67.5±1.9	**68.9±5.9**	66.1±6.5	66.2±1.3	75.2±2.8	**76.5 ±3.4**	**69.6±6.5**	64.8±4.0	66.4±3.3
BA-Shape	91.9±1.7	**95.7±1.4**	94.3±3.7	92.9±0.5	95.2±1.3	**95.5 ±2.4**	92.9±1.2	**93.4±3.4**	93.2±2.0

Trong bảng trên, chúng ta có thể thấy rằng các mô hình được đề xuất đạt được hiệu suất phân loại tương đối khi so sánh từng hàng theo các mô hình khác. Điều này cho thấy đề xuất của tác giả là hợp lý (Định lý 1).

Để kiểm tra kết quả của tác giả, nhóm em chạy ProtGNN/ProtGNN+ lần nữa trên bộ dữ liệu MUTAG , BBBP, Graph-SST2, và BA-2Motifs. Kết quả thu được như sau:

Datasets	GCN	GIN	GAT
	ProtGNN+	ProtGNN+	ProtGNN+
MUTAG	83.3	88.8	88.8
BBBP	91.6	90.6	84.7
Graph-SST2	88.2	91.6	79.8
BA-2Motifs	52.0	100.0	52.0

Có một số khác biệt nhỏ, nhưng nhìn chung kết quả mà nhóm em chạy giống với kết quả của các tác giả. Hơn nữa, khi thử nghiệm mô hình trên bộ dữ liệu BA-2Motifs, độ

chính xác của mô hình có lỗi xảy ra vì mô hình không học được gì khi khi nhóm em train mô hình.

Ngoài ra, có một điểm thú vị là khi sử dụng cấu trúc GCN và GAT, ProtGNN hoạt động tốt hơn ProtGNN+ trên hầu hết các bộ dữ liệu, trong khi ProtGNN+ với cấu trúc GIN hoạt động tốt hơn ProtGNN trên hầu hết các bộ dữ liệu. Vì vậy, nhóm em cho rằng kiến trúc để dựng mô hình ảnh hưởng đến kết quả mô hình.

3.4.3 Giải thích quy trình

Để giải thích quá trình suy luận của mô hình, tác giả đã thực hiện các nghiên cứu điển hình trên bộ dữ liệu MUTAG và Graph-SST2. Cụ thể hơn, với đầu vào x, mô hình tìm mức độ độ phù hợp (likelihood) trong mỗi lớp bằng cách so sánh nó với các prototype từ mỗi lớp. Mô-đun lấy mẫu đồ thị con có điều kiện tìm đồ thị con tương tự nhất trong x sau đó tính toán điểm tương đồng (similarity) và tính tổng chúng lại với nhau để đưa ra giá trị cuối cùng cho x thuộc mỗi lớp.

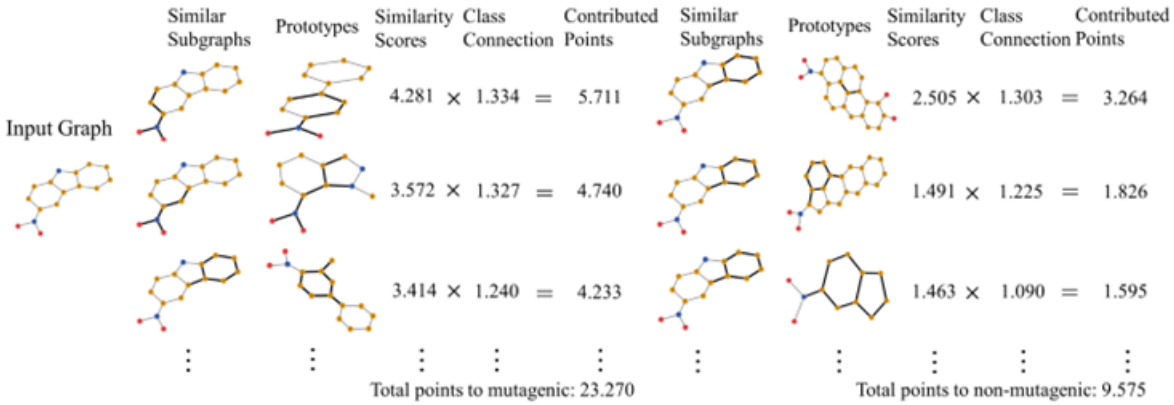


Figure 5: Giải thích quá trình của mô hình trên MUTAG

Trong hình trên, đồ thị đầu vào sẽ được lấy mẫu và so sánh với prototype của từng lớp, sau đó tính tổng tất cả các giá trị similarity để quyết định xem đồ thị đầu vào thuộc lớp gây đột biến hay không gây đột biến. Từ ví dụ trong kiến thức về lĩnh vực hình ảnh và hóa học, các vòng carbon và NO2 có xu hướng gây đột biến, chúng ta có thể thấy rằng các đồ thị con được lấy mẫu có thể bắt được nhóm NO2 và một phần của các vòng carbon, hơn nữa đồ thị đầu vào sẽ được phân loại thành chất gây đột biến - thỏa mãn yêu cầu bài toán. Hoặc, với bộ dữ liệu không có miền tri thức xác định, mô hình vẫn thực hiện tốt hơn. Nhìn chung, phương pháp đề xuất của tác giả cung cấp bằng chứng có thể giải thích được để hỗ trợ phân loại. Nếu không sử dụng mô hình bổ sung, ProtGNN có

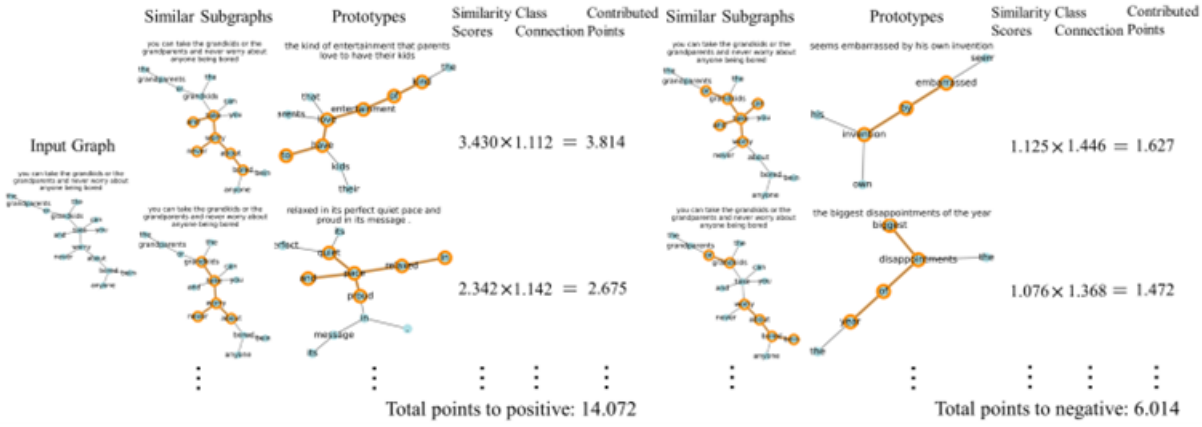


Figure 6: Giải thích quá trình của mô hình trên Graph-SST2

thể tự khám phá và đạt được kết quả tốt trên năm bộ dữ liệu.

3.4.4 Trực quan hóa Prototype

Để xác minh tính hiệu quả của việc nhúng prototype (prototype embedding), tác giả cũng thực hiện trực quan hóa trên bộ dữ liệu BBBP bằng cách sử dụng phương pháp $t-SNE$. Kết quả cho thấy các nguyên mẫu đã học có thể chiếm trung tâm của không gian nhúng đồ thị. Trong đó, các chấm là dữ liệu BBBP và ngôi sao là prototype embedding

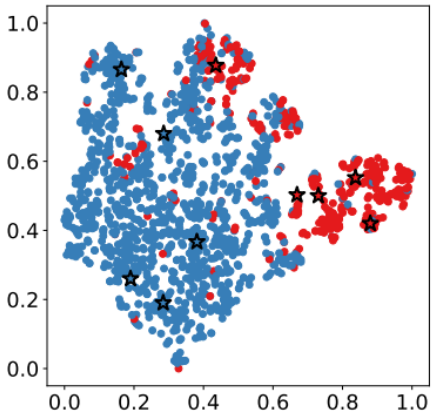


Figure 7: Trực quan hóa trên tập dữ liệu BBBP

sử dụng phương pháp t-SNE.

3.4.5 Kết luận

Tác giả đã đề xuất một phương pháp mà các mô hình có thể tự giải thích bằng cách sử dụng phương pháp học prototype. Phương pháp được đề xuất có thể cung cấp một

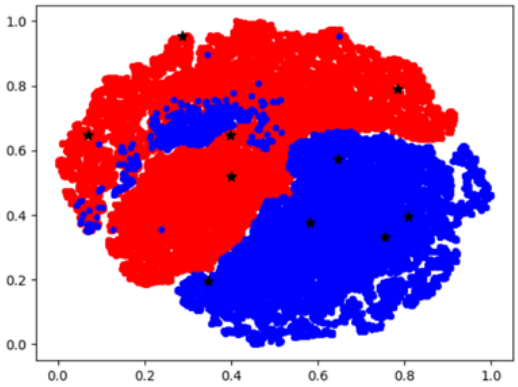


Figure 8: Trực quan hóa trên tập dữ liệu SST2 sử dụng nhúng đồ thị.

quy trình suy luận mà con người có thể hiểu được với độ chính xác phân loại và độ phức tạp về thời gian có thể chấp nhận được. Ngoài ra, công trình này góp phần phát triển ứng dụng GNN cho các lĩnh vực cần độ tin cậy và đảm bảo cao như y sinh học.

References

[1] C. Aggarwal and H. Wang. *Managing and Mining Graph Data*, volume 40. 01 2010. ISBN 978-1-4419-6044-3. doi: 10.1007/978-1-4419-6045-0.

[2] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

[3] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. volume 2, pages 49–54, 06 2014. doi: 10.3115/v1/P14-2009.

[4] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *EMNLP*, 1631:1631–1642, 01 2013.

[5] R. Vogt, S. Rahman, and C. Crespo-Hernández. *Structure–Activity Relationships in Nitro-Aromatic Compounds*, pages 217–240. 03 2009. ISBN 978-90-481-2686-6. doi: 10.1007/978-90-481-2687-3_10.

[6] Z. Wu, B. Ramsundar, E. Feinberg, J. Gomes, C. Geniesse, A. Pappu, K. Leswing, and V. Pande. Moleculenet: A benchmark for molecular machine learning. *Chemical Science*, 9, 03 2017. doi: 10.1039/C7SC02664A.

[7] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure-based approach. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 06 2004. doi: 10.1145/1007568.1007607.

[8] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. GNN explainer: A tool for post-hoc explanation of graph neural networks. *CoRR*, abs/1903.03894, 2019. URL `http://arxiv.org/abs/1903.03894`.

[9] H. Yuan, H. Yu, S. Gui, and S. Ji. Explainability in graph neural networks: A taxonomic survey. *CoRR*, abs/2012.15445, 2020. URL `https://arxiv.org/abs/2012.15445`.

[10] Z. Zhang, Q. Liu, H. Wang, C. Lu, and C. Lee. Protgnn: Towards self-explaining graph neural networks. *CoRR*, abs/2112.00911, 2021. URL `https://arxiv.org/abs/2112.00911`.