Graph Mining

Graph에서 Frequent Subgraph를 찾는 법

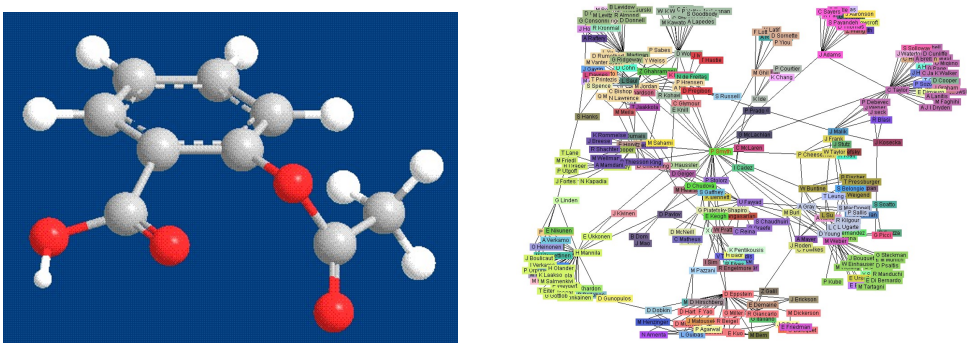
201824633

2021년 12월 3일

1 Graph Mining

Data Mining은 데이터로부터 새로운 지식을 발견해내는 과정으로, 수많은 데이터의 산에서 가치있는 정보를 찾는다. 그 후 우연이 아닌 인과관계로부터 패턴을 찾아내서 추출한다. Data Mining에는 Text Mining, Association Rule Mining, Graph Mining 등의 다양한 분야가 존재한다. 그 중 Graph Mining은 활용도가 매우 높다.

Graph Mining이란, 한 그래프에서 자주 사용되는 Subgraph들을 찾아내는 것이다. 이 때 그래프는 어느 분야에서든 다양한 형태로 존재한다. 그에 대한 예시는 그림 1과 같다.



(a) Aspirin 구조 (b) Co-Author

그림 1: 다양한 분야에서 사용되는 그래프들

위의 그림 1a은 신약 개발을 위한 기존 약의 단백질 구조, 그림 1b는 사람들간의 관계를 표시한 그래프이다.

2 Frequent Subgraph Approaches

1절에서 Graph Mining이란 그래프에서 자주 사용되는 Subgraph들을 찾아내는 것이라고 했다. Subgraph를 찾아내는 방법에 대해 알아보자.

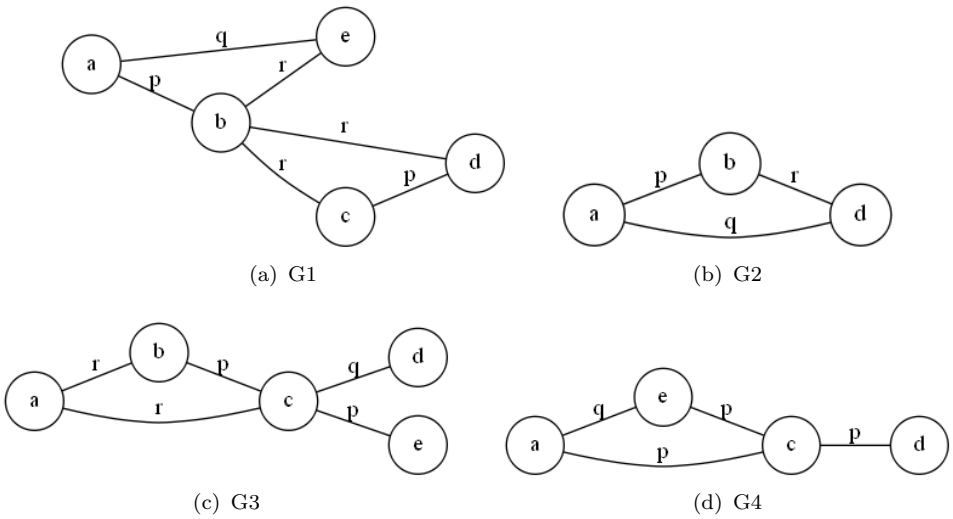
2.1 Apriori-based approach

Apriori-like Algorithm은 vertex 또는 edge를 하나씩 늘려가며 조건에 맞는 subgraph를 다음 단계로 진행시키는 방식이다. 상세 방법은 다음과 같다.

1. 1-subgraph를 찾는다.
2. Candidate Generation : (k-1)-subgraph를 결합해서 k-subgraph를 만든다. 이 k-subgraph를 candidate라고 칭한다.
3. Candidate Pruning : 2단계에서 만들어진 candidate에 infrequent (k-1)-subgraph가 포함되어 있다면 해당 candidate를 제외한다.
4. Support Counting 전체 그래프에 대해 이 subgraph가 몇개나 포함되는지 count한다.
5. Eliminate Candidate k-subgraphs : 만약 support가 threshold를 만족하지 않는 경우 해당 candidate는 Frequent Subgraph가 아니다.
6. k-subgraph(candidate)가 1개 또는 하나도 남지 않을때 까지 2단계부터 5단계 까지의 단계를 반복한다.

위의 순서를 따르는 Apriori-based Algorithm의 예시를 살펴보자.

Apriori-based Algorithm의 적용 예시



(a) G1 (b) G2

(c) G3 (d) G4

그림 2: Graph Dataset For Apriori-based Algorithm

그림 2는 Apriori-based Algorithm을 설명하기 위한 예시 그래프이다. 본격적으로 설명하기 전에 Minimum Support Count가 2임을 알린다.

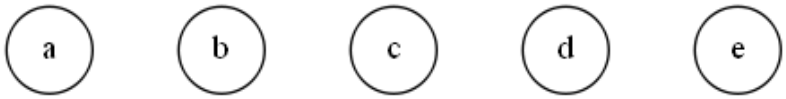
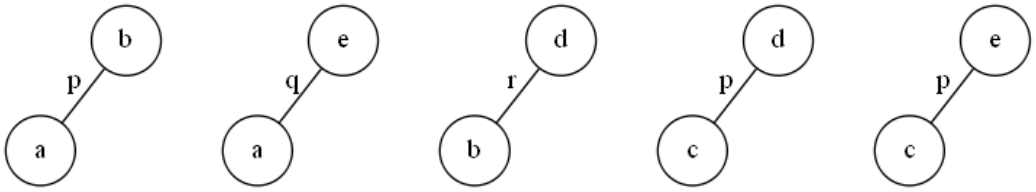


그림 3: k = 1 Frequent Subgraphs

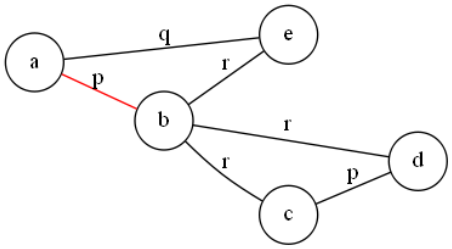
그림 3은 k가 1일때 k-subgraph들을 나타낸다. 그림 2의 그래프 G1, G2, G3, G4를 counting 하면 a는 4번, b는 3번, c는 3번, d는4번, e는 3번이다. 그러므로 a,b,c,d,e 모두 조건을 만족하므로 Frequent Subgraph이다.



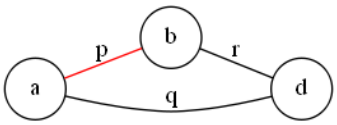
(a) (b) (c) (d) (e)

그림 4: k = 2 Frequent Subgraphs

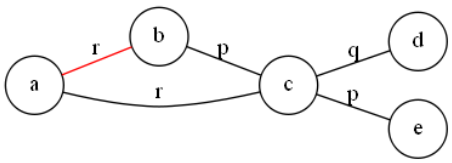
그림 4는 k가 2일때 k-subgraph들을 나타낸다. 그림 2의 그래프 G1, G2, G3, G4를 counting한 결과이다. 예를 들어 vertex a와 b로 이루어진 그래프 counting해 보자.



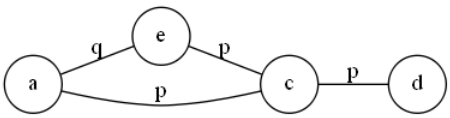
(a) G1



(b) G2



(c) G3

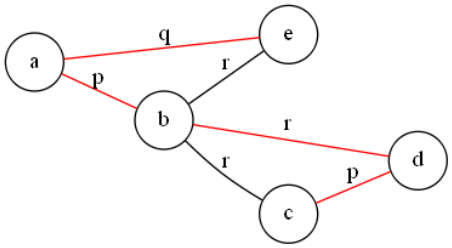


(d) G4

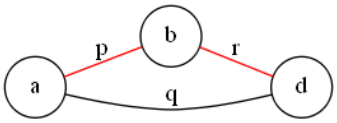
그림 5: 전체 dataset 중 vertex a와 b로 이루어진 edge 표시

1. G1 (그림 5a) : vertex a와 b사이의 weight가 p
2. G2 (그림 5b) : vertex a와 b사이의 weight가 p
3. GE (그림 5c) : vertex a와 b사이의 weight가 r

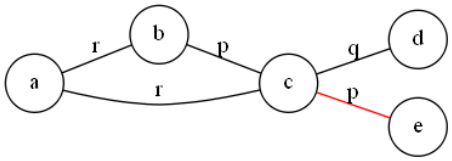
weight가 p인 것은 2개, r인 것은 1개로 p만 threshold를 만족한다. 이에 따라 vertex a와 b 사이의 weight가 p인 것(그림 4a)만이 Frequent Subgraph이다.



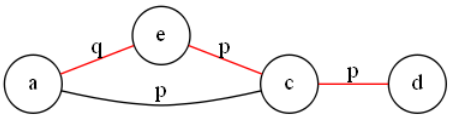
(a) G1



(b) G2



(c) G3



(d) G4

그림 6: G1, G2, G3, G4에 k가 2인 Frequent Graph(그림 4 참조)를 표시함

그림 6를 참고해 k가 3인 Frequent Subgraph를 추출하면 그림 7과 같다.

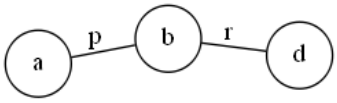


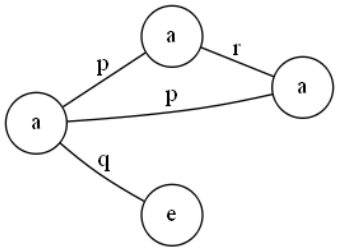
그림 7: k = 3 Frequent Subgraphs

k=3일 때, Frequent Subgraph가 1개이므로 반복을 중단한다.

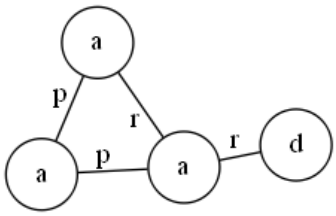
2.2 Pattern Growth approach

k개의 vertex 또는 edge를 가지는 subgraph 2개를 합쳐서 k+1개의 item들을 가지는 candidate를 만든다. 이 candidate는 threshold를 만족할 때 Frequent Subgraph가 된다.

Vertex Growing



(a) G1



(b) G2

그림 8: 임의의 Graph G에 대한 subgraph

위의 그림 8는 임의의 그래프 G의 Subgraph이다. 이 때 이 Subgraph들은 threshold를 만족한다. 그림 8a와 그림 8b을 합쳐서 만든 새로운 candidate는 아래의 그림 9와 같다.

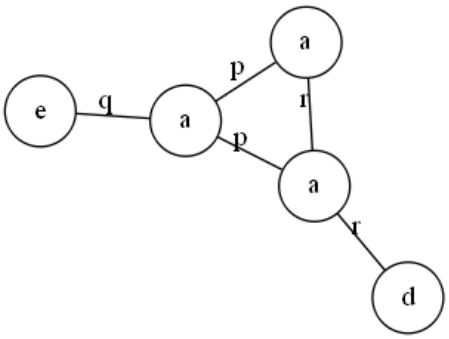


그림 9: G3 = join(G1,G2)

그림 9는 그림 8a에 vertex d, 그림 8b에 vertex e를 추가한 것과 같다. 이 candidate가 threshold를 만족하면 Frequent Subgraph가 된다.

그림 8a, 그림 8b, 그림 9을 행렬으로 나타내보자.

$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \tag{1}$$

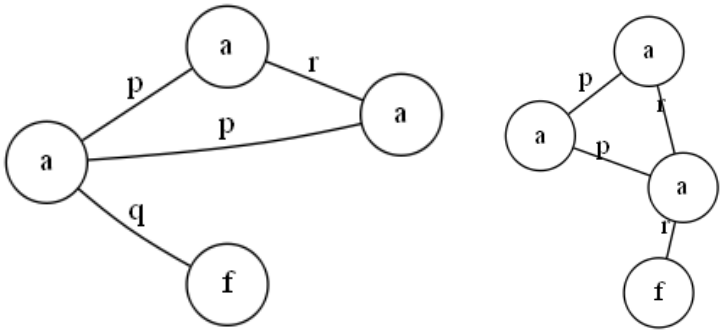
$$M_{G2} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \tag{2}$$

식 (1)은 그림 8a, 식 (2)은 그림 8b에 각각 대응한다. 그림 8a와 그림 8b를 병합한 그림 9에 대응하는 행렬은 아래와 같다.

$$M_{G3} = \begin{pmatrix} 0 & p & p & 0 & q \\ p & 0 & r & 0 & 0 \\ p & r & 0 & r & 0 \\ 0 & 0 & r & 0 & 0 \\ q & 0 & 0 & 0 & 0 \end{pmatrix}$$

Edge Growing

Edge Growing 또한 Vertex Growing과 유사하다. Vertex를 늘려가는 것이 아닌, Edge를 늘려가는 것 외에는 차이점이 없다.



(a) G1 (b) G2

그림 10: 임의의 Graph G에 대한 subgraph

위의 그림 10은 임의의 그래프 G에 대한 subgraph G1, G2이다. 이 때 이 Subgraph들은 threshold를 만족한다. 이 두개의 subgraph를 병합해 만든 candidate는 그림 11과 같다. 이 candidate가 threshold를 만족하면 Frequent Subgraph가 된다.

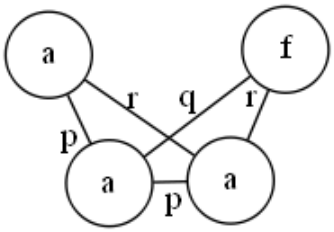


그림 11: G3 = join(G1,G2)

3 Result

Frequent Subgraph를 찾는 2가지 방법을 살펴보았다. 두 방법 모두 좋은 방법이지만 단점도 존재한다. Apriori-based와 같은 경우 n개의 edge가 존재한다고 가정했을때, 2^n개의 Subgraph가 발생한다. 이러한 단점을 극복한 방법을 적절히 사용한다면, 효율적으로 Frequent Subgraph를 찾을 수 있을것이다.