Efficient Combined Approach for Frequent Subgraph mining

By Md Mahamudur Rahaman Sajib

Outline

- Introduction
- Graph reminders
- Depth First Search (DFS) codes and tree
- Algorithm Specification

Introduction

- Extending APriori algorithms for itemsets and sequences to graphs

- Formulate new labeling method for easier graph testing that allows sorting of all graphs: DFS canonical label

- Use depth first search on hierarchical structure for faster performance, instead of breadth first search as in standard apriori algorithms

Graph basics

- It works on labeled simple graphs
- Labeled graph $G = (V, E, L, l)$

  - V set of vertices

  - $E \subseteq V \times V$ set of edges

  - L set of labels

  -$l : (V \cup E) \rightarrow L$ labeling of vertices and edges

Graph basics

- Definition: An isomorphism is a bijective function

  -$f : V(G) \rightarrow V(H)$, $(f(u), f(v)) \in E(H)$

Goal

- Given dataset of graphs $GS=\{Gi \mid i=1..n\}$ and minimum support value, define

$\xi(g, G) = \{ 1$ if g is isomorphic, 0 otherwise $\}$

$\sigma(g, GS) = \sum\xi(g, Gi) \rightarrow$ frequency of graph g in GS

- Frequent Subgraph Mining: – find graphs g in GS such that their frequency is greater of equal to minimum support

Idea Outline

- Instead of searching graphs and testing for isomorphism we construct canonical DFS codes

- Each graph has a canonical DFS code and the codes are equivalent if the graphs are isomorphic

- The codes are based on DFS trees

DFS tree

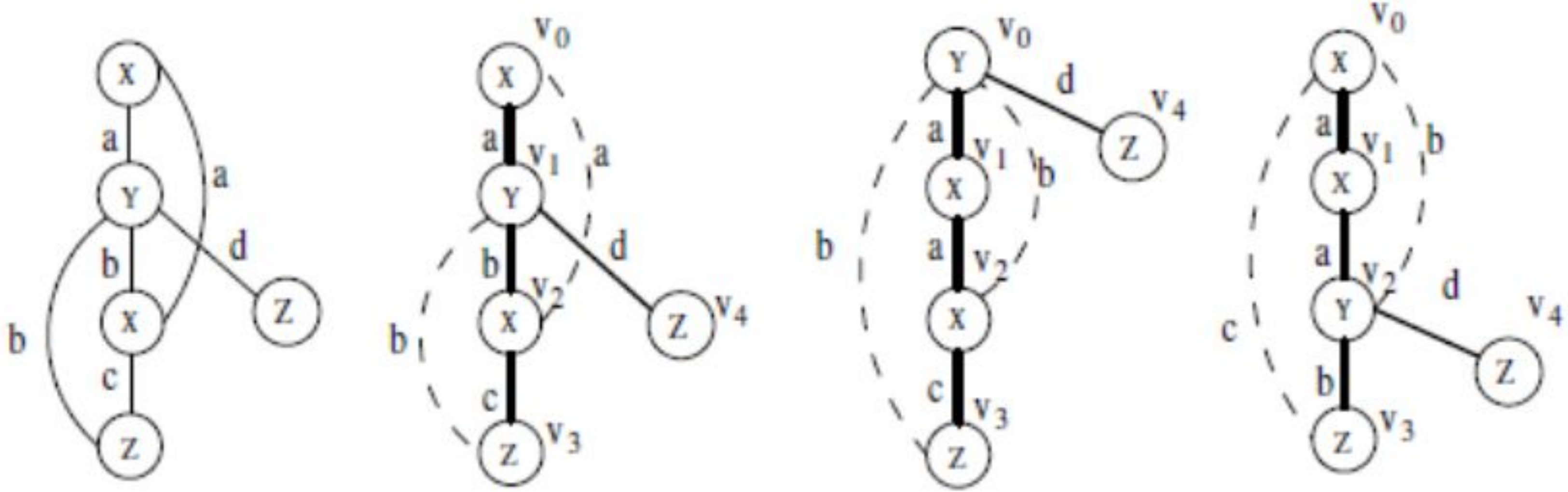
- Mark vertices in the the order they are traversed $v_i < v_j$ if v_i is traversed before v_j this constructs a DFS tree T, denoted GT

- DFS induces a linear order on vertices

- DFS divides edges in two sets – forward edge set: (v_i , v_j) where $v_i < v_j$ – backward edge set: (v_i , v_j) where $v_i > v_j$

- There are huge number of DFS trees for single graph

DFS tree


X
a
a
Y
d
b
X
Z
b
c
Z
v0
X
a
v1
a
Y
d
b
v2
X
Z
v4
b
c
Z
v3
v0
Y
d
v4
a
v1
b
Z
X
b
a
v2
X
c
v3
Z
v0
X
a
v1
b
X
a
v2
d
Y
v4
c
Z
b
Z
v3

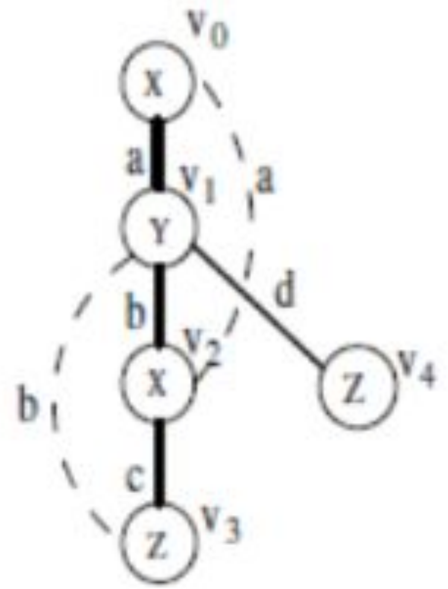
Linear orders

- A linear order of vertices defines a linear order ofedges

  $(u, v) < (u, w)$ if $v < w$ and $(u, v) < (v, w)$ if $u < v$, $e1 < e2$ and $e2 < e3$ implies $e1 < e3$

- Linear order of edges is DFS code

Linear orders



{ (v0 ,v1), (v1 ,v2), (v2 ,v0), (v2,v3), (v3 ,v1), (v1 ,v4) } is the linear order of edges for this dfs tree

DFS codes

- DFS code is a sequence of 4-tuples containing an edge and three labels
- Assume that there is an order on the labels
- This order together with the edge order defines an order for any two 4-tuples
- This extends to DFS code using a lexicographic encoding

DFS codes

- DFS code can be expanded with vertex and edge labels

Minimum DFS code

- Let the canonical DFS code to be the lexicographically smallest code that can be constructed from G (denoted min(G))

- Theorem: Given two graphs G and H, they are isomorphic if and only if min(G)=min(H)

- Subgraph mining: – Mining frequent subgraphs is equivalent mining their corresponding minimum DFS codes – Can be done sequentially by pattern mining algorithms

DFS Code Tree

- Definition: DFS code's parent and child

  A = (a0, a1, a2,...aM)

  B = (a0, a1, a2,...aM, b)

  A is B's parent and B is child of A

- DFS Code Tree: – each node represents DFS code – relations between parents and children complies with previous definition – siblings are consistent with DFS lexicographic order

DFS Code Tree

- Theorem (frequency anti monotone): If a graph G is frequent, then any subgraph of G is frequent. If G is not frequent, then any graph which contains G is not frequent.

OR

- If a DFS code α is frequent, then every ancestor of α is frequent. If α is not frequent then every descendant of α is not frequent.

DFS Code Tree

- Some graphs can have more DFS nodes corresponding to it in DFS Code Tree
- The first occurence is the minimum DFS code
- Theorem: If DFS code is not the minimum one, we can prune the entire subtree below this node, and still preserve DFS Code Tree Covering
- Pre-order searching of DFS Code Tree guarantees that we can enumerate all potential frequent subgraphs

Algorithm

```
GraphSet_projection(GS,FS)
    sort labels of the vertices and edges in GS by frequency;
    remove infrequent vertices and edges;
    relabel the remaining vertices and edges (descending);
    S1 := all frequent 1-edge graphs;
    sort S1 in DFS lexicographic order;
    FS := S1;
    for each edge e in S1
        do init g with e
        set Subgraph_mining(GS,FS,g);
        GS := GS - e;
        if |GS| < minSup break;
```

Algorithm

```
Subgraph_mining(GS,FS,g)
    if g ≠ min(g) return;
    FS := FS U {g};
    enumerate g in each graph in GS and count g's
children;
    for each c (child of g) do
    if support(c) ≥ minSup
        Subgraph_mining(GS,FS,c);
```