Popular algorithms in machine learning and data mining

gSpan
Graph Substructure Pattern Mining

Dušan Sovilj

16.04.2008.

Outline

- Introduction
- Graph reminders
- Depth First Search (DFS) codes and tree
- gSpan algorithm

Introduction

- Extending APriori algorithms for itemsets and sequences to graphs
  - candidate generation is costly
  - kernel of subgraph mining - *isomophism test*
  - costly subgraph isomorphism test (NP-complete)

Introduction

- Formulate new labeling method for easier graph testing that allows sorting of all graphs: *DFS canonical label*
- Use depth first search on hierarchial structure for faster performace, instead of breath first search as in standard apriori algorithms

Graph basics

- gSpan works on labeled simple graphs
- Labeled graph $G = (V, E, L, \ell)$

  V set of vertices
  $E \subseteq V \times V$ set of edges
  L set of labels
  $\ell : V \cup E \to L$ labeling of vertices and edges

Graph basics

- Definition: An isomorphism is a bijective function

$$
\begin{aligned}
& f : V(G) \rightarrow V(H) \\
& l_G(u) = l_H(f(u)) \quad \text{for } u \in V(G) \\
& \left.\begin{aligned} & (f(u), f(v)) \in E(H) \\ & l_G(u,v) = l_H(f(u), f(v)) \end{aligned}\right\} \quad \text{for } (u,v) \in E(G)
\end{aligned}
$$

- A subgraph isomorphism from G to H is an isomorphism from G to subgraph H

Goal

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

$$\zeta(g, G) = \begin{cases} 1 & \text{if graph } g \text{ is isomorphic to a subgraph of } G \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma(g, GS) = \sum_{G_i \in GS} \zeta(g, G_i) \;\rightarrow\; \text{frequency of graph } g \text{ in } GS$$

- 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

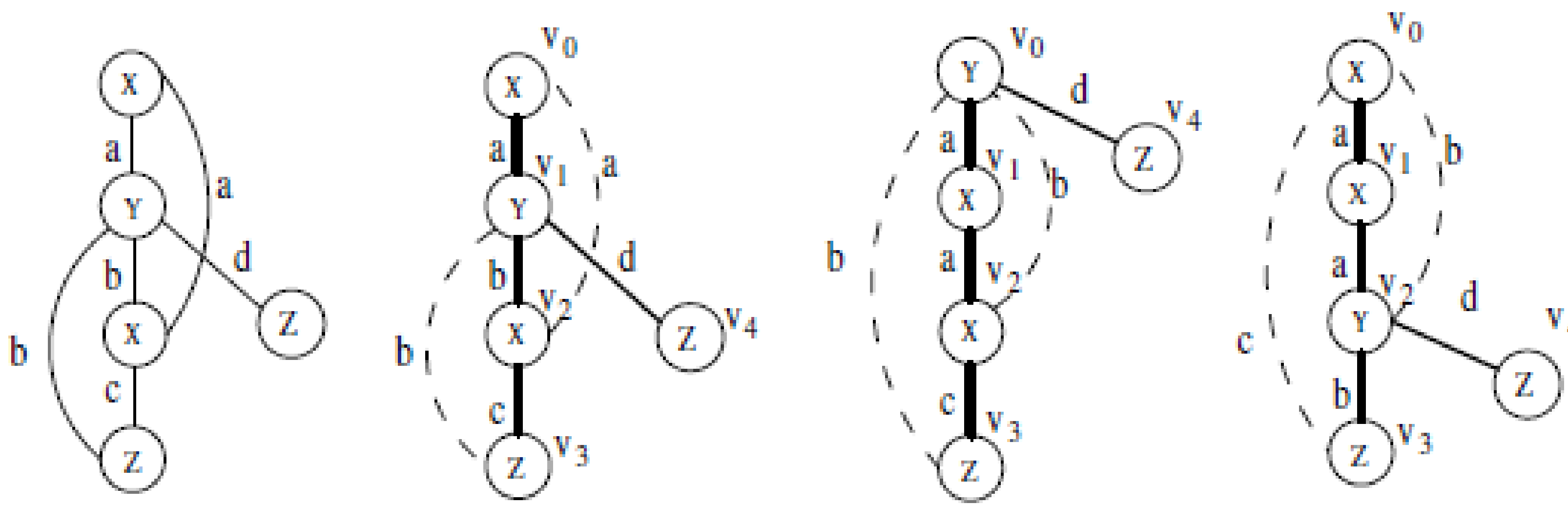
- Mark vertices in the the order they are traversed

  $$v_i < v_j \text{ if } v_i \text{ is traversed before } v_j$$

  this constructs a DFS tree T, denoted G_T
- 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

Example graph with different traversals



bold line - forward edge

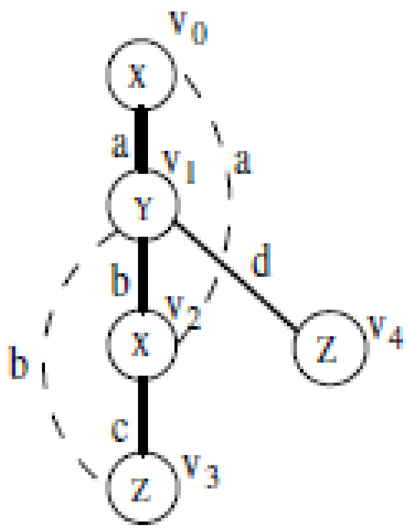
dashed line - backward edge

Linear orders

- A linear order of vertices defines a linear order of edges
  1. $(u, v) <_T (u, w)$ if $v < w$
  2. $(u, v) <_T (v, w)$ if $u < v$
  3. $e_1 <_T e_2$ and $e_2 <_T e_3$ implies $e_1 <_T e_3$
- Linear order of edges is **DFS code**

Linear orders

- Can be easily constructed



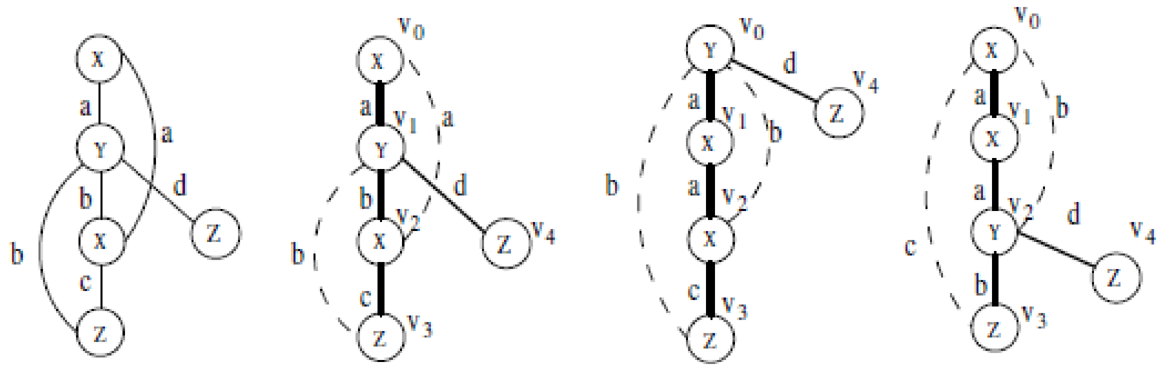
$\{ (v_0,v_1), (v_1,v_2), (v_2,v_0), (v_2,v_3), (v_3,v_1), (v_1,v_4) \}$

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



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

Minimum DFS code

- Let the canonical DFS code to be the 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 sequentally by pattern mining algorithms

DFS Code Tree

- Definition: DFS code's parent and child

  $$\alpha = (a_0, a_1, .., a_m)$$

  $$\beta = (a_0, a_1, .., a_m, \boldsymbol{b})$$

  α is β's parent and β is child of α

- 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

- Properties:
  - With label set L, DFS Code Tree contains all possible graphs for this label set
  - Each graph on the n-th level in the DFS Code Tree contrains n-1 edges
  - DFS code tree contains minimum DFS codes for all grahps (DFS Code Tree Covering)

DFS Code Tree

- Theorem (frequency antimonotone): 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

gSpan 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 g.DS = {h | h∈GS, e∈E(h)};
            Subgraph_mining(GS,FS,g);
            GS := GS - e;
            if |GS| < minSup
                break;
```

gSpan algorithm

```
Subgraph_mining(GS,FS,g)
    if g ≠ min(g)
        return;
    FS := FS ∪ {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);
```

Enumeration of *g*:

Finding all exact positions of *g* in another graph

Last words

- No candidate generation
  - frequent (k+1)-edge subgraphs are grown from frequent k-edge graphs
- Depth First Search of DFS Code Tree
  - saving space
- Beats ealier algorithms by quite a margin
- Easily extendable to other domains (sequences, trees, lattices)

References

- X. Yan and J.Han. gSpan: Graph-based substructure pattern mining. Technical report, 2002.