Fast Frequent Free Tree Mining in Graph Databases

Peixiang Zhao Jeffrey Xu Yu

The Chinese University of Hong Kong

December 18th , 2006

Synopsis

- **Introduction**
- **Existing Approaches**
- **Our Algorithm: *F3TM***
- **Performance Studies**
- **Conclusions**

Introduction

- **Graph, a general data structure to represent relations among entities, has been widely used in a broad range of areas**
  - Computational biology
  - Chemistry
  - Pattern recognition
  - Computer networks
  - etc.
- **Mining frequent sub-graphs in a graph database**
  - If a large graph contains another small graph : the sub-graph isomorphism problem (**NP-complete**)
  - If two graphs are isomorphic : the graph isomorphism problem (either P or **NP-complete**)

Introduction

- **Free Tree (*ftree*)**
  - *Connected*, *acyclic* and *undirected* graph
  - Widely used in bioinformatics, computer vision, networks, etc.
  - **Specialization** of general graph avoiding undesirable theoretical properties and algorithmic complexity incurred by graph
    - determining whether a tree t_1 is contained in another tree t_2 can be solved in $O(m^{3/2}n/\log m)$ time
    - determining whether t_1 is isomorphic to t_2 can be solved in $O(n)$
    - determining whether a tree is isomorphic to some sub-trees of a graph, a costly ***tree-in-graph*** testing which is still **NP-Complete**

Introduction

- **Frequent free tree mining**
  - Given a graph database $D = \{ g_1, g_2, \ldots, g_N \}$. The problem of frequent free tree mining is to find the set of ***all*** frequent free trees where a ftree, t, is ***frequent*** if the ratio of graphs in D, that has t as its sub-tree, is greater than or equal to a user-given threshold Φ
  - **Two key concepts**
    - **Candidate generation**
    - **Frequency counting**
- **Our focus**
  - The less number of candidates generated, the less number of times to apply costly *tree-in-graph* testing
  - the cost of candidate generation itself can be high

Existing Approaches

- **FT-Algorithm**
  - Apriori-based algorithm
  - Builds a conceptual enumeration lattice to enumerate frequent ftrees in the database
  - Follows a *pattern-join* approach to generate candidate frequent ftrees
- **FG-Algorithm**
  - A vertical mining algorithm
  - Builds an enumeration tree and traverses it in a *depth-first* fashion
  - Takes a *pattern-growth* approach to generate candidate frequent ftrees

Our Algorithm: *F3TM*

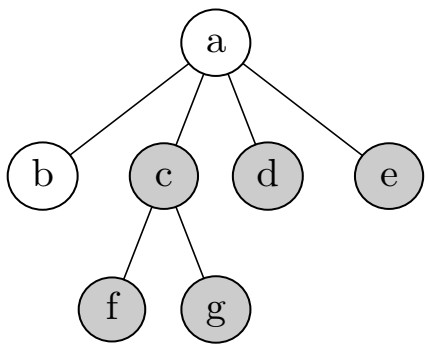
- **F3TM (Fast Frequent Free Tree Mining)**
  - A ***vertical*** mining algorithm
    - Requires a relatively small memory to maintain the frequent ftrees being found
  - Uses the ***pattern-growth*** approach for candidate generation
  - Two ***pruning*** algorithms are proposed to facilitate candidate generation and they contribute a dramatic speedup to the final performance of our ftree mining algorithm
    - Automorphism-based pruning
    - Canonical mapping-based pruning

Canonical Form of Free Tree

- **A unique representation of a ftree**
  - two ftrees, t_1 and t_2, share the same canonical form if and only if t_1 is isomorphic to t_2
- **Only free trees in their canonical form need to be considered in frequent ftree mining process**
- **A two-step algorithm**
  - normalizing a ftree to be a rooted ordered tree
  - assigning a string, as its code, to represent the normalized rooted ordered tree
  - Both steps of the algorithm are O(n), for a n-ftree

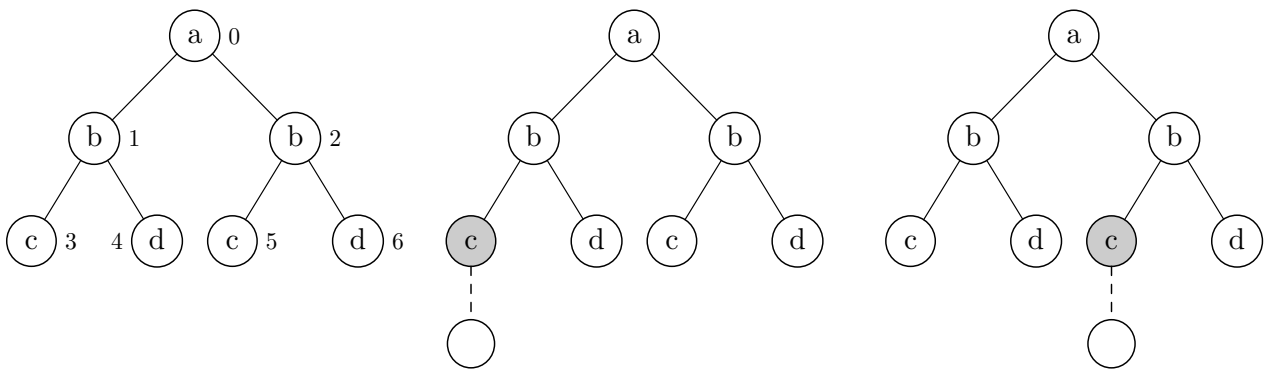
Candidate Generation

- **Theorem:** the completeness of frequent ftrees is ensured if we grow vertices from the predefined positions of a ftree, called ***extension frontier***
- Extension frontier represents all legal positions of an *n*-ftree *t'* on which a new vertex can be appended to achieve the new (*n*+*1*)-ftree *t*, while no ftrees are omitted during this *frontier-extending* process



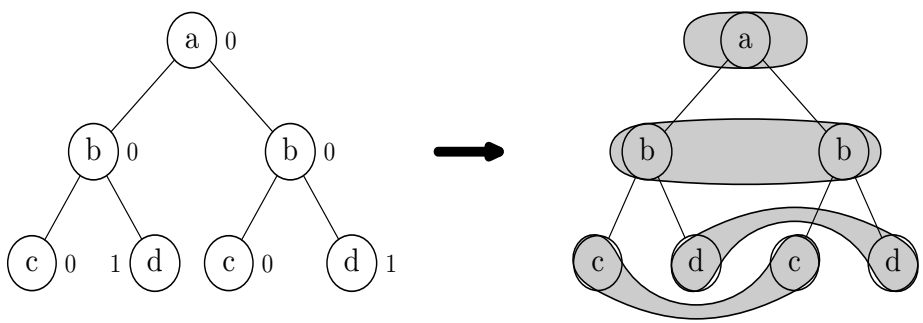
Automorphism-Based Pruning

- **Given a candidate ftree *t* in *T* (the candidates set), in order to reduce the cost of frequency counting, we firstly check if there is a candidate ftree *t′* in *T* such as *t* = *t′***
  - There is no need to count redundancies
- **When *T* becomes large, the cost of checking *t* = *t′* for every *t′* in *T* can possibly become the dominating cost**



Automorphism-Based Pruning

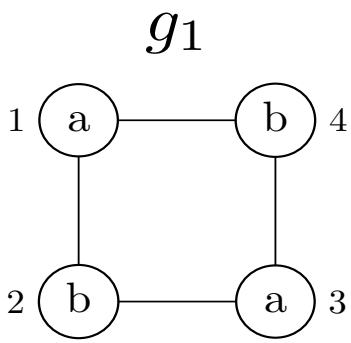
- **Automorphism-based pruning**
  - efficiently prunes redundant candidates in *T* while avoids checking if a ftree has existed in *T* already, repetitively
  - All vertices of a free tree can be ***partitioned*** into different ***equivalence classes*** base on automorphism
  - We only need to grow vertices from ***one representative*** of an equivalence class, if vertices of the equivalence class are in the extension frontier of the ftree

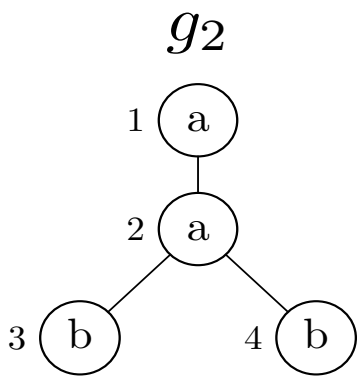


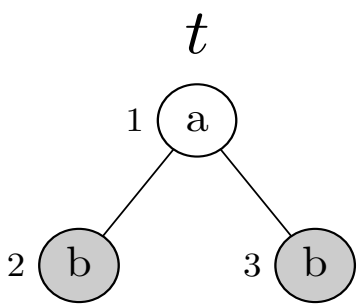
Canonical Mapping-based Pruning

- **How to select potential labels to be grown on the frequent ftrees during candidate generation?**
  - Existing algorithms maintain ***mappings*** from a ftree t to all its k occurrences in g_i
  - Based on these mappings, it is possible to know which labels, that appear in graph g_i, can be selected and assigned to generate a candidate $(n+1)$-ftree
  - there are a lot of redundant mappings between a ftree t and occurrences in g_i

Canonical Mapping-based Pruning







mapping list

(1;1,2,4)
(1;1,4,2)
(1;3,2,4)
(1;3,4,2)
(2;2,3,4)
(2;2,4,3)

Canonical Mapping-based Pruning

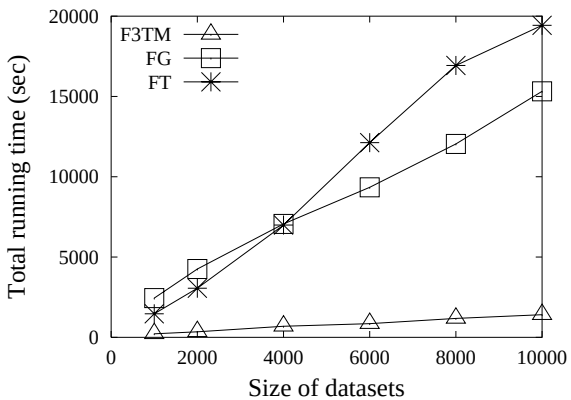
- **Canonical mapping**
  - efficiently avoid ***multiple mappings*** from a ftree to ***the same occurrence*** of the tree in a graph g_i of D
  - After orienting frequent ftree t to its canonical mapping t' of g_i in D, We can select potential labels from graph g_i for candidate generation
  - Given a n-ftree t, and assume that the number of equivalence classes of t is c, and the number of vertices in each equivalence class C_i is n_i ($1 \leq i \leq c$)
    - The number of mappings between t and an occurrence t' in graph g_i is up to $\prod_{i=1}^{c}(n_i)!$
    - With canonical mapping, we only need to consider one out of $\prod_{i=1}^{c}(n_i)!$ mappings for candidate generation

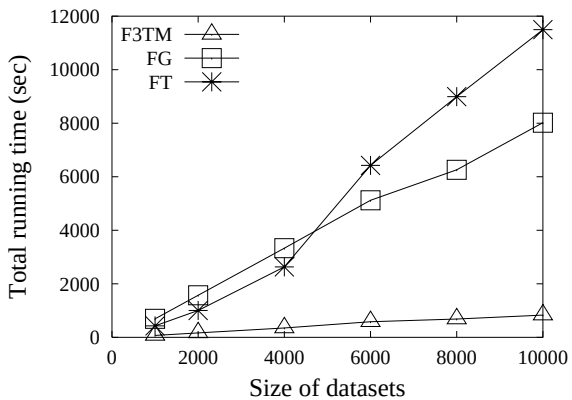
Performance Studies

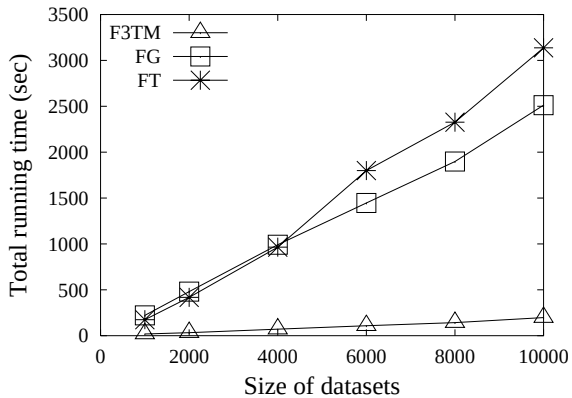
- **The Real Dataset**
  - The AIDS antiviral screen dataset from Developmental Theroapeutics Program in NCI/NIH
  - 42390 compounds retrieved from DTP's Drug Information System
  - 63 kinds of atoms in this dataset, most of which are C, H, O, S, etc.
  - Three kinds of bonds are popular in these compounds: single-bond, double-bond and aromatic-bond
  - On average, compounds in the dataset has 43 vertices and 45 edges.
  - The graph of maximum size has 221 vertices and 234 edges

Real Data Set

- **Performance comparisons (with different minimum threshold: 10%, 20%, 50%)**







Conclusion

- **Free tree** has computational advantages over general graph, which makes it a suitable candidate for computational biology, pattern recognition, computer networks, XML databases, etc.
- ***F3TM*** discovers all frequent free trees in a graph database with the focus on reducing the cost of candidate generation
  - *F3TM* outperforms the up-to-date existing free tree mining algorithms by an order of magnitude
  - *F3TM* is scalable to mine frequent free trees in a large graph dataset with a low minimum support threshold

Thank you