Mining Frequent Subgraphs

COMP 790-90 Seminar

Spring 2011

Overview

- Introduction
  - Finding recurring subgraphs from graph databases.
  - FSG
  - gSpan
  - FFSM
**Labeled Graph**

- We define a *labeled graph* $G$ as a five element tuple $G = \{V, E, \sum_V, \sum_E, \delta\}$ where
  - $V$ is the set of vertices of $G$,
  - $E \subseteq V \times V$ is a set of undirected edges of $G$,
  - $\sum_V (\sum_E)$ are set of vertex (edge) labels,
  - $\delta$ is the labeling function: $V \rightarrow \sum_V$ and $E \rightarrow \sum_E$ that maps vertices and edges to their labels.

**Frequent Subgraph Mining**

*Input*: A set $GD$ of labeled undirected graphs

*Output*: All frequent subgraphs (w. r. t. $\sigma$) from $GD$. 

:sigma: = 2/3
Finding Frequent Subgraphs

- Given a graph database \( GD = \{G_0, G_1, \ldots, G_n\} \), find all subgraphs appearing in at least \( \sigma \) graphs.
  - Isomorphic subgraphs are considered the same subgraph.

Apriori approaches

- Generation of subgraph candidates is complicated and expensive.
- Subgraph isomorphism is an NP-complete problem, so pruning is expensive.

Apriori-Based, Breadth-First Search

- Methodology: breadth-search, joining two graphs

- AGM (Inokuchi, et al. PKDD’00)
  - generates new graphs with one more node

- FSG (Kuramochi and Karypis ICDM’01)
  - generates new graphs with one more edge
**FSG Algorithm**

- $K = 1$
- $F_1 = \text{all frequent edges}$
- Repeat
  - $K = K + 1$
  - $C_K = \text{join}(F_{K-1})$
  - $F_K = \text{frequent patterns in } C_K$
- Until $F_K$ is empty

**Join: Key Operation**

- $\text{Join}(L) = \bigcup \text{join}(P, Q)$ for all $P, Q \in L$
- $\text{Join}(P, Q) = \{ G | P, Q, \subseteq G, |G| = |P| + 1, |P| = |Q| \}$
- Two graphs $P$ and $Q$ are **joinable** if the join of the two graphs produces a non-empty set
- Theorem: two graphs $P$ and $Q$ are joinable if $P \cap Q$ is a graph with size $|P| - 1$ or share a common “core” with size $P-1$
Multiplicity of Candidates

Case 1: identical vertex labels

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

3/3/2011

Core: The (k-1) subgraph that is common between the joint graphs

Multiplicity of Candidates

Case 2: Core contains identical labels

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

3/3/2011
Multiplicity of Candidates

Case 3: Core multiplicity

- Apriori-based approach
- Building blocks: edge-disjoint path
- Identify all frequent paths
- Construct frequent graphs with 2 edge-disjoint paths
- Construct graphs with \( k+1 \) edge-disjoint paths from graphs with \( k \) edge-disjoint paths
- Repeat
PATH Algorithm

- $K = 1$
- $F_1 = \text{all frequent paths}$

Repeat
- $K = K + 1$
- $C_K = \text{join}(F_{K-1})$
- $F_K = \text{frequent patterns in } C_K$
- Until $F_K$ is empty

Challenges

- **Graph isomorphism**
  - Two graphs may have the same topology though their layouts are different

- **Subgraph isomorphism**
  - How to compute the support value of a pattern
Graph Isomorphism

A graph is isomorphic if it is topologically equivalent to another graph.

Why Redundant Candidates?

All the algorithms may propose the same candidate several times.

We need to keep track of the identical candidates to

- Avoid redundancy in results
- Avoid redundant search
**gSpan**

- DFS without candidate generation
  - Relabels graph representation to support DFS.
  - Discovers all frequent subgraphs without candidate generation or pruning.

- DFS Representation
  - Map each graph to a DFS code (sequence).
  - Lexicographically order the codes.
  - Construct a search tree based on the lexicographic order.

---

**Depth-First Search Tree**

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
DFS Codes

Given \( e_1 = (i_1, j_1), e_2 = (i_2, j_2) \): \( e_1 < e_2 \) if:
- \( i_1 = i_2 \) \& \( j_1 < j_2 \)
- \( i_1 < j_1 \) \& \( j_1 = i_2 \)

\[ code(G, T) = \text{edge sequence of } e_i < e_{i+1} \]

<table>
<thead>
<tr>
<th>edge</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,1, X,a,Y)</td>
<td>(0,1, Y,a,X)</td>
<td>(0,1, X,a,X)</td>
</tr>
<tr>
<td>1</td>
<td>(1,2, Y,b,X)</td>
<td>(1,2, X,a,X)</td>
<td>(1,2, X,a,Y)</td>
</tr>
<tr>
<td>2</td>
<td>(2,0, X,a,X)</td>
<td>(2,0, X,b,Y)</td>
<td>(2,0, Y,b,X)</td>
</tr>
<tr>
<td>3</td>
<td>(2,3, X,c,Z)</td>
<td>(2,3, X,c,Z)</td>
<td>(2,3, Y,b,Z)</td>
</tr>
<tr>
<td>4</td>
<td>(3,1, Z,b,Y)</td>
<td>(3,0, Z,b,Y)</td>
<td>(3,0, Z,c,X)</td>
</tr>
<tr>
<td>5</td>
<td>(3,4, Y,d,Z)</td>
<td>(0,4, Y,d,Z)</td>
<td>(2,4, Y,d,Z)</td>
</tr>
</tbody>
</table>

DFS Lexicographic Order

- \( \partial = code(G_\partial, T_\partial) = (a_0, a_1, \ldots, a_m) \)
- \( \beta = code(G_\beta, T_\beta) = (b_0, b_1, \ldots, b_n) \)
- \( \partial \leq \beta \) iff (1) or (2):
  - (1) \( \exists t, \ 0 \leq t \leq \min(m,n), \ a_k = b_k \) for \( k < t \), \( a_t < e b_t \)
  - (2) \( a_k = b_k \) for \( 0 \leq k \leq m, \ n \geq m \)

Minimum DFS code

- The minimum DFS code \( min(G) \), in DFS lexicographic order, is the canonical label of graph G.
- Graphs A and B are isomorphic if \( min(A) = min(B) \).
DFS Codes: Parents and Children

- If $\partial = (a_0, a_1, \ldots, a_m)$ and $\beta = (a_0, a_1, \ldots, a_m, b)$:
  - $\beta$ is the child of $\partial$.
  - $\partial$ is the parent of $\beta$.
- A valid DFS code requires that $b$ grows from a vertex on the rightmost path.

DFS Code Trees

- Organize DFS code nodes as parent-child.
- Pre-order traversal follows DFS lexicographic order.
- If $s$ and $s'$ are the same graph with different DFS codes, $s'$ is not the minimum and can be pruned.
D is the set of all graphs.
S is the result set.

Algorithm 1: GraphSet Projection(D, S)
1: sort labels in D by frequency
2: remove infrequent vertices and edges
3: relabel remaining vertices and edges
4: S' = all frequent 1-edge graphs in D
5: sort S' in DFS lexicographic order
6: S = S'
7: foreach edge e in S' do
8: s = graph defined by e
9: s.D = subgraphs in D containing e
10: Subgraph_Mining(D, S, s)
11: D = D - e
12: if |D| < minSup
13: break

Subprocedure 1: Subgraph_Mining(D, S, s)
1: if s != min(s)
2: return
3: S = S ∪ {s}
4: s' = +1-edge children of s in s.D
5: foreach child c of s' do
6: if support(c) ≥ minSup
7: Subgraph_Mining(D, S, c)

Runtime: Synthetic

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
Runtime: Chemical

--- Apriori (FSG)  --- gSpan

```
\begin{figure}
\centering
\includegraphics[width=\textwidth]{runtime.png}
\caption{Runtime comparison between Apriori (FSG) and gSpan.}
\end{figure}
```

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

--- gSpan Advantages

- Lower memory requirements.
- Faster than naïve FSG by an order of magnitude.
- No candidate generation.
- Lexicographic ordering minimizes search tree.
- False positives pruning.

Any disadvantage?

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL