gSpan: Graph-Based Substructure Pattern Mining

Xifeng Yan and Jiawei Han

University of Illinois at Urbana-Champaign

February 3, 2017
Agenda

- What motivated the development of gSpan?
- Technical Preliminaries
- Exploring the gSpan algorithm
- Experimental Performance Evaluation
Introduction and Motivation

Why are we interested in mining graphs?

- Social Network Data, Electronic Transactions (Fraud Detection)
- Chemical Compounds
- Computer vision, video indexing, text retrieval

Issues with Previous Graph Mining Approaches

- Previous benchmark approaches called AGM and FGS inefficient
- Take Apriori-like, level-wise approaches
- 1) Generate huge candidate sets
- 2) Require multiple scans of database
- 3) Difficult to mine long patterns
What does gSpan have to offer?

Apriori-based algorithms suffer from:
- Costly subgraph isomorphism test
- Subgraph candidate generation complicated and expensive

gSpan combats these issues by:
- **Eliminating candidate generation!**
- Applying Depth First instead of Breadth First search
- Introducing a new lexicographic canonical labeling system
- Combining isomorphism test and subgraph growth into one procedure
Preliminaries: Formal Problem Definition

**Given:** A graph dataset \( GS = \{ G_i | i = 0, \ldots, n \} \), and \( \text{minsup} \)

- Each \( G_i \) a labeled, undirected graph \( (V, E, L, l) \)
- \( \text{minsup} \) is the minimum support

**Find:** All \( g \) s.t. \( \sum_{G_i \in GS} \zeta(g, G_i) \geq \text{minsup} \)

\[
\zeta(g, G_i) = \begin{cases} 
1 & \text{if } g \text{ is isomorphic to a subgraph of } G, \\
0 & \text{if } g \text{ not isomorphic to a subgraph of } G 
\end{cases}
\]
Are these graphs isomorphic?
Yes! An **NP**-complete Problem!
Preliminaries: DFS Lexicographic Ordering

In class we have seen ordering used during itemset mining:

- Items in "baskets" lexicographically ordered A, B, ..., F
- **Systematically** search for frequent itemsets
- Eliminates redundancy, improving efficiency

gSpan applies the same concept!!

- Each graph has a unique, ordered label
- Efficiently discover frequent subgraphs
- Prune the search space
Preliminaries: DFS Lexicographic Ordering

Graph 1

Depth-First Search Trees

Graph 1a

Graph 1b

Graph 1c

Xifeng Yan and Jiawei Han

gSpan: Graph-Based Substructure Pattern Mining
Preliminaries: DFS Lexicographic Ordering

Let \( e_1 = (i_1, j_1), e_2 = (i_2, j_2) \)

1) \( e_1 \prec_T e_2 \) if \( i_1 = i_2 \) and \( j_1 < j_2 \)

2) \( e_1 \prec_T e_2 \) if \( i_1 < i_2 \) and \( j_1 = j_2 \)
Preliminaries: DFS Lexicographic Ordering

Graph 1a

Graph 1b

Graph 1c

<table>
<thead>
<tr>
<th>Edge</th>
<th>Code 1a</th>
<th>Code 1b</th>
<th>Code 1c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,1,B,b,R)</td>
<td>(0,1,R,a,B)</td>
<td>(0,1,B,a,B)</td>
</tr>
<tr>
<td>1</td>
<td>(1,2,R,b,B)</td>
<td>(1,2,B,a,B)</td>
<td>(1,2,B,a,R)</td>
</tr>
<tr>
<td>2</td>
<td>(2,0,B,a,B)</td>
<td>(2,0,B,b,R)</td>
<td>(2,0,R,b,B)</td>
</tr>
<tr>
<td>3</td>
<td>(2,3,B,c,G)</td>
<td>(2,3,B,c,G)</td>
<td>(2,3,R,d,G)</td>
</tr>
<tr>
<td>4</td>
<td>(3,1,G,d,R)</td>
<td>(3,0,G,b,R)</td>
<td>(3,0,G,c,B)</td>
</tr>
<tr>
<td>5</td>
<td>(1,4,R,d,G)</td>
<td>(0,4,R,d,G)</td>
<td>(2,4,R,d,G)</td>
</tr>
</tbody>
</table>
We have several DFS codes for the same graph!

- Fortunately, an ordering on DFS codes too!
- Specified by DFS Lexicographic Order
- In our case, Graph 1c $\prec$ Graph 1b $\prec$ Graph 1a

Graph 1c is the **Minimum DFS Code** of Graph 1

- This is the unique, canonical label of Graph 1

**Theorem 1**

Given two graphs $G$ and $G'$, $G$ is isomorphic to $G'$ if and only if $\min(G) = \min(G')$.  

Xifeng Yan and Jiawei Han | gSpan: Graph-Based Substructure Pattern Mining
Preliminaries: DFS Lexicographic Ordering

(i) for some \( t, 0 \leq t \leq \min\{m, n\} \), we have \( a_k = b_k \) for \( k < t \), and

\[
\begin{align*}
    a_t < b_t = \ & \begin{cases} 
    \text{true if } a_t \in E_{\alpha, b} \text{ and } b_t \in E_{\beta, f}. \\
    \text{true if } a_t \in E_{\alpha, b}, b_t \in E_{\beta, b}, \text{ and } j_a < j_b. \\
    \text{true if } a_t \in E_{\alpha, b}, b_t \in E_{\beta, b}, j_a = j_b, \text{ and } l_{(i_a, j_a)} < l_{(i_b, j_b)}. \\
    \text{true if } a_t \in E_{\alpha, f}, b_t \in E_{\beta, f}, \text{ and } i_b < i_a. \\
    \text{true if } a_t \in E_{\alpha, f}, b_t \in E_{\beta, f}, i_a = i_b, \text{ and } l_{i_a} < l_{i_b}. \\
    \text{true if } a_t \in E_{\alpha, f}, b_t \in E_{\beta, f}, i_a = i_b, l_{i_a} = l_{i_b}, \text{ and } l_{(i_a, j_a)} < l_{(i_b, j_b)}. \\
    \text{true if } a_t \in E_{\alpha, f}, b_t \in E_{\beta, f}, i_a = i_b, l_{i_a} = l_{i_b}, l_{(i_a, j_a)} = l_{(i_b, j_b)}, \text{ and } j_a < j_b. 
\end{cases}
\end{align*}
\]

(ii) \( a_k = b_k \) for \( 0 \leq k \leq m \), and \( n \geq m \).

Determining the Minimum DFS Code Lexicographically
Preliminaries: DFS Lexicographic Ordering

DFS Code Tree: The "search space" of gSpan algorithm
"Growing" a tree in DFS Lexicographic Order during search
DFS Code Tree Covering

- The DFS Code Tree contains minimum DFS codes for all graphs

Frequency Antimonotone

- If graph $G$ is frequent, any subgraph of $G$ is frequent
- If graph $G$ is infrequent, any graph containing $G$ is not frequent

DFS Code Pruning

- Let the DFS Codes of $G$ be $\alpha_0, \alpha_1, ..., \alpha_n$ with $\alpha_0$ being $\min(G)$. Pruning $\{\alpha_i : 1 \leq i \leq n\}$ preserves DFS Code Tree Covering.
Algorithm 2 GraphSetProjection(GS,S).
1: sort labels of the vertices and edges in GS by their frequency;
2: remove infrequent vertices and edges;
3: relabel the remaining vertices and edges in descending frequency;
4: $S_1 \leftarrow$ all frequent 1-edge graphs in GS;
5: sort $S_1$ in DFS lexicographic order;
6: $S \leftarrow S_1$;
7: for each edge $e \in S_1$ do
8: initialize $s$ with $e$, set $s.GS = \{g \mid \forall g \in GS, e \in E(g)\}$; (only graph ID is recorded)
9: Subgraph_Mining(GS, $S$, $s$);
10: $GS \leftarrow GS - e$;
11: if $|GS| < minSup$;
12: break;
**Steps 1-2**: Remove infrequent vertices and edges
- Begin pruning dataset early!
- Clearly if edge \( e \) or vertex \( v \) are \( \leq \minsup \), any graph \( G \) containing \( e \) or \( v \) is guaranteed to be infrequent!

**Step 3**: Relabel edges and vertices in descending order
- Need an ordering on labels to determine DFS Lexicographic Order on DFS Codes
- \( V = \{ \text{Red, Green, Blue} \}, \ E = \{ X, Y, Z \} \)
Steps 4-6: Sort frequent 1-edge graphs in DFS Lexicographic order
Steps 7-9: Loop over frequent 1-edge subgraphs
- Begin with least edge in lexicographic order
- Set $s.GS = G \in GS$ where $e \in E(G)$
- Call $SubgraphMining(GS, S, s)$;
Subprocedure 1 Subgraph_Mining(GS, S, s).

1: if $s \neq \text{min}(s)$
2: return;
3: $S \leftarrow S \cup \{s\};$
4: generate all $s'$ potential children with one edge growth;$^\dagger$
5: Enumerate($s$);
6: for each $c$, $c$ is $s'$ child do
7: if $support(c) \geq \text{minSup}$
8: $s \leftarrow c$;
9: Subgraph_Mining(GS, S, s);
The gSpan Algorithm

Xifeng Yan and Jiawei Han

gSpan: Graph-Based Substructure Pattern Mining
The gSpan Algorithm

Xifeng Yan and Jiawei Han

gSpan: Graph-Based Substructure Pattern Mining
Step 11: $\mathcal{D} \leftarrow \mathcal{D} - e$

- Very important step!
- Because of Lexicographic DFS, have already mined ALL subgraphs containing edge $e$
- Remove all instances of edge $e$ from graph dataset
- Mining process takes less time for later iterations
- Drastically improves efficiency!
Tested on synthetic and real (Chemical Compounds) datasets

- Tested against previous mining benchmark FSG algorithm
- Performs 6-45x faster on synthetic data
- Performs 15-100x better on chemical compound data
Experimental Performance Results

Xifeng Yan and Jiawei Han

gSpan: Graph-Based Substructure Pattern Mining
Experimental Performance Results

![Graph-based substructure pattern mining](image)
gSpan sets new benchmark for mining frequent graphs

- No candidate generation! Reduce expensive false candidate testing
- Introduce DFS Lexicographic order to systematically SEARCH and PRUNE search space
- Depth-first as opposed to Breadth-first search
- Possible can fit in main memory, reduce IOs
- Shrink graph dataset with each iteration!