Mining Scientific Data Sets: Challenges and Opportunities

George Karypis
Department of Computer Science & Engineering
University of Minnesota

(In collaboration with Kuramochi Michihiro)
Outline

- Scientific Data Mining
  - Opportunities & Challenges
- Graph Mining
- Pattern Discovery in Graphs
- FSG & GFSG Algorithm
- Going Forward
Data Mining In Scientific Domain

- Data mining has emerged as a critical tool for knowledge discovery in large data sets.
  - It has been extensively used to analyze business, financial, and textual data sets.
- The success of these techniques has renewed interest in applying them to various scientific and engineering fields.
  - Astronomy
  - Life sciences
  - Ecosystem modeling
  - Fluid dynamics
  - Structural mechanics
Challenges in Scientific Data Mining

- Most of existing data mining algorithms assume that the data is represented via
  - Transactions (set of items)
  - Sequence of items or events
  - Multi-dimensional vectors
  - Time series
- Scientific datasets with structures, layers, hierarchy, geometry, and arbitrary relations can not be accurately modeled using such frameworks.
  - e.g., Numerical simulations, 3D protein structures, chemical compounds, etc.

Need algorithms that operate on scientific datasets in their native representation
How to Model Scientific Datasets?

- There are two basic choices
  - Treat each dataset/application differently and develop custom representations/algorithms.
  - Employ a new way of modeling such datasets and develop algorithms that span across different applications!

- What should be the properties of this general modeling framework?
  - Abstract compared with the original raw data.
  - Yet powerful enough to capture the important characteristics.

Labeled directed/undirected topological/geometric graphs and hypergraphs
Graphs can accurately model and represent scientific data sets.

Graphs are suitable for capturing arbitrary relations between the various elements.

Provide enormous flexibility for modeling the underlying data as they allow the modeler to decide on what the elements should be and what type of relations to be modeled.
Example: Protein 3D Structure

PDB; 1MWP
N-Terminal Domain Of The Amyloid Precursor Protein
Alzheimer's disease amyloid A4 protein precursor
Example: Fluid Dynamics

- Vertices ↔ Vortices
- Edges ↔ Proximity
Graph Mining

- Goal: to develop algorithms to mine and analyze graph data sets.
  - Finding patterns in these graphs
  - Finding groups of similar graphs
  - Building predictive models for the graphs

- Applications
  - Structural motif discovery
  - Toxicology prediction
  - Protein fold recognition
  - A lot more ...
Finding Interesting Patterns

- A pattern is a relation between the object’s elements that is **recurring** over and over again.
  - Common structures in a family of chemical compounds or proteins.
  - Similar vortices arrangements found in numerical simulations of turbulent fluid flows.
  - ...

- There are many methods to measure the **interestingness** of a pattern.
  - Occurrence frequency
    - The support (s) of a pattern
  - Its predictive ability
    - Rules
  - Its novelty
  - Its complexity

Finding the frequently occurring patterns is required for most of these measures of interestingness.
Finding Frequently Occurring Patterns in Graphs

Develop computationally efficient algorithms for finding frequently occurring subgraphs in large graph datasets.
Finding Frequent Subgraphs: Input and Output

- Problem setting: similar to finding frequent itemsets for association rule discovery

- Input
  - Database of graph transactions
  - Undirected simple graph (no loops, no multiples edges)
  - Each graph transaction has labeled edges/vertices.
  - Transactions may not be connected
  - Minimum support threshold \( s \)

- Output
  - Frequent subgraphs that satisfy the support constraint
  - Each frequent subgraph is connected.
Finding Frequent Subgraphs: Input and Output

Input: Graph Transactions
Output: Frequent Connected Subgraphs

Support = 100%
Support = 66%
Support = 66%
Support = 66%
Methods for Discovering Frequent Patterns

Discovering frequent patterns is one of the key operations in data mining!

- The problem is NP as there can be an exponential number of patterns.
  - Minimum support constraint is the key to limiting this complexity!
    - Downward closure property.

There are numerous approaches for solving this problem.

- Level-by-level approaches
  - Start with finding all patterns with one element, then all patterns with two elements, etc.
    - Candidate generation—candidate counting approach
      - Apriori

- Database projection approaches
  - All frequent patterns involving a particular element are discovered first before moving to the next.
    - Database is shrunk as the complexity of the pattern increases
      - Tree-projection, FP-growth, LPMiner
FSG
Frequent Subgraph Discovery Algorithm

- Level-by-level approach Incremental on the number of edges of the frequent subgraphs.
- Counting of frequent single and double edge subgraphs
- For finding frequent size $k$-subgraphs ($k = 3$),
  - Candidate generation
    - Joining two size $(k - 1)$-subgraphs similar to each other.
  - Candidate pruning by downward closure property
  - Frequency counting
    - Check if a subgraph is contained in a transaction.
  - Repeat the steps for $k = k + 1$
    - Increase the size of subgraphs by one edge.
FSG: Algorithm

Single edges

Double edges

3-candidates

3-frequent subgraphs

4-candidates

4-frequent subgraphs
So what is the big deal!
Trivial Operations Become **VERY** Complicated & Expensive on Graphs
Candidate generation
- To determine two candidates for joining, we need to perform subgraph isomorphism.
- Isomorphism for redundancy check

Candidate pruning
- To check downward closure property, we need subgraph isomorphism.

Frequency counting
- Subgraph isomorphism for checking containment of a frequent subgraph

Key to computational efficiency:
- How to reduce the number of graph/subgraph isomorphism operations?
FSG Approach: Candidate Generation

- Generate a size $k$-subgraph by merging two size $(k - 1)$-subgraphs
FSG Approach:
Candidate Generation

- Intersection of the parent lists of two 3-frequent subgraphs
- Without subgraph isomorphism, we can detect the core of the two 3-frequent subgraphs.
- Redundancy check by canonical labeling
Candidate Generation Based On Core Detection

Multiple candidates for the same core!
Candidate Generation Based On Core Detection

First Core

Second Core

Multiple cores between two \((k-1)\)-subgraphs
FSG Approach: Candidate Pruning

- Downward closure property
- Every \((k - 1)\)-subgraph must be frequent
- Keep the list of those \((k - 1)\)-subgraphs
FSG Approach: Candidate Pruning

Pruning of size $k$-candidates

- For all the $(k - 1)$-subgraphs of a size $k$-candidate, check if downward closure property holds.
  - Canonical labeling is used to speedup the computation.
- Build the parent list of $(k - 1)$-frequent subgraphs for the $k$-candidate.
  - Used later in the candidate generation, if this candidate survives the frequency counting check.
FSG Approach: Frequency Counting

Transactions

\[ T1 = \{ f1, f2, f3 \} \]
\[ T2 = \{ f1 \} \]
\[ T3 = \{ f2 \} \]

Frequent Subgraphs

\[ \text{TID}(f1) = \{ T1, T2 \} \]
\[ \text{TID}(f2) = \{ T1, T3 \} \]

Candidate

\[ c = \text{join}(f1, f2) \]
\[ \text{TID}(c) = \text{subset}(\text{TID}(f1) \text{ AND } \text{TID}(f2)) \]

- Perform only \text{subgraph}_\text{isomorph}(c, T1)
- \text{TID} lists trade memory for performance.
FSG Approach: Frequency Counting

- Keep track of the TID lists.
- If a size $k$-candidate is contained in a transaction, all the size $(k - 1)$-parents must be contained in the same transaction.
- Perform subgraph isomorphism only on the intersection of the TID lists of the parent frequent subgraphs of size $k - 1$.
  - Significantly reduces the number of subgraph isomorphism checks.
  - Trade-off between running time and memory
Key to Performance: Canonical Labeling

- Identity determination of graphs
- To give a unique code to a given graph.
  - Its complexity is proven to be equivalent to graph isomorphism
  - No known polynomial algorithm.
- Given a graph, we want to find a unique order of vertices, by permuting rows and columns of its adjacency matrix.
Canonical Labeling

Find the vertex order so that the matrix becomes lexicographically the largest when we compare in the column-wise way.

\[
\begin{bmatrix}
0 & 1 & 2 \\
0 & e & \\
1 & e & f \\
2 & f & \\
\end{bmatrix}
\]

Code = “e0f”

\[
\begin{bmatrix}
1 & 2 & 0 \\
1 & f & e \\
2 & f & \\
0 & e & \\
\end{bmatrix}
\]

Code = “fe0”
Canonical Labeling

- Partitioning drastically reduces the steps
  \[ N! = ?(p_i!) \text{ where } N = ?p_i \]

- How to get finer partitions (smaller \( p_i \))
  - By vertex degrees and labels
  - By ordering partitions
  - By adjacent vertex/edge labels
  - By iteratively applying these partitioning operations
Canonical Labeling: Optimization Effect

![Graph showing running time vs. minimum support percentage]
## Canonical Labeling: Optimization Effect

<table>
<thead>
<tr>
<th>s [%]</th>
<th>Running Time [seconds]</th>
<th>Largest Pattern Size</th>
<th># Frequent Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Degree</td>
<td>Partition Ordering</td>
<td>Neighbor Labels</td>
</tr>
<tr>
<td>10.0</td>
<td>17</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>8.0</td>
<td>64</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>6.0</td>
<td>5293</td>
<td>160</td>
<td>38</td>
</tr>
<tr>
<td>5.0</td>
<td>69498</td>
<td>2512</td>
<td>93</td>
</tr>
<tr>
<td>4.0</td>
<td>---</td>
<td>35322</td>
<td>330</td>
</tr>
<tr>
<td>3.0</td>
<td>---</td>
<td>---</td>
<td>8848</td>
</tr>
<tr>
<td>2.0</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Empirical Evaluation

- Synthetic datasets
  - Sensitivity study
    - Number of labels (N) used in input graphs
    - Transaction size (T)
    - Number of transactions (D)
- Real dataset
  - Chemical compounds from PTE challenge
- Pentium III 650 MHz, 2GB RAM
Evaluation: Synthetic Datasets

- Try to mimic the idea of the data generator for frequent itemset discovery, used in the Apriori paper (Agrawal and Srikant, VLDB, 1994).
- Generate a pool of potential frequent subgraphs (“seeds”).
- Embed randomly selected seeds into each transaction until the transaction reaches the specified size.
Sensitivity: Number of Vertex Labels

- 10000 transactions
- 2% support
- Average seed size $I = 5$
- Average transaction size $T = 40$

More labels $\rightarrow$ Faster execution
**Sensitivity: Transaction Size**

- 10,000 transactions
- 2% support
- Average seed size \( I = 5 \)

Average transaction size \( T \) significantly affects the execution time, especially with fewer labels.
Sensitivity: Number of Transactions

- 2% support
- Transaction size $T = 5, 10, 20, 40$
- Average seed size $I = 5$
- 10 edge/vertex labels
- Linear scalability
Evaluation:

Chemical Compound Dataset

- Predictive Toxicology Evaluation (PTE) Challenge (Srinivasan et al., IJCAI, 1997)
- 340 chemical compounds
- Sparse
  - Average transaction size 27.4 edges, 27.0 vertices
  - Maximum transaction has 214 edges.
- 4 edge labels, 66 vertex labels
Evaluation: Chemical Compound Dataset
FSG Summary

- Linear scalability w. r. t. # of transactions
- FSG runs faster as the number of distinct edge/vertex labels increases.
- Average size of transactions $|T|$
  - Significant impact on the running time
  - Subgraph isomorphism for frequency counting
  - Edge density
    - Increases the search space of graph/subgraph isomorphism exponentially.
- Suitable for sparse graph transactions
Topology Is Not Enough
(Sometimes)

- 100 chemical compounds with 30 atoms
- Support = 10%
- 3 patterns of 14 edges found

3-(3,5-dibromo-4-hydroxyphenyl)-2-(4-iodophenyl)acrylic acid

1-methoxy-4-(2-phenylvinyl)benzene
(4-phenyl-1,3-butadienyl)benzene
Extension To Discovering Geometric Patterns

- Ongoing work
- Geometric graphs
  - Most of scientific datasets naturally contain 2D/3D geometric information.
  - Each vertex has 2D/3D coordinates associated.
  - Geometric graphs are the same as the purely topological graphs except the coordinates (i.e., edges and vertices have labels assigned).
GFSG:
FSG for Geometric Graphs

- Similar to the original (i.e., topological) FSG
  - Input
    - Geometric graph transactions (graphs with vertex coordinates)
    - Minimum support
  - Output
    - Geometric frequent subgraphs
      - rotation/scaling/translation invariant
      - tolerance radius around each vertex

- How it works
  - Level-by-level approach
    - Candidate generation
    - Candidate pruning
    - Frequency counting

- Make use of geometric information
What Is Good With Geometry?

- **Coordinates** on vertices are helpful
- In topological graph finding, **isomorphism** which is known to be expensive operation, is inevitable.
  - Candidate generation
  - Candidate pruning
  - Frequency counting
- By using coordinates, we can narrow down the search space of (sub)graph isomorphism drastically.
  - **Geometric hashing** (pre-computing geometric configurations)
    - Rotation
    - Scaling
    - Translation
Preliminary Result

- A sample synthetic dataset
  - 1000 transactions
  - 5 vertex labels
  - 1 edge label
  - Edge density 0.75
  - average transaction size 5
  - 10% support
  - Tough setting for FSG because of the # of labels and edge density

- FSG didn’t finish in 13000 seconds
  (also would require post processing)

- GFSG 20 seconds
Future Work

- Pattern Discovery
  - Extension to hypergraphs
  - Incorporating inexact matching

- Classification
  - Provides building blocks for the feature-based classification methods.
    - cf: Evaluation of techniques for classifying biological sequences, M. Deshpande and G. Karypis, 2002
    - cf: Using Conjunction of Attribute Values for Classification, M. Deshpande and G. Karypis, 2002

- Clustering
  - Extend CLUTO to cluster graph transactions.
Thank you!

http://www.cs.umn.edu/~karypis