TALE: A Tool for Approximate Large Graph Matching

Yuanyuan Tian and Jignesh M. Patel
University of Michigan
Motivation

- Graphs are everywhere.
  - Social networks, computer networks, biological networks
- Graph databases are large and growing rapidly in size.
- Wealth of information is encoded in graph databases.

Need: Graph Matching
Motivation

- Previous studies largely focus on **exact** graph matching.
  - Assume precise graph data
  - Subgraph isomorphism (NP-Complete)

- Real life graphs are noisy and incomplete.
  - More challenging (need heuristic methods)

**Need: Approximate Graph Matching**
Motivation

- Most existing methods are applicable to small query graphs.
  - 10s of nodes and edges
- Supporting large queries is more and more desired.
  - Protein Interaction Networks (PINs):
    - 100s ~ 1000s nodes and edges
    - Compare PIN of one species against other species

Need: **Approximate Large Graph Matching**
TALE: A Tool for Approximate Large Graph Matching

- A Novel Disk-based Indexing Method
  - High pruning power
  - Linear index size with the database size

- Index-based Matching Algorithm
  - Significantly outperforms existing methods
  - Gracefully handles large queries and databases

- Experiments on Real Datasets
  - Effectiveness
  - Efficiency
TALE: A Tool for Approximate Large Graph Matching

- **A Novel Disk-based Indexing Method**
  - High pruning power
  - Linear index size with the database size

- **Index-based Matching Algorithm**
  - Significantly outperforms existing methods
  - Gracefully handles large queries and databases

- **Experiments on Real Datasets**
  - Effectiveness
  - Efficiency
Neighborhood Indexing

- Index Unit?

**Neighborhood**
(induced subgraph of a node and its neighbors)

<table>
<thead>
<tr>
<th>Index Unit</th>
<th>Pruning Power</th>
<th>Index Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subgraphs</td>
<td>High 😊</td>
<td>O(n^k) 😞</td>
</tr>
<tr>
<td>Nodes</td>
<td>Low 😞</td>
<td>O(n) 😊</td>
</tr>
<tr>
<td><strong>Neighborhoods</strong></td>
<td>High 😊</td>
<td>O(n) 😊</td>
</tr>
</tbody>
</table>
Index Unit

- Index Unit: Neighborhood
  - Which node is at the center?
    - Node label
  - How many neighbors does the node have?
    - Node degree
  - How do the neighbors connect to each other?
    - NeighborConnection: # edges between neighbors
  - Who are the neighbors?
Index Unit

- Who are the neighbors?
  - Naïve approach: list the labels of the neighbors
    - Problem: the number of neighbors varies.
  - If # labels in the problem domain is a small constant.
    - Deterministic bit array.
      
      \[
      \begin{array}{ccccc}
      A & B & C & D & E \\
      1 & 0 & 0 & 1 & 1 \\
      \end{array}
      \]
      Neighbor Array
  - What if the number of labels is huge?
    - Bloom filter: label $\xrightarrow{\text{hash}}$ position in a m-bit array.

- Information in the index unit
  - (label, degree, nConn, nArray)
Match a Query Neighborhood

**Exact**

- ✓ $N_q$.label = $N_{db}$.label
- ✓ $N_q$.degree $\leq$ $N_{db}$.degree
- ✓ $N_q$.nConn $\leq$ $N_{db}$.nConn
- ✓ (NOT $N_{db}$.nArray)

**Approximate**

- ✓ $\text{group}(N_q$.label) = $\text{group}(N_{db}$.label)
- ✓ $N_q$.degree $\leq$ $N_{db}$.degree + $\epsilon$
- ✓ $N_q$.nConn $\leq$ $N_{db}$.nConn + $\delta$
- ✓ $\left|\text{(NOT } N_{db} \text{.nArray}) \text{ AND } N_q \text{.nArray}\right| \leq \epsilon$

$\rho$ : % of neighbors of a query node with no corresponding matches in the neighborhood of a database node

max # missing neighbors: $\epsilon = \rho \cdot (N_q$.degree)

max # missing nConn: $\delta = \epsilon \cdot (\epsilon - 1)/2 + \epsilon \cdot (N_q$.degree - $\epsilon$)
Index Structure

- Support efficient search for DB neighborhoods.
  
  \[
group(N_{db}.label) = \group(N_q.label) \\
N_{db}.degree \geq N_q.degree - \varepsilon \\
N_{db}.nConn \geq N_q.nConn - \delta \\
|\text{NOT } N_{db}.nArray \text{ AND } N_q.nArray| \leq \varepsilon
\]

- Simple implementation in RDBMSs.
  - Use existing robust disk-based index structures in RDBMSs.

Hybrid Index Structure
Index Probing

- Probe the B+tree for group, degree and nConn
  - Easy

- Probe bitmaps for nArrays
  - Naïve approach: look at each row of a bitmap
  - A better approach
    - Operate on bit slices.
    - Up to 12X speedup!

12
TALE: A Tool for Approximate Large Graph Matching

- A Novel Disk-based Indexing Method
  - High pruning power
  - Linear index size with the database size

- Index-based Matching Algorithm
  - Significantly outperforms existing methods
  - Gracefully handles large queries and databases

- Experiments on Real Datasets
  - Effectiveness
  - Efficiency
Observations

- **Observation 1**: Not every node plays the same role in a graph.
  - Node importance

- **Observation 2**: A good match should be more tolerant towards missing unimportant nodes than missing important nodes.
Matching Algorithm Overview

- **Step 1**: Match the important nodes from the query.
- **Step 2**: Progressively extends the node matches.
TALE Matching Algorithm

- **Step 1**: Match important nodes from the query.
  - Select important nodes.
    - Importance measure: *degree centrality*
    - The percentage of important nodes: \( P \)
  - Probe Neighborhood Index to match important nodes.
  - For each candidate graph in the database, find the one-to-one mappings to the important query nodes.
    - Maximum weighted bipartite graph matching
      - Query nodes
      - Nodes in a DB graph
      - Weight (matching neighbors & neighbor connections)
TALE Matching Algorithm

- **Step 2**: Progressively extends the node matches.
  - Start from the importance node matches.
  - Match “nearby” nodes of already matched nodes.
    - Not just immediate neighbors
    - Also nodes two hops away
      - gap nodes
      - differences in node connectivity
TALE: A Tool for Approximate Large Graph Matching

- A Novel Disk-based Indexing Method
  - High pruning power
  - Linear index size with the database size

- Index-based Matching Algorithm
  - Significantly outperforms existing methods
  - Gracefully handles large queries and databases

- Experiments on Real Datasets
  - Effectiveness
  - Efficiency
Experimental Evaluation

- Implementation
  - C++ on top of PostgreSQL

- Evaluation Platform
  - 2.8GHz P4, 2GB RAM, 250GB SATA disk, FC2
  - PostgreSQL: version 8.1.3, 512 MB buffer pool

- Experimental Datasets
  - BIND protein interaction networks
  - ASTRAL protein structures

- Evaluation Measures:
  - Effectiveness
  - Efficiency
# Effectiveness Experiment

- **Protein Interaction Network Comparison (BIND)**

## Table

<table>
<thead>
<tr>
<th></th>
<th>#node</th>
<th>#edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>rat</td>
<td>830</td>
<td>942</td>
</tr>
<tr>
<td>mouse</td>
<td>2991</td>
<td>3347</td>
</tr>
<tr>
<td>human</td>
<td>8470</td>
<td>11260</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>#KEGGs hit</th>
<th>KEGG coverage</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rat vs. human</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>0</td>
<td>NA</td>
<td>910.0</td>
</tr>
<tr>
<td><strong>TALE</strong></td>
<td>6</td>
<td>3.2%</td>
<td>0.3</td>
</tr>
<tr>
<td><strong>mouse vs. human</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>18</td>
<td>5.0%</td>
<td>16305.5</td>
</tr>
<tr>
<td><strong>TALE</strong></td>
<td>42</td>
<td>13.6%</td>
<td>0.8</td>
</tr>
</tbody>
</table>

- **# KEGGs hit**: number of pathways aligned between 2 species
- **KEGG coverage**: fraction of proteins aligned within a pathway.
Efficiency Experiment

- Query increasing sized ASTRAL datasets
  - 20 queries (153.1n, 592.0e)
  - Top 20 results
Related Work

- Index-based Approximate Graph Matching
  - Graphfil, PIS, CDIndex, C-Tree, SAGA
  - Limited approximation: Graphfil, PIS, CDIndex, C-Tree
  - For small queries: Graphfil, PIS, CDIndex, SAGA

- Pairwise Graph Alignment Methods
  - NetworkBlast, MaWIsh, Graemlin
  - Specific to protein interaction networks
  - Very slow for database search (no index)
Conclusion

- TALE → *Approximate Large Graph Matching*
- **Neighborhood Indexing**
  - Disk-based index using existing index structures in RDBMSs
  - High pruning power
  - Linear index size with the database size
- **Index-based Matching Algorithm**
  - Distinguish nodes by importance
  - Match important nodes then extend to others
- **Experiments on Real Datasets**
  - Improved effectiveness and efficiency over existing methods
Questions?
Suggestions?
Thanks! 😊