Fast and Practical Indexing and Querying of Very Large Graphs

Silke Trißl & Ulf Leser
Humboldt-Universität zu Berlin
Knowledge Management in Bioinformatics

Published at SIGMOD Conference 2007
Outline

- Motivation
- The GRIPP Index Structure
  - Construction
  - Properties
- Querying GRIPP
  - General query strategy
  - Pruning – simple, skip, and stop
- Evaluation and Comparison
- Conclusion
Very Large Graphs

- Biological networks
  - Metabolic networks
    - KEGG: 20,000 nodes, 40,000 edges
  - Protein-protein interaction networks
    - PubGene – literature mining for interactions
    - 1 million nodes, 10 million edges

- XML-documents containing XLinks
- Social interaction networks
- World Wide Web
Queries on Graphs: Reachability

- Given a graph, \( G = (V, E) \)
  - Labeled nodes, \( V \)
  - Directed, unlabeled edges, \( E \) \( V \times V \)

Answering reachability queries
# Reachability – Naïve Methods

<table>
<thead>
<tr>
<th>Recursive query strategies</th>
<th>Indexing</th>
<th>Querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>- PL/SQL functions</td>
<td>+ No index required</td>
<td>- Slow query answering</td>
</tr>
<tr>
<td>- SQL – Standard 2003</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transitive Closure</th>
<th>indexing</th>
<th>querying</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Agrawal et al. 1989]</td>
<td>- Space to store and time to compute index</td>
<td>+ Fast query answering</td>
</tr>
</tbody>
</table>

**GRIPP:**

<table>
<thead>
<tr>
<th>Query time: 3.0 ms</th>
<th>Index creation: 23 seconds</th>
</tr>
</thead>
</table>

60 mio. tuples
Outline

- Motivation
- The GRIPP Index Structure
  - Construction
  - Properties
- Querying GRIPP
  - General query strategy
  - Pruning – simple, skip, and stop
- Evaluation and Comparison
- Conclusion
Pre- and Postorder Numbering

- Indexing scheme for trees
  - Node labeling during depth-first traversal of $G$
    - $w$ reachable from $v$ iff $v_{pre} < w_{pre} < v_{post}$

  ![Diagram]

- Very popular for XML [Grust et al., 2004]
From Trees to Graphs

- Inserting a “non-tree” edge

- Additional pre- and postorder values
  - target node of additional edge
  - all successors of that node

- For DAGs: exponential growth of the index size [Trißl et al., 2005]
Extension to Graphs – GRIPP

- General Idea of GRIPP

- Given a graph, $G = (V, E)$

- Create the GRIPP index table, $IND(G)$
  - one instance for every node $v$ in $G$
    - Node identifier
    - Preorder value
    - Postorder value
    - Instance type
Index creation

- Depth-first traversal of $G$

- We reach a node $v$
  - for the first time
    - add *tree instance* of $v$ to $\text{IND}(G)$
    - proceed traversal
  - again
    - add *non-tree instance* of $v$ to $\text{IND}(G)$
    - do not traverse child nodes of $v$

Graph, $G$ is partitioned in *tree* (solid) and *non-tree edges* (dashed)
GRIPP Index Table, $\text{IND}(G)$

- **Size of $\text{IND}(G)$:** $O(|E|+|V|)$

<table>
<thead>
<tr>
<th>node</th>
<th>pre</th>
<th>post</th>
<th>inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0</td>
<td>21</td>
<td>tree</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>20</td>
<td>tree</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>7</td>
<td>tree</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>4</td>
<td>tree</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>6</td>
<td>tree</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>9</td>
<td>tree</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>19</td>
<td>tree</td>
</tr>
<tr>
<td>G</td>
<td>11</td>
<td>14</td>
<td>tree</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
<td>13</td>
<td>non</td>
</tr>
<tr>
<td>H</td>
<td>15</td>
<td>18</td>
<td>tree</td>
</tr>
<tr>
<td>A</td>
<td>16</td>
<td>17</td>
<td>non</td>
</tr>
</tbody>
</table>

Motivation

GRIPP index

Construction

Properties

Querying

General

Pruning

Evaluation

Conclusion
Order Tree, $O(G)$

<table>
<thead>
<tr>
<th>node</th>
<th>pre</th>
<th>post</th>
<th>inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0</td>
<td>21</td>
<td>tree</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>20</td>
<td>tree</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>7</td>
<td>tree</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>4</td>
<td>tree</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>6</td>
<td>tree</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>9</td>
<td>tree</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>19</td>
<td>tree</td>
</tr>
<tr>
<td>G</td>
<td>11</td>
<td>14</td>
<td>tree</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
<td>13</td>
<td>non</td>
</tr>
<tr>
<td>H</td>
<td>15</td>
<td>18</td>
<td>tree</td>
</tr>
</tbody>
</table>

Advantage:
now we can query a tree

Non-tree instances: Always leaf node
Outline

- Motivation
- The GRIPP Index Structure
  - Construction
  - Properties
- Querying GRIPP
  - General query strategy
  - Pruning – simple, skip, and stop
- Evaluation and Comparison
- Conclusion
General query strategy

- Is node C reachable from node D?

  - Querying $O(G)$ once is not sufficient
  - We must extend the search

Yes in $G$  No in $O(G)$
Query strategy – Step 1

- Retrieve the *reachable instance set* of start node \( v \), called \( RIS(v) \)

- Retrieve \( RIS(D) \)
  - Requires only a single query on \( IND(G) \)
  - If \( C \in RIS(D) \)
    - return *true*
    - stop the search
  - Else
    - proceed to **Step 2**
Query strategy – Step 2

- Search for non-tree instances in $RIS(v)$
  - The nodes of these instances are *hop nodes*

- Check every $i \in RIS(D)$
  - If $i$ is tree instance
    - [G and H]
    - Done
  - If $i$ is non-tree instance
    - [A and B]
    - $i$ has no successors in $O(G)$, but possibly in $G$
    - proceed to **Step 3**
Query strategy – Step 3

- Extend the search
  - using hop nodes $v_1, \ldots, v_n$

- Obtain the tree instance of node B
- Proceed to Step 1

- Repeat steps 1…3 until
  - an instance of node C is found
  - no more hop nodes are available

Depth-first traversal of $O(G)$ using hop nodes
Naïve Implementation of GRIPP

- Using every non-tree instance as hop
  - Worst case: $|E| - |V|$ hop nodes
  - Still better than recursive search

- Acceptable for tree-like graphs
  - few hop nodes

- Disaster on denser graphs
  - many hop nodes
Pruning strategies

- Pruning hop nodes
  - Simple
    - Never retrieve reachable instance sets twice
  - Skip
    - Skip already searched areas
  - Stop
    - Certain nodes cannot have hop nodes in their reachable instance set
    - This property can be pre-computed
Simple and Skip-Strategy

- Keep list of used hop nodes, \( U \)
- When examining a new hop node \( h \)
  - four possible positions of \( h \) relative to \( u \in U \)
  - \( h = u \)
  - \( h \in \text{RIS}(u) \)
  - \( h \) sibling to \( u \)
  - \( u \in \text{RIS}(h) \)

\[
\begin{align*}
\text{No} & \quad \text{No} & \quad \text{Yes} & \quad \text{Yes, but skip}
\end{align*}
\]
Stop strategy

- Node $s$ is a **stop node** iff
  - all non-tree instances in $RIS(s)$ also have a tree instance in $RIS(s)$ or are $s$

- Node A is a stop node
  - no hop node in $RIS(A)$ can be used
Advantage of Stop Nodes

- Use s as query or hop node
  - no need to check non-tree instances in $RIS(s)$
- List of stop nodes can be precomputed

GRIPP order tree for graph with 100 nodes and 200 edges

Silke Trißl & Ulf Leser - Indexing and Querying of Very Large Graphs - SIGMOD 2007
Queries in GRIPP

- Querying GRIPP
  - Constantly cover larger parts of $O(G)$
  - Preferably use stop nodes as hop nodes
- Behavior independent of graph size
  - Only size of intervals increases

GRIPP order tree for graph with 100 nodes and 200 edges
Outline

- Motivation
- The GRIPP Index Structure
  - Construction
  - Properties
- Querying GRIPP
  - General query strategy
  - Pruning – simple, skip, and stop
- Evaluation and Comparison
- Conclusion
Setup

- Inside a RDBMS as user defined function
  - Very simple installation
    - Just define tables and functions
  - Non-intrusive
    - Index is stored as relation
    - Answering reachability queries – just one function call using SQL
  - Memory management delegated to RDBMS
Evaluation

Graphs tested
- Generated random and scale-free graphs
  - graphs with up to 5 million nodes
  - graph densities between 1 and 20
- Real life graph – biological networks
  - KEGG, aMAZE, Reactome

Evaluation
- Index creation time and size
- Average query time and
  average number of recursive queries
  - over 1,000 randomly selected node pairs
Competing methods

- **Label+SSPI** [Chen et al., 2005]
  - Pre- and postorder labeled spanning tree $T$
  - Non-tree edges are stored in an additional data structure, called SSPI
  - At query time: First scan $T$, then traverse SSPI

- **Dual labeling** [Wang et al., 2006]
  - Pre- and postorder labeled spanning tree $T$
  - Non-tree edges are stored in a compressed transitive closure
  - Queries are answered in constant time
Index Creation

- Comparing index creation times
  - GRIPP, Label + SSPI, and Dual Labeling

<table>
<thead>
<tr>
<th>No. nodes</th>
<th>GRIPP</th>
<th>Label + SSPI</th>
<th>Dual Labeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>2.3</td>
<td>4.0</td>
<td>11.7</td>
</tr>
<tr>
<td>5,000</td>
<td>11.3</td>
<td>23.0</td>
<td>922.9</td>
</tr>
<tr>
<td>10,000</td>
<td>23.0</td>
<td>50.6</td>
<td>7,971.7</td>
</tr>
<tr>
<td>100,000</td>
<td>235.8</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>1,000,000</td>
<td>2,539.8</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>5,000,000</td>
<td>16,062.5</td>
<td>...</td>
<td>-</td>
</tr>
</tbody>
</table>

Scale-free graphs with 100% more edges than nodes, index creation time in seconds.

- Index size – see paper
## Queries

### Average query time (ms) and standard deviation

<table>
<thead>
<tr>
<th>No. nodes</th>
<th>GRIPP</th>
<th>Label + SSPI</th>
<th>Dual Labeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>1.5 ± 1.45</td>
<td>5.9 ± 13.39</td>
<td>0.8 ± 0.33</td>
</tr>
<tr>
<td>5,000</td>
<td>2.0 ± 2.15</td>
<td>22.7 ± 59.39</td>
<td>0.8 ± 0.32</td>
</tr>
<tr>
<td>10,000</td>
<td>2.1 ± 2.56</td>
<td>48.8 ± 127.67</td>
<td>0.8 ± 0.32</td>
</tr>
<tr>
<td>50,000</td>
<td>4.4 ± 6.74</td>
<td>253.0 ± 637.68</td>
<td>-</td>
</tr>
</tbody>
</table>

### Average number of queries and standard deviation

<table>
<thead>
<tr>
<th>No. nodes</th>
<th>GRIPP</th>
<th>Label + SSPI</th>
<th>Dual Labeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>1.8 ± 0.74</td>
<td>22.0 ± 52.30</td>
<td>1.0 ± 0.00</td>
</tr>
<tr>
<td>5,000</td>
<td>1.9 ± 0.82</td>
<td>92.1 ± 238.31</td>
<td>1.0 ± 0.00</td>
</tr>
<tr>
<td>10,000</td>
<td>1.8 ± 0.77</td>
<td>194.7 ± 497.68</td>
<td>1.0 ± 0.00</td>
</tr>
<tr>
<td>50,000</td>
<td>1.9 ± 0.77</td>
<td>944.3 ± 2,419.83</td>
<td>-</td>
</tr>
</tbody>
</table>
Query times – GRIPP itself

- Increasing graph size
  - from 1,000 nodes and 2,000 edges
  - to 5 million nodes and 10 million edges

- Maximum 10 recursive queries
Query times – GRIPP itself

- Increasing graph density
  - from $|E| = 1.5 \cdot |V|$ ($|V| = 100,000$)
  - to $|E| = 4.5 \cdot |V|$

![Graph showing query times](image)
Conclusion

- GRIPP is **well suited for very large graphs**
  - Index creation and index size *linear*
  - Query times almost *constant*

- GRIPP is **easy to use**
  - Implemented inside a RDBMS

- GRIPP is **extensible**
  - Set-oriented queries
    - given a node $v$ and a set of nodes $W$
  - Path length queries
Thanks for your attention

Silke Trißl & Ulf Leser
Humboldt-Universität zu Berlin

Work sponsored by