Path-Hop: efficiently indexing large graphs for reachability queries

Tylor Cai and C.K. Poon
CityU of Hong Kong
Reachability Query

- Query(u,v): Is there a directed path from vertex u to vertex v in graph G?
Reachability Query

- Query(u,v): Is there a directed path from vertex u to vertex v in graph G?
Reachability Query

• Query($u,v$): Is there a directed path from vertex $u$ to vertex $v$ in graph $G$?
Reachability Query

- Query(u,v): Is there a directed path from vertex u to vertex v in graph G?
Graph Indexing for Reachability Queries

• Given a directed graph $G$, build an index to support reachability queries

Reachability Index

$\leftarrow(u_1,v_1), (u_2,v_2), (u_3,v_3), \ldots$
Applications

- XML with IDREF/ID reference links
  - Modelled as directed graph (not just tree)
- Semantic Webs
  - Ontology graph
- Bioinformatics
  - Data such as metabolic pathways, protein interaction, etc. modelled as directed graph
The challenge

• Graph is huge
  - DFS/BFS at query time too slow, $O(m)$ time
  - Pre-computing the Transitive Closure requires too much storage, $O(n^2)$ storage

• Goal: to design an index with
  - small size (better than TC),
  - fast query time (better than DFS/BFS), and
  - reasonable construction time
Assumptions & Terminologies

• Assume G is a Directed Acyclic Graph (DAG) – a standard assumption
• (u,v) is called a *reachable pair* iff u can reach v
• Set of all reachable pairs = Transitive Closure
Previous Work: Tree Cover

- Find a tree cover (spanning tree)
- Label vertices with an interval labeling
  - Reachable pairs \((u,v)\) where \(u\) is a tree-ancestor of \(v\) covered
- Uncovered reachable pairs compressed using tree cover
  (*to be explained*)

[Agarwal et al. SIGMOD’89]
Previous Work: Tree Cover

- Find a tree cover (spanning tree)
- Label vertices with an interval labeling
  - Reachable pairs \((u,v)\) where \(u\) is a tree-ancestor of \(v\) covered
- Uncovered reachable pairs compressed using tree cover (to be explained…)

[Agarwal et al. SIGMOD’89]
Previous Work: Tree Cover

- Find a tree cover (spanning tree)
- Label vertices with an interval labeling
  - Reachable pairs \((u,v)\) where \(u\) is a tree-ancestor of \(v\) covered
- Uncovered reachable pairs compressed using tree cover
  
  (to be explained…)
Previous Work: Tree Cover

• Find a tree cover (spanning tree)
• Label vertices with an interval labeling
  ➢ Reachable pairs \((u, v)\) where \(u\) is a tree-ancestor of \(v\) covered
• Uncovered reachable pairs compressed using tree cover
  (to be explained…)

[Agarwal et al. SIGMOD’89]
Previous Work: Chain Cover

• Find a chain cover (collection of disjoint chains covering all vertices)

• Label each vertex by ChainID & Sequence No. in its chain
  - Reachable pairs \((u,v)\) where \(u, v\) in same chain covered

• Uncovered reachable pairs compressed using chains \((to be explained… )\)
Previous Work: Chain Cover

- Find a chain cover (collection of disjoint chains covering all vertices)
- Label each vertex by ChainID & Sequence No. in its chain
  - Reachable pairs \((u,v)\) where \(u, v\) in same chain covered
- Uncovered reachable pairs compressed using chains \((to be explained...\))
Previous Work: Chain Cover

- Find a chain cover (collection of disjoint chains covering all vertices)
- Label each vertex by ChainID & Sequence No. in its chain
  - Reachable pairs \((u,v)\) where \(u, v\) in same chain covered
- Uncovered reachable pairs compressed using chains (*to be explained*...)

[Jagadish TODS'90]
Previous Work: Chain Cover

• Find a chain cover (collection of disjoint chains covering all vertices)
• Label each vertex by ChainID & Sequence No. in its chain
  ➢ Reachable pairs \((u,v)\) where \(u, v\) in same chain covered
• Uncovered reachable pairs compressed using chains (*to be explained...*)

[Diagram of chained vertices with ChainIDs and Sequence Nos.]

[Jagadish TODS'90]
Previous Work: 2-Hop Cover

- Each vertex \( v \) keeps two sets:
  - \( L_{\text{in}}(v) \): set of some vertices that can reach \( v \)
  - \( L_{\text{out}}(v) \): set of some vertices that \( v \) can reach
- For each reachable pair \((u,v)\):
  - find an intermediate vertex \( c \) along some path from \( u \) to \( v \);
  - add \( c \) to \( L_{\text{out}}(u) \) and \( L_{\text{in}}(v) \)
- Processing query \((u,v)\):
  - \((u,v)\) reachable pair iff \( L_{\text{out}}(u) \cap L_{\text{in}}(v) \neq \emptyset \)

[Cohen et al. SODA'02]
Finding a good 2-hop cover

• Aim: to minimize $\sum_u |L_{in}(u)| + |L_{out}(u)|$
• Equivalent to a weighted set cover problem:
  ✓ Ground Set: set of reachable pairs
  ✓ Collection of candidate sets:
    \[ \{ C_{in} \times C_{out} \mid C_{in} \text{ subset of vertices that reach } w, \]
    \[ C_{out} \text{ subset of vertices that } w \text{ reaches } \} \]
  ✓ Weight of $C_{in} \times C_{out} = |C_{in}| + |C_{out}|$
• Apply greedy set cover algorithm
  ✓ Exponentially many candidates for each $w$
  ✓ Consider only some of them
• Slow construction time, $O(n^4)$
Previous Work (cont’d)

• Dual Labeling [Wang et al. ICDE’06]
  - Tree Cover plus TLC matrix
• Path-Tree Cover [Jin et al. SIGMOD’08]
  - Combine Tree and Chain Cover
  - Fast query time
  - Space-efficient for sparse graphs
• 3-Hop Cover [Jin et al. SIGMOD’09]
  - Enhanced Chain Cover + 2-Hop
  - Most space-efficient, slow construction time
3-Hop Cover

- TC Contour: corners of off-diagonal blocks
- Compressed using 2-hop
- Diagonal blocks covered by Chain ID, Sequence No.
• (1,5) dominated by (1,4)
• (2,6) dominated by (3,6)
An Observation

- Chain cover
  - CiteSeer: 50% chains are single vertices, 87% chains have \( \leq 2 \) vertices
  - Yago: 87% chains are single vertices, 96% chains have \( \leq 2 \) vertices

- Many short chains

- Can we cover the graph using graph structures other than chains?
Tree Cover & Residual TC

- Only roots of subtrees need to be stored
<table>
<thead>
<tr>
<th>Dataset</th>
<th>TC Contour</th>
<th>RTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citeseer</td>
<td>152K</td>
<td>61K</td>
</tr>
<tr>
<td>Go</td>
<td>46K</td>
<td>14K</td>
</tr>
<tr>
<td>Pubmed</td>
<td>288K</td>
<td>82K</td>
</tr>
<tr>
<td>Yago</td>
<td>30K</td>
<td>22K</td>
</tr>
</tbody>
</table>

- Size of residual TC much smaller than size of TC Contour
Path-Hop

• Combine Tree Cover & 2-Hop

2-hop (u,v) in TC
3-hop (u,v) in TC Contour
path-hop (u,v) in Residual TC
Path-Hop

• Index Construction:
  - Find a tree cover; label vertices by an interval labeling scheme
  - Compute RTC
  - Repeat
    • select a path-hop $x \rightarrow y$
    • Add $x$ to $L_{out}(u)$ if $(u,x)$ in RTC
    • Add $y$ to $L_{in}(v)$ if $(y,v)$ in RTC
  Until all reachable pairs in RTC covered
Path-Hop

• Processing Query(u,v):
  - If u is tree-ancestor of v, return “YES”
  - For each ancestor z of v
    - If there is x in $L_{out}(u)$ and y in $L_{in}(z)$ such that x is tree-ancestor of y, return “YES”
  - return “NO”
Experiments

- Implemented in C++ on OpenSolaris
- Intel 3.4 GHz x86 / 2GB RAM
- 4 synthetic datasets, 4 real datasets (available online)

| Dataset | $|V_{DAG}|$ | $|E_{DAG}|$ |
|---------|--------|--------|
| **Synthetic** | | |
| rand2k_8 | 2000 | 15781 |
| rand2k_10 | 2000 | 19740 |
| rand2k_12 | 2000 | 23789 |
| Xmark | 6080 | 6957 |
| **Real Life** | | |
| Citeseer | 10720 | 44258 |
| Go | 6793 | 13361 |
| PubMed | 9000 | 40028 |
| Yago | 6642 | 42392 |
• Methods tested
  ➢ Dual labeling (Dual II)
  ➢ Path-Tree (PTree I)
  ➢ 3-Hop labeling (3-Hop Contour)
  ➢ Path-Hop
## Index Size

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dual II</th>
<th>PTree I</th>
<th>3-Hop Contour</th>
<th>Path-Hop</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rand2k_8</td>
<td>781</td>
<td>89</td>
<td>33</td>
<td>31</td>
<td>930</td>
</tr>
<tr>
<td>Rand2k_10</td>
<td>979</td>
<td>82</td>
<td>34</td>
<td>30</td>
<td>1130</td>
</tr>
<tr>
<td>Rand2k_12</td>
<td>1119</td>
<td>76</td>
<td>33</td>
<td>29</td>
<td>1273</td>
</tr>
<tr>
<td>Xmark</td>
<td>12</td>
<td>22</td>
<td>27</td>
<td>19</td>
<td>3429</td>
</tr>
<tr>
<td>Citeseer</td>
<td>314</td>
<td>92</td>
<td>62</td>
<td>54</td>
<td>433</td>
</tr>
<tr>
<td>Go</td>
<td>47</td>
<td>38</td>
<td>35</td>
<td>27</td>
<td>111</td>
</tr>
<tr>
<td>PubMed</td>
<td>394</td>
<td>108</td>
<td>64</td>
<td>53</td>
<td>532</td>
</tr>
<tr>
<td>Yago</td>
<td>52</td>
<td>39</td>
<td>34</td>
<td>33</td>
<td>73</td>
</tr>
</tbody>
</table>

*(rounded to nearest Kbytes)*
Query Processing Time

![Bar chart showing query processing time for different datasets and query processing methods. The x-axis represents synthetic and real datasets, and the y-axis represents query processing time in milliseconds. The legend includes PTree I, 3-Hop Contour, and Path-Hop.]
Query Processing Time

![Query Processing Time Graph]

Total time for processing 100,000 random queries

- **Query Processing Time (ms)**
- **Synthetic and Real Datasets**
- **PTree I**
- **3-Hop Contour**
- **Path-Hop**

Dual II is too slow
Index Construction Time

Synthetic and Real Data Sets:
- rand2k_8
- rand2k_10
- rand2k_12
- Xmark
- Citeseer
- Go
- Pubmed
- Yago

Index Construction Time (s)

- Dual II
- PTree I
- 3-Hop Contour
- Path-Hop
Future Directions

• Can we design an even more succinct index structure for reachability?
• Can we design a scheme that offers a tradeoff between index size and query time?
• Is randomization helpful?
• What about other queries (e.g. shortest distance/path)?
The END

Questions?