Invited Talk: GraphSM/DBKDA-2014

The Sixth International Conference on Advances in Databases, Knowledge, and Data Applications

April 20 - 26, 2014 - Chamonix, France

About Reachability in Graphs

Andreas Schmidt

(1) Department of Informatics and Business Information Systems
University of Applied Sciences Karlsruhe
Moltkestraße 30
76133 Karlsruhe
Germany

(2) Institute for Applied Sciences
Karlsruhe Institute of Technology
PO-box 3640
76021 Karlsruhe
Germany
Outlook

- Motivation
- Some Graph definitions
- Different Approaches
- Summary & Further Readings
Motivation

Reachability queries are a very basic type of a graph query

Why do we need reachability queries?

- Bioinformatics (biological networks, genome biology)
- Social Science, link analysis, citation analysis
- XML Queries/Database query optimizer
- Internet routing
- Source Code Analysis
- Geographic navigation systems
- Ontology queries (RDF/OWL)
Directed Graphs

• Graph G:

\[ G = (V, E) \]

\[ V = \{v_1, v_2, ..., v_n\} \]

E: binary relation on V
\[ E = \{(v_1, v_2), (v_2, v_3), (v_2, v_5), ... \} \]

• Further concepts:
  • Path
  • Path length
  • Cyclic/Acyclic graph
Representation Forms for Directed Graphs

**Adjacency list**

1. \(1 \rightarrow 2\)
2. \(2 \rightarrow 3 \rightarrow 4 \rightarrow 5\)
3. \(3 \rightarrow 7\)
4. \(4 \rightarrow 1 \rightarrow 4 \rightarrow 6\)
5. \(5 \rightarrow 3 \rightarrow 6\)
6. \(6\)
7. \(7 \rightarrow 5\)

Memory: \(O(|V| + |E|)\)
Access: \(O(|V|)\) - nodes unsorted
\(O(\log_2|V|)\) - nodes sorted

**Adjacency matrix**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Memory: \(O(|V|^2)\)
Access: \(O(1)\)
Reachability Query Types

• Query Types:
  • single pair
  • single source
  • multi source reachability

• Approaches:
  • Query on demand using breath- or depth-first search.
  • Precalculate the transitive closure, which contains all the reachability information
  • Something in between the two above solutions
Summary Breath-/Depth-First Search

- Query Time: $O(|V| + |E|)$
- Additional memory consumption: none
- For large graphs too slow to answer queries efficiently
Transitive Closure

\[ G^+ = (V, E^+) \]
\[ E^+ = \{u, v \in V: u \rightarrow v \in G\} \]
$G^+ = (V, E^+)$
$E^+ = \{u, v \in V : u \rightarrow v \in G\}$:

Transitive Closure
Transitive Closure

Adjacency list

1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7
2 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7
3 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7
4 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 6
5 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7
6
7 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7

Memory: $\mathcal{O}(|V| + |E^+|)$
Access: $\mathcal{O}(|V|)$ - nodes unsorted
$\mathcal{O}(\log_2|V|)$ - nodes sorted

Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Memory: $\mathcal{O}(|V^2|)$
Access: $\mathcal{O}(1)$
Summary Transitive Closure

- Query Time
  - adjacency matrix: $O(1)$
  - unsorted adjacency list: $O(|V|)$
  - sorted adjacency list: $O(\log_2|V|)$
- Memory Consumption
  - adjacency matrix: $O(|V|^2)$
  - adjacency list: $O(|V| + |E^+|)$
- Additional Index construction time: $O(|V| \times |E|)$
### Time/Space Complexity of different approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Query Time</th>
<th>Index Const. Time</th>
<th>Index Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transitive Closure</td>
<td>O(1)</td>
<td>O(n * m)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Tree+SSPI</td>
<td>O(m - n)</td>
<td>O(n + m)</td>
<td>O(n + m)</td>
</tr>
<tr>
<td>GRIPP</td>
<td>O(m - n)</td>
<td>O(n + m)</td>
<td>O(n + m)</td>
</tr>
<tr>
<td>Dual-Labeling</td>
<td>O(1)</td>
<td>O(n + m + t³)</td>
<td>O(n + t²)</td>
</tr>
<tr>
<td>Tree Cover</td>
<td>O(log n)</td>
<td>O(nm)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Chain Cover</td>
<td>O(log k)</td>
<td>O(n² + knk¹/₂)</td>
<td>O(n * k)</td>
</tr>
<tr>
<td>Path-Tree Cover</td>
<td>O(log² k´)</td>
<td>O(m * k´) or O(n * m)</td>
<td>O(n * k´)</td>
</tr>
<tr>
<td>2-Hop Cover</td>
<td>O(m¹/₂)</td>
<td>O(n³</td>
<td>T * C</td>
</tr>
<tr>
<td>3-Hop Cover</td>
<td>O(log n + k)</td>
<td>O(kn² *</td>
<td>Con(G)</td>
</tr>
<tr>
<td>BFS/DFS</td>
<td>O(n + m)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Some ideas ...

- Algorithms optimized for spares/dense graphs
- Transitive closure only over subgraphs
- Spanning tree (i.e. single interval tree coding schema - SIT) + additional data structure
- Represent adjacency matrix as compressed bitmaps

=> see literature at the end for details ...

- Reduction of graph size
  (Strongly connected components)
Strongly Connected Components
Strongly Connected Components
Tarjan’s Algorithm

- Depth first search
- start at arbitrary node
- Time complexity:
  \[ \mathcal{O}(|V|+|E|) \]
- Looks for cycles in graph
- Cycles are shrunked to a single nodes
Summary

- Reachability queries in graphs seem at first glance very simple queries
- But ...
  - in reality they have a wide range of use (query optimization, bioinformatics, social science, internet routing, geographic information systems, ...)
  - are not so simple to answer (quickly)
  - A wide range of algorithms have been developed to solve this problem for special cases (published at SIGMOD, ICDE, VLDB)
  - Always tradeoff between query time and memory consumption + index construction time
Literature

• [SM11] Sebastiaan J. van Schaik, Oege de Moor: A memory efficient reachability data structure through bit vector compression. SIGMOD Conference 2011: 913-924
How to find strongly connected components?

• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: $O(|V+E|)$
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:

```plaintext
(a, 1)
```

---

Andreas Schmidt - GraphSM/DBKDA - 2014
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  \[O(|V+E|)\] Time complexity:

  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \(O(|V+E|)\)
  • Looks for cycles in graph
  • Cycles are shrunked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
Tarjan’s Algorithm:
- Depth first search
- Start at arbitrary node
- Time complexity: $O(|V+E|)$
- Looks for cycles in graph
- Cycles are shrunk to a single node
- Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \( \mathcal{O}(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrunked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    $O(|V+E|)$
  • Looks for cycles in graph
  • Cycles are shrunk to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( \mathcal{O}(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \( \mathcal{O}(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( \mathcal{O}(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \[ O(|V+E|) \]
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity:
    \( \mathcal{O}(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example:
• Tarjan’s Algorithm:
  • Depth first search
  • start at arbitrary node
  • Time complexity: \( O(|V+E|) \)
  • Looks for cycles in graph
  • Cycles are shrinked to a single node
  • Example: