Advanced Data Management

Medha Atre

Office: KD-219
atrem@cse.iitk.ac.in

Sept 26, 2016

Reachability defined

Given a graph $G(V, E)$ with V as the set of nodes and E as the set of edges, a reachability query asks – does there exists *any* path between nodes x and y. The pair of nodes can be any two nodes from the given graph.
In case of directed graph, the path is directed and other for undirected graphs the path is undirected.
The graphs may have cycles too, giving rise to the *strongly connected components*.

Query Processing Challenges

- Identification of strongly connected components and merging them.
- Efficient traversal over large graphs, e.g., number of vertices and edges being several millions.
- High computational complexity $O(|V|^3)$ for standard algorithms like all-pair shortest path, which also gives us *transitive clousure* for answering reachability.
- High storage space $O(|V|^2)$ if the entire transitive clousure is meant to be stored.

Key aspects to consider

- Index creation time, e.g., $O(|V|^3)$ for a naïve technique like all pair shortest path.
- Index size, e.g., $O(|V|^2)$ for storing complete transitive closure.
- Query answering time, $O(1)$ if we have complete transitive closure.
- However, the first two aspects – time to create index and the index size – become prohibitives large if the graph is dense and has a large number of nodes.
- Hence several approaches defined over the past decade to either reduce index creation time, index size, or both, at the cost of compromising *some* query answering time.

Existing published approaches

- Approaches which answer queries using only index (node-idx-labels)
  - These focus on building a reachability index such that for any pair of nodes (u, v) the reachability query over them can be answered using *only* this index without traversing any part of the original graph.
  - 2-Hop [SODA 2002], Path-Tree [SIGMOD 2008], 3-Hop [SIGMOD 2009], PWAH8 [SIGMOD 2011], TF-label [SIGMOD 2013], Hierarchical Labeling (HL) and Distribution Labeling (DL) [VLDB 2013].
  - Not all of these build indexes of $O(|V|^2)$, they try to improve upon this worst case bound.

Existing published approaches

- Approaches that answer queries by partly using the index
  - These focus on reducing the time spent in building the index and the size (space) of the index.
  - For certain queries they have to traverse part of the graph to give a definitive answer to the given reachability query. For such queries the index alone cannot answer the reachability query definitively, since it is *incomplete*.
  - Tree+SSPI [VLDB 2005], GRIPP [SIGMOD 2007], GRAIL [VLDB 2010], Ferrari [ICDE 2013], SCARAB [SIGMOD 2012].

2-hop cover [SODA 2002]

- The aim is to reduce the size of the reachability index using a method called *2-hop cover* such that entire transitive closure is not needed.
- 2-hop Reachability Labeling: each vertex $v \in V$ has a label $L(v) = (L_{in}(v), L_{out}(v))$. $L_{in}(v)$ contains all nodes $x \in V$ such that v can be reached from x. L_{out} is defined similarly.
- $u \rightsquigarrow v$ iff $L_{out}(u) \cap L_{in}(v) \neq \phi$.
- 2-hop covers: for every (u, v) s.t. $u \rightsquigarrow v$, let P_{uv} be all the paths between (u, v).
- A "hop" is defined as $(\mathcal{H}, u)$ s.t. $\mathcal{H}$ is a path which ends in u. u is said to be the "handle" of hop $\mathcal{H}$.

2-hop cover

- A collection of such hops is called a 2-hop cover, if for every $u, v \in V$, where v is reachable from u, there exist two hops $(\mathcal{H}_1, u)$ and $(\mathcal{H}_2, v)$ where $\mathcal{H}_1.\mathcal{H}_2$ gives *some* path between u and v.
- Optimal 2-hop cover problem is NP-hard as it can be reduced to the optimal set-cover problem.
- But due to the above observation, we use greedy method of set-cover for 2-hop cover problem.
- Start with a ground set $T = \{(u, v) | P_{uv} \neq \phi\}$.
- We will update this T and call it T' in each iteration, by removing the pairs of nodes, as we *cover* them with the hops.

2-hop cover

- For each vertex $w \in V$, we construct an undirected bipartite graph $G_w = (V_w, E_w)$ where V_w contains L_{in} and L_{out} nodes representing the respective sets for *every* other node in the graph.
- There is an edge between these nodes, if w is a node on *some* path between u, v.
- From all such bipartite graphs for each w, iteratively choose the w for which the following ratio is *maximized* until T' is not empty (equivalent to greedy set-cover):

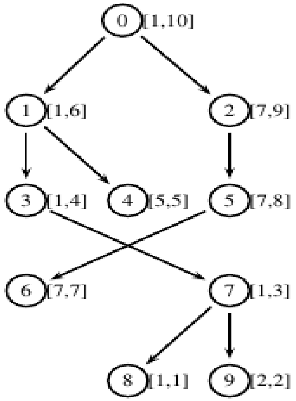
$$\frac{\{(u, v)\} \cap T'}{|C_{in}(v)| + |C_{out}(u)|}$$

- This greedy method is also equivalent to finding *densest subgraph* of each G_w.

GRAIL [VLDB 2010]

- **Background:** for a directed tree-shaped graph, we can create complete *interval* reachability label of type $[l, r]$ for each node in 1-dimension, s.t., for any pair of nodes (u, v) if $u \rightsquigarrow v$ then v's label interval is completely contained within u's label interval.
- The above observation does not hold for an acylic graph which is *not* a tree.

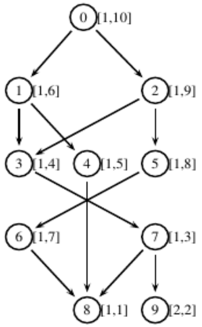
GRAIL



(a) Tree

Figure taken from Grail, PVLDB 2010.

GRAIL



(b) DAG: Single Interval

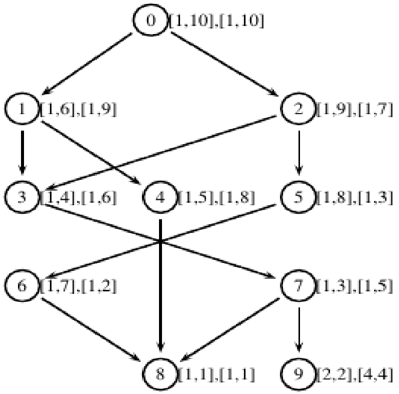
Exceptions: (1, 2), (2, 4), (3, 4), (4, 7) etc. Exercise: compute all the exceptions from this node labeling scheme.

Figure taken from Grail, PVLDB 2010.

GRAIL

- Aim is to cover as many pairs of vertices of the graph as possible by *hyperdimensional* labels, because 1-dimensional labels are insufficient to report the true reachability, i.e., they may generate *false positives* as seen in the example before.
- For generating hyperdimensional labels, consider k different spanning trees over the given directed acyclic graph, and for each spanning tree, generate 1-dimensional labels for each node.
- At the end combine these k labels for each node from each spanning tree.
- A node v is **not** reachable from u, if for any of these k label intervals of type $[l_v^i, r_v^i], [l_u^i, r_u^i], 1 \leq i \leq k$ v's label is not contained within u's label. That is to say: with GRAIL, we will certainly know if $u \not\rightsquigarrow v$ by just checking the interval labels of u, v.

GRAIL



(c) DAG: Multiple Intervals

Figure taken from Grail, PVLDB 2010.

GRAIL

- If the k interval labels of u, v indicate that $u \rightsquigarrow v$, then we have to do a DFS walk over the original graph beginning from u to check if we actually reach v, due to the possibility of *false positives*.
- **Key obervation:** If we consider large enough DFS trees over the given DAG and generate interval labels for each of these trees, then we will most likely eliminate all false positives. **Open question:** can we come up with an upper bound $\bar{k}$ such that interval labeling of $\bar{k}$-dimension will $\underline{\text{always}}$ eliminate any false positives?
- In practice k is kept fixed and false positives are eliminated at the run-time.
- **Optimization:** While doing a DFS walk on the graph from u, if you are at vertex w, and if v's interval label is outside w's, you can ignore that branch of DFS since $w \not\rightsquigarrow v$.

Partitioned Word-Aligned Hybrid compression [SIGMOD 2011]

- Merge strongly connected components.
- Sort all the vertices in the resulting DAG using a *topological sort* – assign level l_v to each node $v \in V_{dag}$ such that $l_v =$ (longest possible path from a node u to v, s.t., $indegree(u) = \phi$).
- Do a *reverse* DFS walk over this DAG considering the topological sort.
- **Key observation:** When you do a reverse DFS walk on a DAG, vertices that are adjacent in the topological sort tend to cluster together in the adjacency list – hence techniques like *run-length-encoding* can be applied on them.
- While doing a reverse DFS walk to compute reachability, compress contiguous sequence of vertices using compressed bitvectors – bitvector compression favours contiguous lengths of 1s and 0s.

Partitioned Word-Aligned Hybrid compression

- Computing the reachability of a any given intermediate node w, is nothing but doing a bitwise OR of the compressed interval lists of all its children and adding the children themselves to the reachability bitvector, and then applying compression on them.
- Word-Aligned Hybrid compression scheme has limitations, hence authors introduce Partitioned Word-Aligned Hybrid (PWAH) scheme.
- The generated index is complete, and saves significant amount of memory compared to approaches that store interval lists without any compression.

Next Class

In the next class we will mainly review other reachability computation methods such as IP-labeling [SIGMOD 2014], TF-folding [SIGMOD 2013] and some others as Chain-cover, Path-tree etc (time permitted).