On Graph Query Optimization in Large Networks

Peixiang Zhao, Jiawei Han

Department of Computer Science
University of Illinois at Urbana-Champaign

pzhao4@illinois.edu, hanj@cs.uiuc.edu

September 14th, 2010

Outline

1. Introduction
2. The Pattern-based Graph Indexing Framework
3. SPath: Graph Indexing on Large Networks
4. Graph Query Processing and Optimization
5. Experimental Evaluation
6. Conclusion

Introduction

- The burgeoning size and heterogeneity of networks call for effective **graph query processing** methods in a diverse range of applications:
  1. Bioinformatics and Cheminformatics
  2. Social Networks and Communication Networks
  3. Software Systems

Graph Query

Given a network G and a query graph Q, the graph query problem is to find as output all distinct matchings of Q in G.

- The graph query problem is **hard**
  1. Subgraph isomorphism checking is proven to be NP-complete
  2. The heterogeneity and sheer size of networks hinder a direct application of well-known graph matching methods

A Running Example

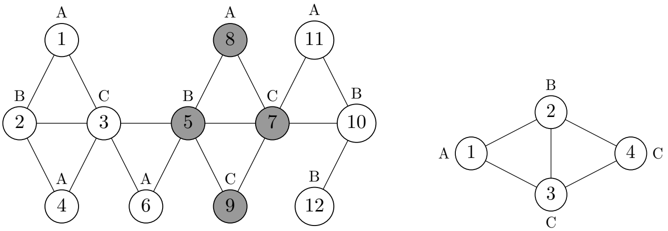

Figure: A Network G and a Query Graph Q

Introduction

- **Motivation**: Can we take advantage of well-studied database indexing and query optimization techniques to address the graph query problem on large networks?
- **SPath**:
  1. Indexes **neighborhood signatures** of vertices in the network, which maintains decomposed shortest path information within vertex vicinity
     1. Space-efficient
     2. Effective search space pruning ability
     3. High scalability in large networks
  2. Boosts graph query processing from vertex-at-a-time to **path-at-a-time**

The Baseline Algorithm

Exploring a tree-structured search space by considering **all possible vertex-to-vertex correspondences** from Q to G

Matching Candidate

$\forall v \in V(Q)$, the matching candidates of v is a set $C(v)$ of vertices in G bearing the same vertex label with v, i.e., $C(v) = \{u | l(u) = l'(v), u \in V(G)\}$, where l and l' are vertex labeling functions for G and Q, respectively.

- Total search space size: $\prod_{i=1}^{N} |C(v_i)|$
- Worst-case time complexity: $O(M^N)$ (M and N: the sizes of G and Q, respectively)

The Pattern Based Graph Indexing Framework

Objective: to reduce the search space size $\prod_{i=1}^{N} |C(v_i)|$

1. Minimize the number of one-on-one correspondence checkings, i.e, **min** N;
   - **Vertex-at-a-time**: $N = |V(Q)|$
   - **Pattern-at-a-time**: $N = k$, if a set of structural patterns $p_1, p_2, \ldots, p_k \subseteq Q$ $(k < N)$ is indexed
2. Minimize for each vertex in the graph query its matching candidates, i.e., **min** $|C(v_i)|$
   - It is unnecessary to check every vertex in $C(v_i)$!
   - For $v_i \in V(Q)$, we consider a **neighborhood induced subgraph** of Q, $G^k_{v_i}$, which contains all vertices (and induced edges) within k hops away from v_i

The Pattern Based Graph Indexing Framework

Theorem

If $Q \subseteq G$ w.r.t. a subgraph isomorphism matching f, for any structural pattern $p \subseteq G^k_{v_i}, v_i \in V(Q)$, there must be a matching pattern, denoted as $f(p) \subseteq G$, s.t. $f(p) \subseteq G^k_{f(v_i)}, f(v_i) \in V(G)$. □

- If structural patterns in the k-neighborhood subgraphs are indexed in advance, false positives in $C(v_i)$ can be pre-pruned, such that $|C(v_i)|$ is reduced

By extracting and indexing structural patterns within the k-neighborhood subgraphs, can we **achieve both objectives!**

The Pattern Based Graph Indexing Framework

Question

Among different kinds of structural patterns, which one (or ones) is most suitable for graph indexing on large networks?

The *graph indexing cost*, C, can be formulated as a combination of

1. The pattern selection cost C_s in G
2. The pattern selection cost C_s in Q
3. The pattern pruning cost of Q

The Graph Indexing Cost

$$C = (|V(G)| * n + |V(Q)| * n') * C_s + \frac{|V(Q)| * |V(G)| * n' * C_p}{|\Sigma|}$$

n and n' are the number of structural patterns in the k-neighborhood subgraph of vertices in G and Q, respectively

The Pattern Based Graph Indexing Framework

- We evaluate three different patterns, i.e., paths, trees and graphs for indexing

Cost	$n(n')$	C_s	C_p
Path	exponential	linear time	linear time
Tree	exponential	linear time	polynomial time
Graph	exponential	linear time	NP-complete

- **Paths** excel trees and graphs for indexing on large networks
  1. **Shortest paths** are further selected and decomposed into a distance-wise structure, **SPath**, as a high-performance graph indexing mechanism on large networks
  2. During graph query processing, decomposed shortest paths in SPath are reconstructed and joined for query optimization

k-DISTANCE SET

Given $u \in V(G)$, and a nonnegative distance k, the k-distance set of u, $S_k(u)$, is defined as

$$S_k(u) = \{S_k^l(u) | l \in \Sigma\} \backslash \{\emptyset\}$$

NEIGHBORHOOD SIGNATURE

Given $u \in V(G)$, and a nonnegative neighborhood scope k_0, the *neighborhood signature* of u, denoted as $NS(u)$, is defined as

$$NS(u) = \{S_k(u) | k \leq k_0\}$$

- All shortest path information in the k_0-neighborhood subgraph $G_u^{k_0}$ of u is (indirectly) encoded in the neighborhood signature, $NS(u)$

A Running Example

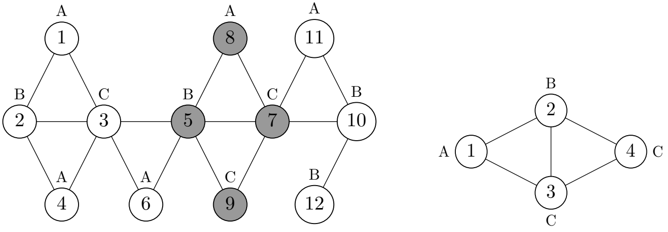

Figure: A Network G and a Graph Query Q

Example (Neighborhood Signature)

If the neighborhood scope k_0 is set 2, the neighborhood signature of $u_1 \in G$,
$NS(u_1) = \{\{A : \{1\}\}, \{B : \{2\}, C : \{3\}\}, \{A : \{4, 6\}, B : \{5\}\}\}$;
The neighborhood signature of $v_1 \in Q$,
$NS(v_1) = \{\{A : \{1\}\}, \{B : \{2\}, C : \{3\}\}, \{C : \{4\}\}\}$

SPath

NS CONTAINMENT

Given $u \in V(G)$ and $v \in V(Q)$, $NS(v)$ is contained in $NS(u)$, denoted as $NS(v) \sqsubseteq NS(u)$, if $\forall k \leq k_0$, $\forall l \in \Sigma$, $|\bigcup_{k \leq k_0} S_k^l(v)| \leq |\bigcup_{k \leq k_0} S_k^l(u)|$

Theorem

Given a network G and a graph query Q, if Q is subgraph-isomorphic to G w.r.t. f, i.e., $Q \subseteq G$, then $\forall v \in V(Q), NS(v) \sqsubseteq NS(f(v))$, where $f(v) \in V(G)$

- if $NS(v)$ is not contained in $NS(u)$, u is a false positive and can be safely pruned from v's matching candidates $C(v)$. Therefore, the search space size $|C(v)|$ is reduced

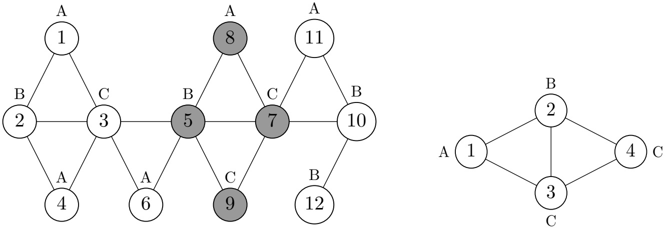

Figure: A Network G and a Graph Query Q

Example (NS Containment Pruning)

Based on NS pruning, the search space can be pruned for $C(v_1)$ from $\{u_1, u_4, u_6, u_8, u_{11}\}$ to $\{u_6, u_8, u_{11}\}$, for $C(v_2)$ from $\{u_2, u_5, u_{10}, u_{12}\}$ to $\{u_5\}$, for $C(v_3)$ from $\{u_3, u_7, u_9\}$ to $\{u_7\}$, and for $C(v_4)$ from $\{u_3, u_7, u_9\}$ to $\{u_7, u_9\}$. The total search space size has been reduced from 180 to 6

SPath Implementation

- SPath, maintains the neighborhood signature for each vertex of the network G
  1. **Global Lookup Table** $\mathcal{H} : l^* \rightarrow \{u | l(u) = l^*\}, l^* \in \Sigma$
     - Given a vertex v in the query graph, its matching candidates $C(v) = \mathcal{H}(l(v))$;
  2. **Histogram:** $|S_k^l(u)|$ for $0 < k \leq k_0$ in the neighborhood signature
  3. **ID-List:** $S_k^l(u), u \in V(G)$
- Index construction cost:
  - **Time:** $O(|V(G)| * |E(G)|)$
  - **Space:** $O(|V(G)| + |\Sigma| + k_0|\Sigma||V(G)|)$

A Running Example

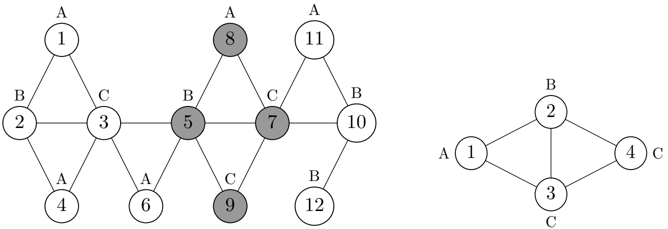

Figure: A Network G and a Graph Query Q

label	vid				
A	1	4	6	8	11
B	2	5	10	12	
C	3	7	9		

Histogram / **ID-List**

distance	label	count	vid		
1	A	3	1	4	6
	B	2	2	5	
2	A	1	8		
	C	2	7	9	

Figure: The Global Lookup Table $\mathcal{H}$ and the Histogram and ID-List of $NS(u_3)$, $u_3 \in V(G)$ ($k_0 = 2$)

Graph Query Processing and Optimization

1. **Query Decomposition:** To decompose the query graph Q into a set of indexed shortest paths
2. **Path Selection and Join:** To choose an optimal set of paths to "*recover*" the query graph
   - $\forall e \in E(Q)$, there should exist at least one selected shortest path p, such that $e \in p$
   - The set of shortest paths should be cost-effective and help reconstruct the query Q in an efficient way
3. **Path Instantiation:** To instantiate the path for exact matching and cross-check the path join predicates

Path Selection and Join

- We consider two objectives in the query plan optimizer for path selection and join
  1. To choose the smallest set of shortest paths which can cover the query
     - Reduced to the NP-complete *set-cover* problem
  2. To choose shortest paths with good selectivity, such that the total search space can be minimized during real graph matching
- **Selectivity** of a path p

$$sel(p) = \frac{\psi(l)}{\prod_{v \in V(p)} |C'(v)|}$$

  - A greedy approach to always picking the edge-disjoint path with highest selectivity first

Experimental Evaluation

- SPath *v.s.* GraphQL [SIGMOD'08]
- One real data set (memory resident)
    - Yeast Protein Interaction Network
- A series of synthetic data set (disk resident)
    - G-MAT Synthetic Graph Generator
- Queries to be Examined
    1. Clique query
    2. Path query
    3. General subgraph query

- The yeast protein interaction network
  - $3,112$ vertices
  - $12,519$ edges
  - 183 GO terms as vertex labels

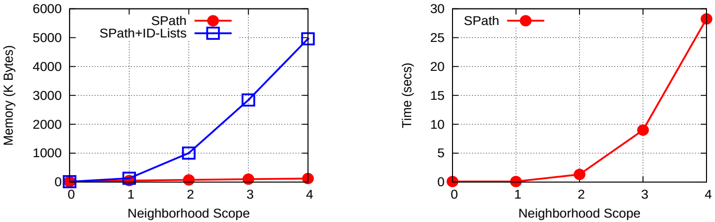

Figure: Index Construction Cost for SPath

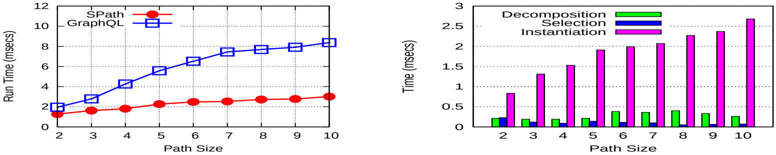

Figure: Query Response Time for Path Queries

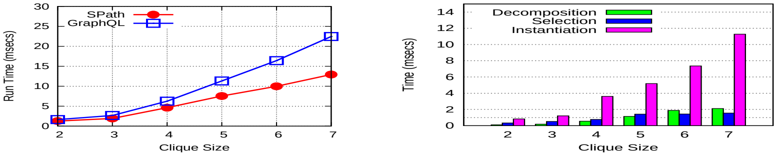

Figure: Query Response Time for Clique Queries

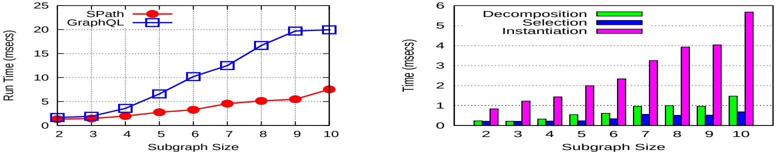

Figure: Query Response Time for Subgraph Queries

Synthetic Disk-resident Network: Index Construction

- A series of disk-resident synthetic graphs are generated based on R-MAT model, which follows power-law in- and out-degree distribution
  - $|V(G) = 500,000; 1,000,000; 1,500,000$ and $2,000,000$
  - $|E(G)| = 5 * |V(G)|$
  - $|\Sigma| = 1\% * |V(G)|$

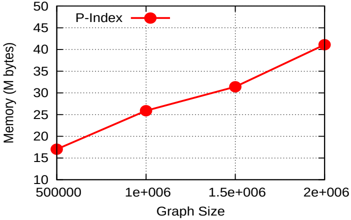

Figure: Index Construction Cost for SPath

Synthetic Disk-resident Network: Subgraph Query

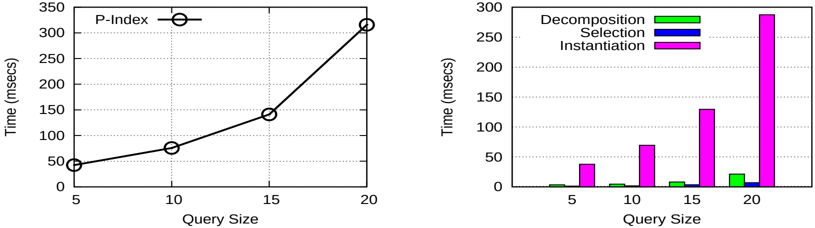

Figure: Query Response Time for Subgraph Queries in the Synthetic Graph

Conclusion

1. **Graph queries** are frequently issued on large networks
   - Existing data models, query languages and access methods no longer fit well in the large networks to support graph query processing effectively
2. **Graph indexing** plays a key role in facilitating graph query processing
   - Different structural patterns are evaluated based on a cost-sensitive model and shortest paths are chosen as good indexing features in large networks
3. **SPath**
   - Revolutionizes the way of graph query processing from *vertex-at-a-time* to *path-at-a-time*
   - Exhibits good scalability and satisfactory query performance

Thank you