I/O Efficient Algorithms for Exact Distance Queries on Disk-Resident Dynamic Graphs

Yishi Lin, Xiaowei Chen, John C.S. Lui
The Chinese University of Hong Kong
Distance on Graphs

Distance is fundamental
- social network analysis
- communication networks
- road networks
- biological networks
- ...

Real graphs
- large
- dynamic

Trade off
- indexing cost / query efficiency

Erdős number: the collaborative distance between mathematician Paul Erdős and another person.
The co-author path is obtained from http://academic.research.microsoft.com/VisualExplorer#1022791&1112639
Our Focus

Given a **dynamic disk-resident** graph $G = (V, E)$

1. Construct & **incrementally update** an index
2. Answer exact distance $d_G(s, t)$ in the latest graph

**Focus of this work**
Previous Methods for exact distance queries

No method for dynamic disk-based graphs!

1. **Bound** on label size for scale-free networks
2. **Disk-based** canonical labeling construction method

- 2-Hop Labeling
  - [Cohen et al. SODA’02]
- Hierarchical Hub Labeling
  - [Abraham et al., ESA’12]
- Hop Doubling Labeling
  - [Jiang et al., VLDB’14]
- Pruned Landmark Labeling
  - [Akiba et al., SIGMOD’13]
- Dynamic PLL
  - [Akiba et al., WWW’14]

Canonical Labeling
- a special 2-hop labeling

State-of-the-art **memory-based** method to **construct** and **update** the canonical labeling
Canonical Labeling for distance queries

Data structure (given a ranking \( r \))

- In-label and out-label for each node \( u \)
- \( L_{out}(u) = \{(v_1, d_1), (v_2, d_2), \ldots \} \), \( d_i = d(u, v_i) \)
- \((v, d) \in L_{out}(u) \iff v \) has the highest rank among all shortest paths from \( u \) to \( v \)
- \( L_{in}(u) = \{(w_1, d_1), (w_2, d_2), \ldots \} \), \( d_i = d(w_i, u) \)
- \((v, d) \in L_{in}(u) \iff v \) has the highest rank among all shortest paths from \( v \) to \( u \)

Larger nodes have higher ranks.

\[
L_{out}(s) = \{(s, 0), (v_0, 1), (v_1, 1), (v_2, 2)\}
L_{in}(t) = \{(t, 0), (v_1, 2), (v_2, 1), (v_3, 1)\}
\]

Other notations:
\((u \rightarrow v, d) \iff (u, d) \in L_{in}(v), (u \rightarrow v, d) \iff (v, d) \in L_{out}(u)\)
\((u \rightarrow v, d) \iff (u, d) \in L_{in}(v) \text{ or } (v, d) \in L_{out}(u)\)
Canonical Labeling for distance queries

Data structure
- \( L_{out}(u) = \{(v_1, d_1), (v_2, d_2), \ldots \}, d_i = d(u, v_i) \)
- \( L_{in}(u) = \{(w_1, d_1), (w_2, d_2), \ldots \}, d_i = d(w_i, u) \)

Query algorithm \( \text{QUERY}(L, s, t) \)
- \( \min\{d_1 + d_2 | (w, d_1) \in L_{out}(s), (w, d_2) \in L_{in}(t)\} \)
- 2-hop paths using labels

Properties
- **Correctness**: Distance queries are answered correctly.
- **Minimum**: There is no non-necessary entry.

Incremental Maintenance Objective:
Given a canonical labeling \( L^{t-1} \) for graph \( G_{t-1} \) based on rank \( r \), update \( L^{t-1} \) and obtain an \( r \)-based canonical labeling \( L \) for the latest graph \( G_t \).
Contribution

We consider disk-resident dynamic graphs.

Update methods
- Single edge update algorithm
- Batch update algorithm

Latest distance query
- Answer exact distance queries with the outdated labeling and new edges (without update)
Single Edge Update (Contribution 1)

Two phases
- **Patch generation**: to answer distance queries correctly
- **Patch merge**: to remove non-necessary entries

\[ G_t = G_{t-1} \cup e_{xy} \]

\[ L^t \subseteq L^{t-1} \cup P \]

2-hop labeling of \( G_t \)
**Single Edge Update: Patch Generation**

**Patch Generation** (focus on the patch $P_{in}$ of $L_{in}^{t-1}$)
- New edge: $e_{xy}$.
- (when) $P_{in}(v) \neq \emptyset \Rightarrow$ distance from $x$ to $v$ decreases ($v$ is a “sink node”)
- (how) $P_{in}(v)$ should contain $(u, d) \Rightarrow (u, d_{t-1}(u, x)) \in L_{in}^{t-1}(x)$

---

**BFS method to generate entries in the patch $P_{in}$**

---

Visit node 6
- node 6 is a sink node
- add $(5,1)$ and $(4,2)$ to $P_{in}(6)$

Visit node 2, not a sink, stop

Visit node 3, ...

... 

I/O cost: $O(|\text{sink nodes and their outneighbors}|)$
Single Edge Update: Patch Merge

Patch Merge

- **Goal**: $L^t = \text{merge}(L^{t-1}, P)$
- Although $P$ is minimum, $L^{t-1} \cup P$ may not be minimum.
- **Pruning rule**: we remove an entry $(u \to v, d)$ if there exist $(u \to w, d_1)$ and $(w \to v, d_2)$ so that $d_1 > 0, d_2 > 0$ and $d_1 + d_2 \leq d$.
- Standard pruning rule for the canonical labeling.
- **Merge with pruning**: using block-nested loops
- **I/O cost**: $O \left( \left[ |L^{t-1}| + |P| \right] / M \right) \cdot \left[ |L^{t-1}| + |P| \right] / B$ 

Refinements for the Single edge update method

- lazy patch merge, label prefetch

\[ d \geq d_1 + d_2 \]

Larger nodes have higher ranks.
Batch Update (Contribution 2)

Motivation

Graph $G_{t-1}$
A set of new edges $E_{new}$
$G = G_{t-1} + E_{new}$

$L^{t-1} (r$-based)$

Single edge update method
I/O cost $\sim |E_{new}|$

Batch update method

$L^t (r$-based)$
Batch Update: High Level Ideas

**Iteratively** generate entries in $L$
- each iteration = candidate generation + candidate merge
- utilize entries in $L^{t-1}$

**Candidate generation** (to correctly answer distance queries)
- $L_{cand}$: candidates generated by “concatenate” existing entries
- The 0-th iteration: $L_{cand} = \text{new edges}$.

![Diagram of candidate generation and merge]

**Candidate merge**
- $L = \text{merge} (L, L_{cand})$ (≈ the patch merge phase for the single edge update method)

**I/O Cost** per iteration: $O \left( \left\lceil \frac{|L| + |L_{cand}|}{M} \right\rceil \cdot \left\lceil \frac{|L| + |L_{cand}|}{B} \right\rceil \right)$ (Lemma 7)
Can we answer queries before the update finishes?

Yes! 😊

Latest Distance Query

- We could answer distance queries with the outdated labeling.
- We need not wait until the update finishes.
- We need not update the labeling, if we do not want to.
- It works for all 2-hop labeling, not only for the canonical labeling.
Latest Distance Query: framework

Preprocess

Query graph $G_Q$
(in main memory)

Update-related edges

Dijkstra

Query $d_G(s, t)$

$O(1)\text{I/Os}$

New edges $E_{new}$

Query graph $G_Q$

Query-related edges

$\Delta^{t-1}$

Query $d_G(s, t)$

$= d_{G_Q}(s, t)$
Latest Distance Query: Query Graph $G_Q$

**Query-related:** edge $s \rightarrow t$ with distance $\text{QUERY}(L^{t-1}, s, t)$

$$d_{G_Q}(s, t) = d_G(s, t)$$

**Update-related:**
- Edges in $E_{new}$.
- If $v \in V(E_{new})$, every entry in $L_{out}^{t-1}(v)$ and $L_{in}^{t-1}(v)$ corresponds to an edge in $G_Q$.

**Query-related:**
- $\forall v \in V_{new}$, edge $s \rightarrow v$, distance $\text{QUERY}(L^{t-1}, s, v)$
- $\forall v \in V_{new}$, edge $v \rightarrow t$, distance $\text{QUERY}(L^{t-1}, v, t)$

$L^{t-1}$: 2-hop labeling for $G_{t-1}$ / $E_{new}$: new edges / $V(E_{new})$: endpoints of new edges
Experiments: Update Time

Remarks:
1. We treat all datasets as directed networks.
2. For Gowalla and Wiki-Talk, we randomly generate new edges. For Youtube, edges come with timestamps.
3. Experiments are conducted using 4GB memory on a Linux machine with Intel 3.20GHz CPU and 7200 RPM SATA hard disk.

Figure. Comparison among update methods and the reconstruction method.

(a) Gowalla
\(|V| = 197K, |E_{t-1}| = 1.9M\)
(b) Wiki-Talk
\(|V| = 2.4M, |E_{t-1}| = 5.0M\)
(c) Youtube
\(|V| = 3.2M, |E_{t-1}| = 9.4M\)
Experiments: Query Time

Figure. Results of query algorithm on real datasets.

Table. Real datasets.

Remarks:
1. For each dataset, we answer 5K random distance queries and report the average query time.
2. We clear the file system memory cache before answering each query. So we are actually measuring the worst case query time because every I/O request results in a physical I/O.
Conclusion

Distance queries of **disk-resident dynamic** graphs
- based on the *canonical labeling*

Contribution 1: **Single Edge Update** method

Contribution 2: **Batch Update** method

Contribution 3: **Latest Distance Query** method
- based on the outdated labeling

**Future work**
- Update methods of the *canonical labeling* for **memory-based / disk-based fully dynamic** graphs (both insertion and deletion of edges are allowed)
Thank you!