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

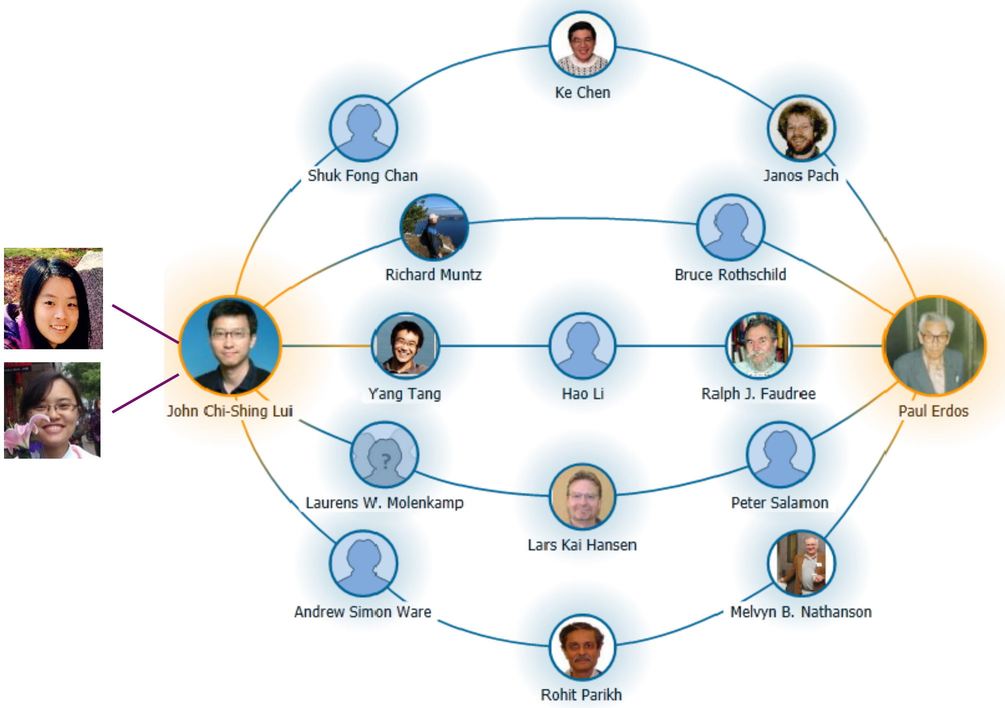
- social network analysis
- communication networks
- road networks
- biological networks
- …

Real graphs

- large
- dynamic

Trade off

- indexing cost / query efficiency



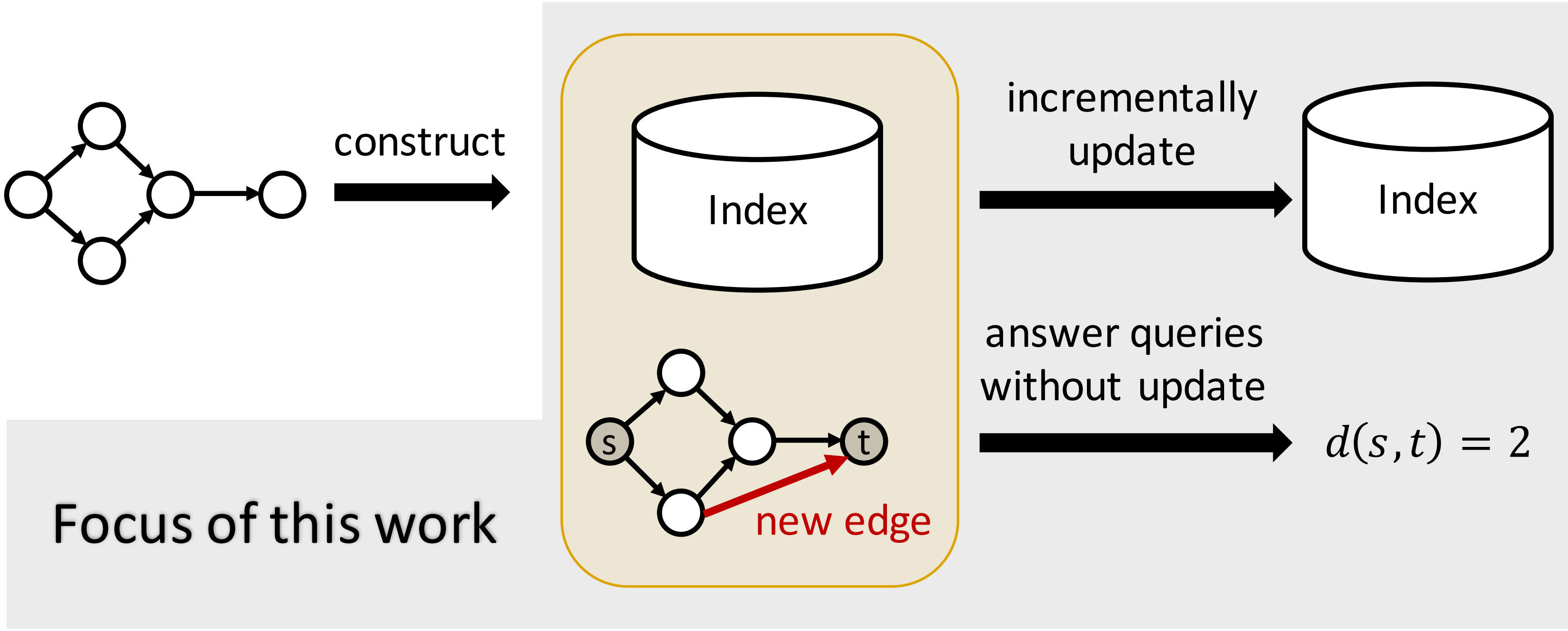
Erdős number: the collaborative distance between mathematician Paul Erdős and another person.

The co-author path is obtained from http://academic.research.microsoft.com/VisualExplorer#1022791&1112639

Our Focus

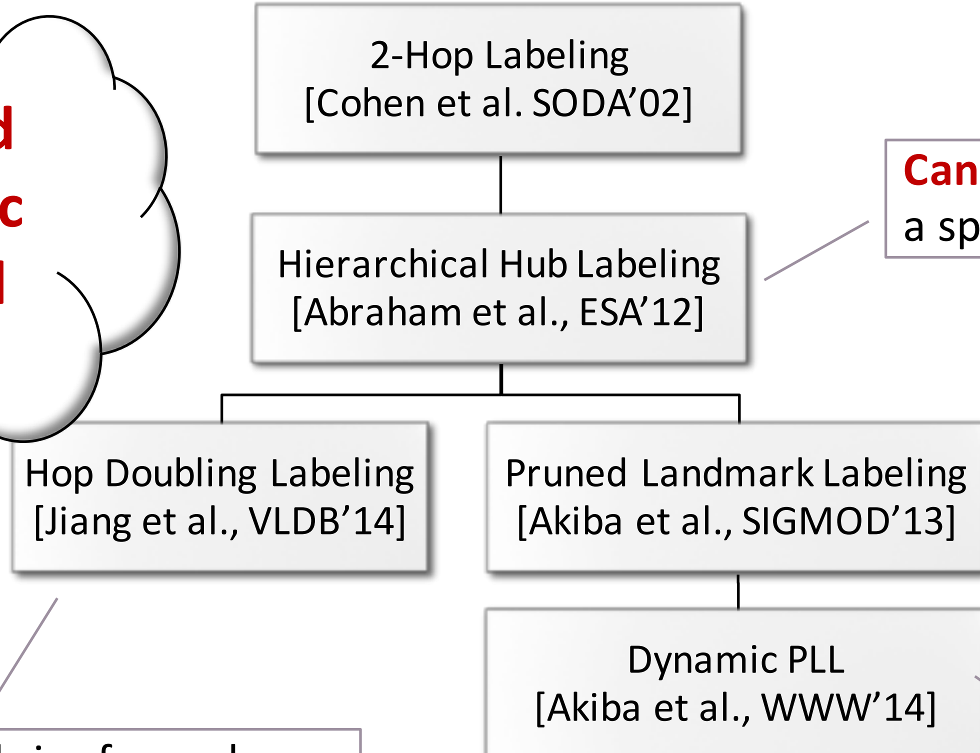
Given a **dynamic disk-resident** graph G=(V,E)

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



Previous Methods for exact distance queries

No method for dynamic disk-based graphs!



Canonical Labeling
a special 2-hop labeling

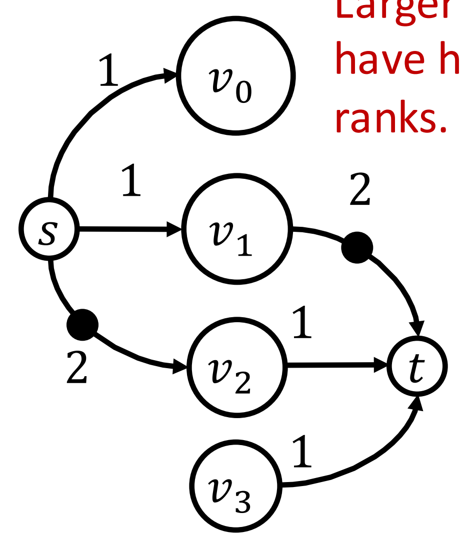
1. **Bound** on label size for scale-free networks
2. **Disk-based** canonical labeling construction method

State-of-the-art **memory-based** method to **construct** and incrementally **update** the canonical labeling

Canonical Labeling for distance queries

Data structure (given a ranking r)

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



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

Other notations:

$(u \to \underline{v}, d) \Leftrightarrow (u, d) \in L_{in}(v)$, $(\underline{u} \to v, d) \Leftrightarrow (v, d) \in L_{out}(u)$

$(u \to v, d) \Leftrightarrow (u, d) \in L_{in}(v)$ or $(v, d) \in L_{out}(u)$

Canonical Labeling for distance queries

Data structure

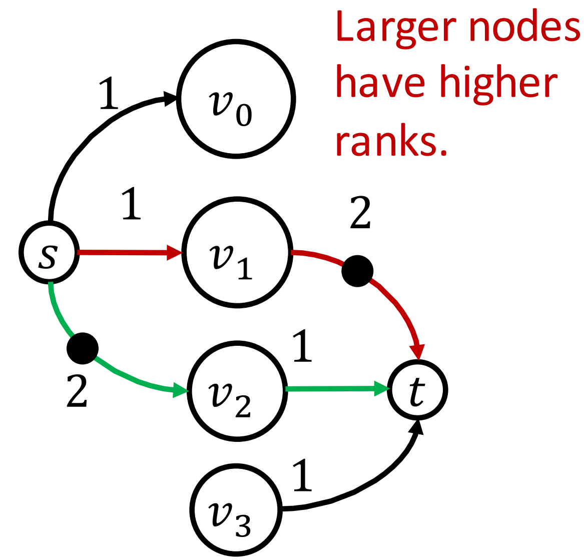
- $L_{out}(u) = \{(v_1, d_1), (v_2, d_2), \ldots\}, d_i = d(u, v_i)$
- $L_{in}(u) = \{(w_1, d_1), (w_2, d_2), \ldots\}, d_i = d(w_i, u)$

Query algorithm $QUERY(L, s, t)$

- $\min\{d_1 + d_2 | (w, d_1) \in L_{out}(s), (w, d_2) \in L_{in}(t)\}$
- 2-hop paths using labels

Properties

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



$L_{out}(s) = \{(s, 0), (v_0, 1), (v_1, 1), (v_2, 2)\}$

$L_{in}(t) = \{(t, 0), (v_1, 2), (v_2, 1), (v_3, 1)\}$

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

Contribution

We consider **disk-resident dynamic** graphs.

Update methods

- Single edge update algorithm
- Batch update algorithm

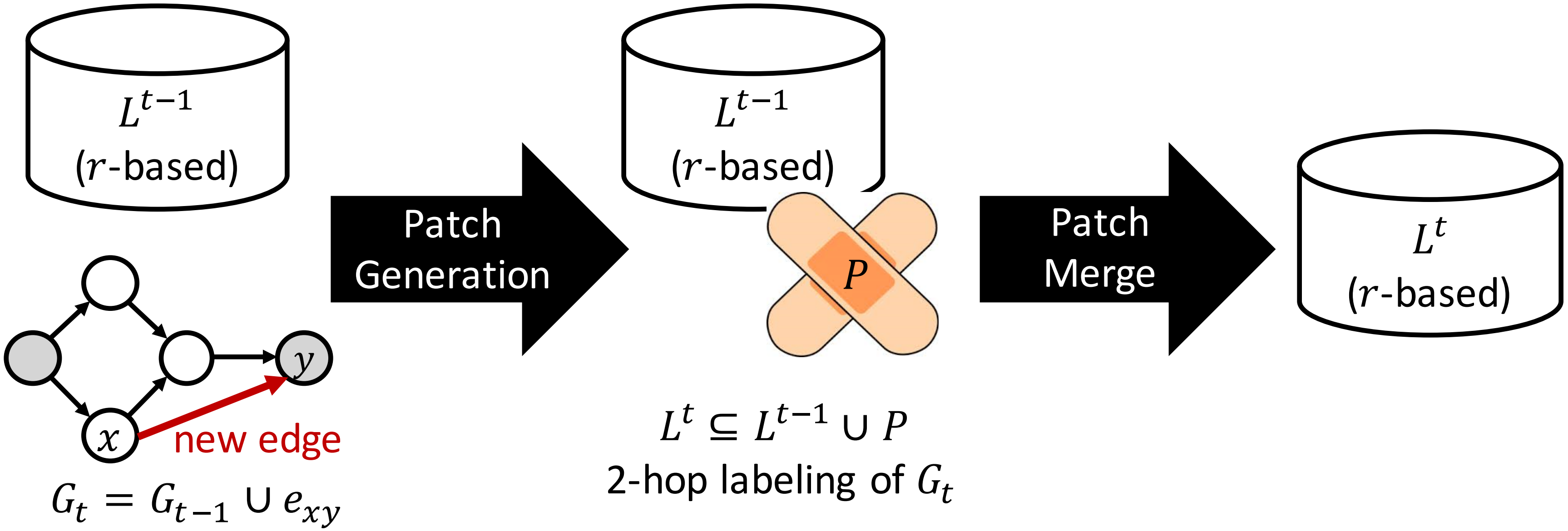
Latest distance query

- 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

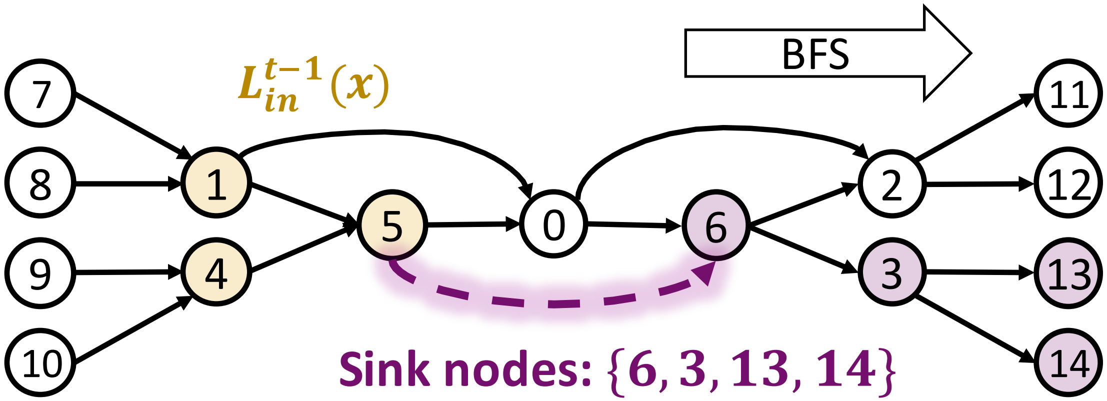


Single Edge Update: Patch Generation

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 $(u, d) \Rightarrow \left(u, d_{t-1}(u, x)\right) \in L_{in}^{t-1}(x)$

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



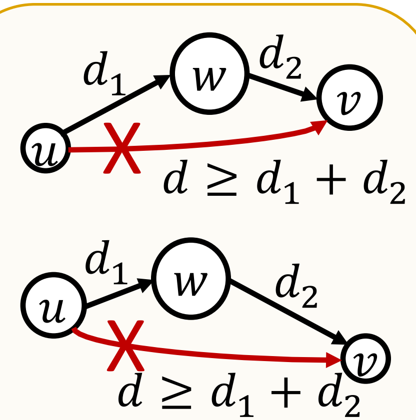
visit node 6
node 6 is a sink node
add $(5,1)$ and $(4,2)$ to $P_{in}(6)$
visit node 2, not a sink, stop
visit node 3, …
…

I/O cost: $O(|\textbf{sink nodes and their outneighbors}|)$

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 $(\underline{u} \to w, d_1)$ and $(w \to \underline{v}, d_2)$ 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**: $O\left(\left\lceil\frac{|L^{t-1}|+|P|}{M}\right\rceil \cdot \left\lceil\frac{|L^{t-1}|+|P|}{B}\right\rceil\right)$



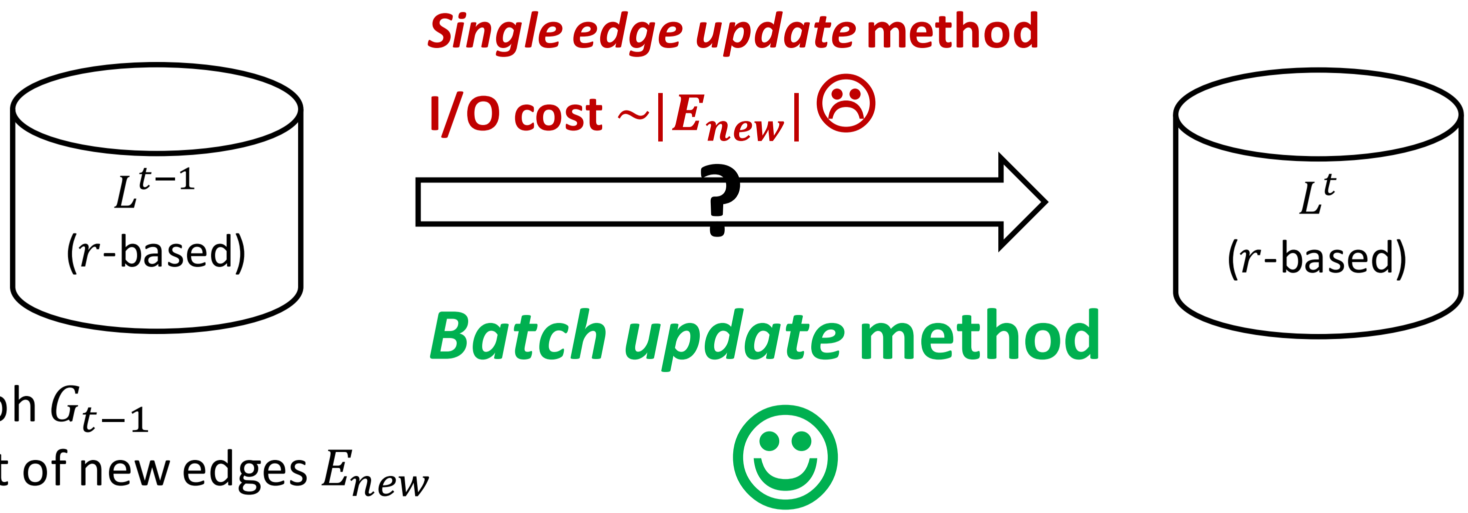
Larger nodes have higher ranks.

Refinements for the Single edge update method

- lazy patch merge, label prefetch

Batch Update (Contribution 2)

Motivation



Graph G_{t-1}

A set of new edges E_{new}

$G = G_{t-1} + E_{new}$

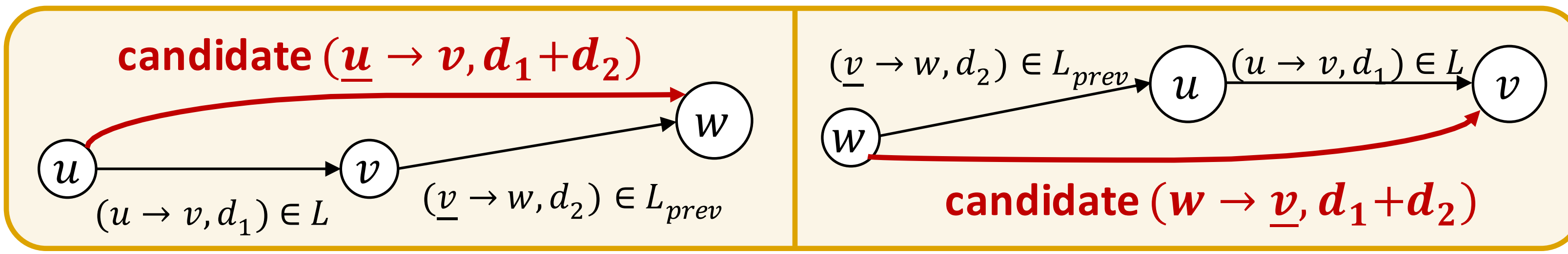
Batch Update: High Level Ideas

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: $L_{cand} := new\ edges$.



Candidate merge

- $L := merge(L, L_{cand})$ (≈ 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)

Latest Distance Query (Contribution 3)

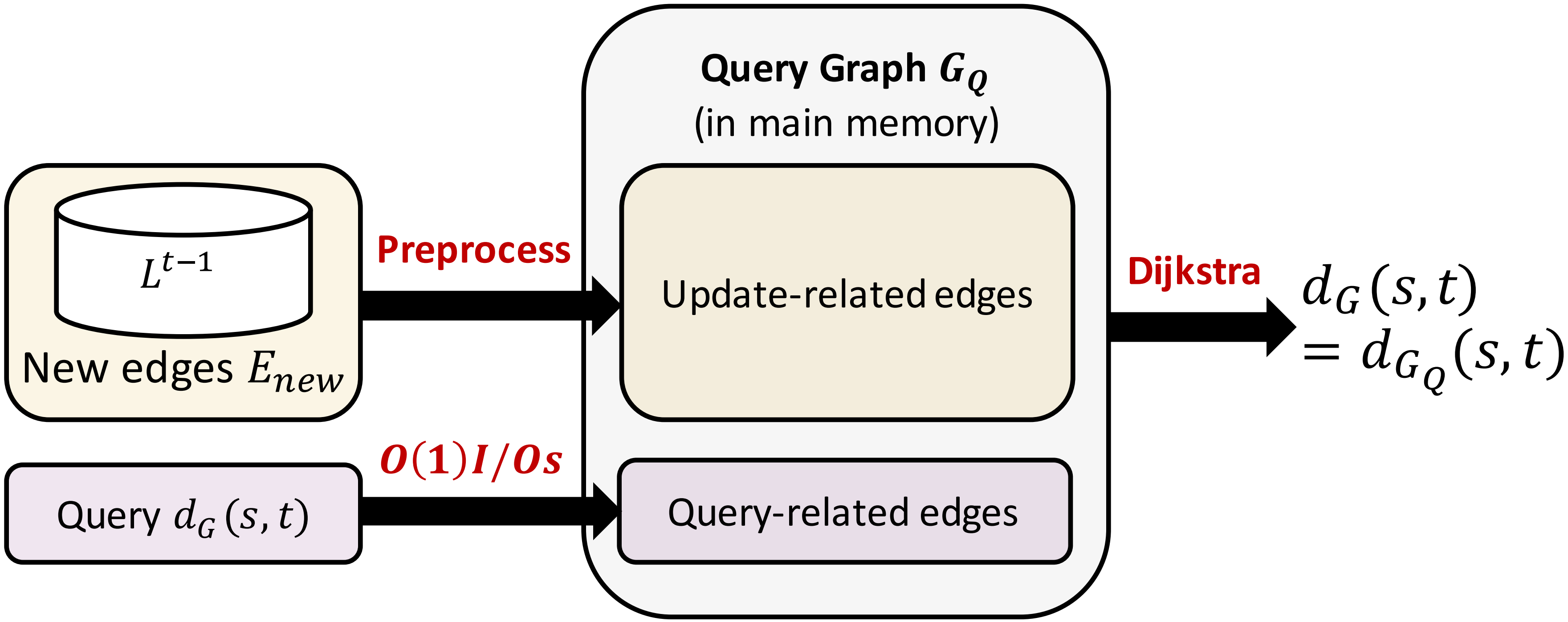
Can we answer queries before the update finishes?

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



Latest Distance Query: Query Graph G_Q

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

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

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.

$v \in V(E_{new})$

s

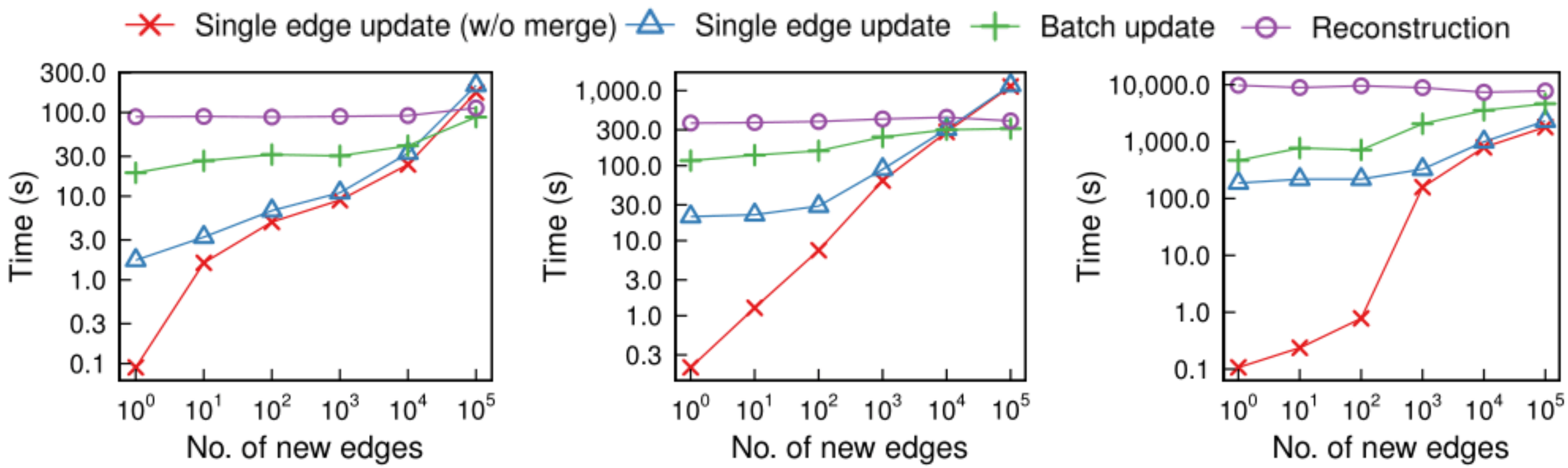
t

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

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

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

Experiments: Update Time



(a) *Gowalla*
$(|V| = 197K, |E_{t-1}| = 1.9M)$

(b) *Wiki-Talk*
$(|V| = 2.4M, |E_{t-1}| = 5.0M)$

(c) *Youtube*
$(|V| = 3.2M, |E_{t-1}| = 9.4M)$

Figure. Comparison among update methods and the reconstruction method.

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.

Experiments: Query Time

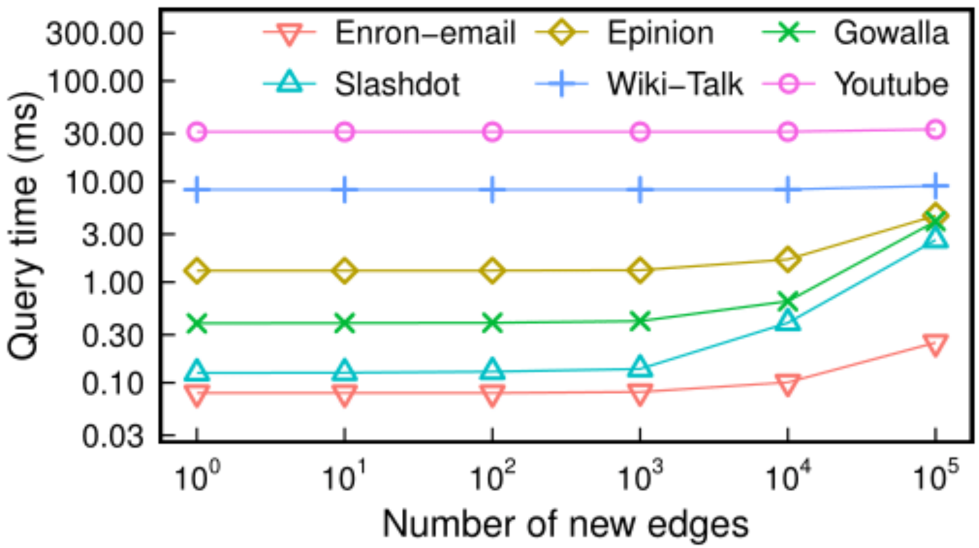


Figure. Results of query algorithm on real datasets.

Dataset	$\|V\|$	$\|E_{t-1}\|$
Youtube	$3.2M$	$9.4M$
Wiki-Talk	$2.4M$	$5.0M$
Epinion	$76K$	$509K$
Gowalla	$197K$	$1.9M$
Slashdot	$77K$	$905K$
Enron-email	$87K$	$160K$

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

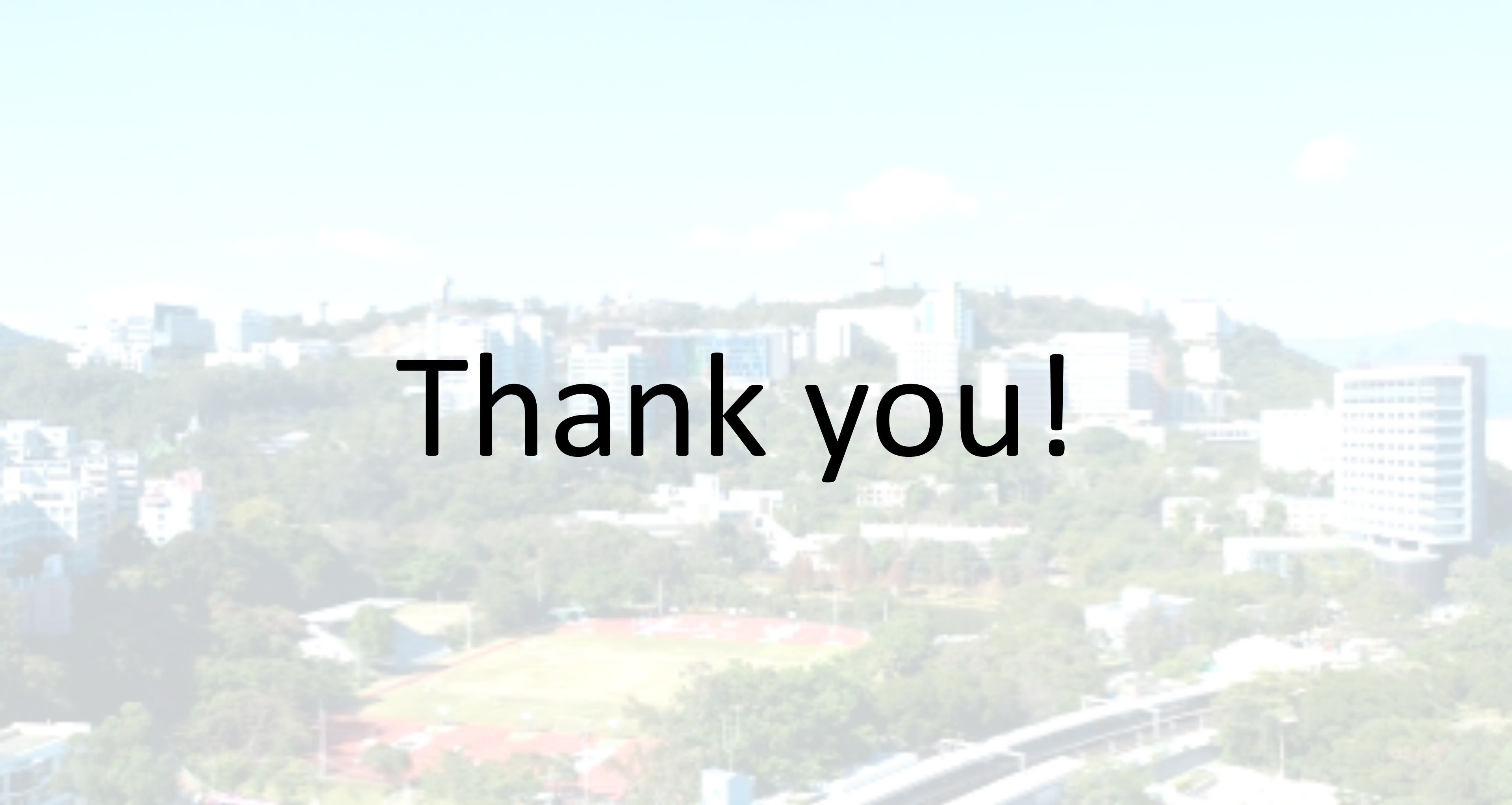
Contribution 2: **Batch Update** method

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!

香港中文大學
The Chinese University of Hong Kong