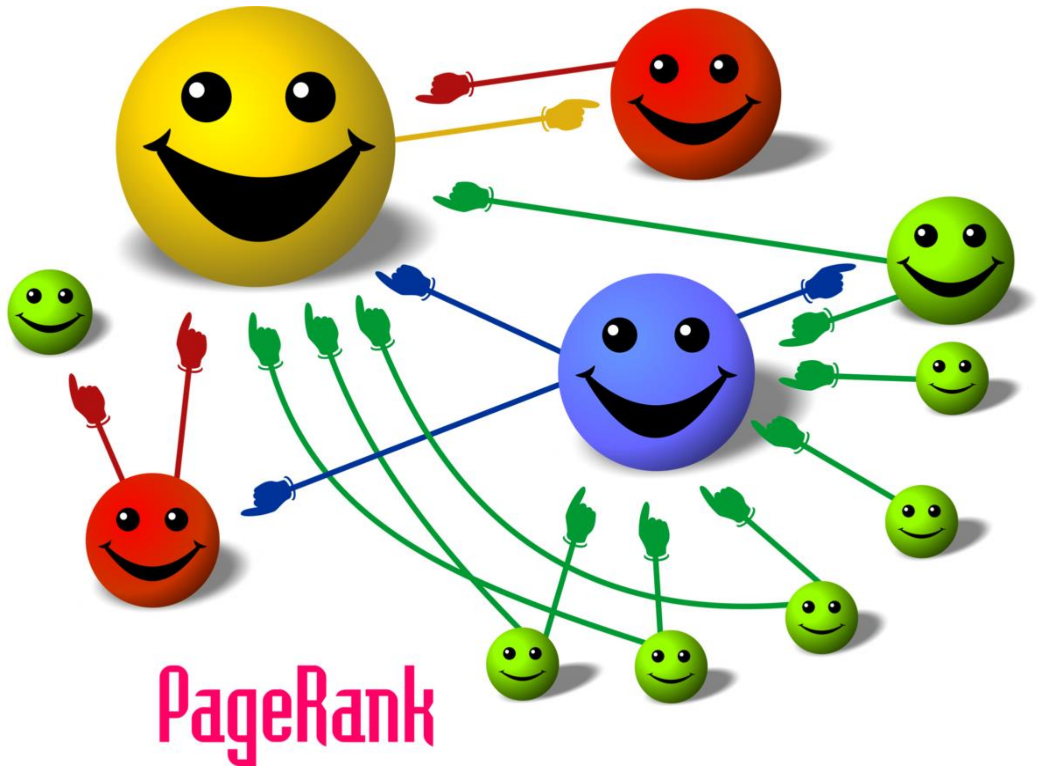
Fast Incremental PageRank on Dynamic Networks

Zexing Zhan
Wuhan University

What is PageRank?

- PageRank is an algorithm for measuring the centrality of nodes in a network.
- Each node's PageRank value depends on the number and quality of links to that node.

$$\pi_v = \alpha \sum_{u \in \boldsymbol{in}_v} \frac{\pi_u}{outdeg_u} + (1 - \alpha)$$



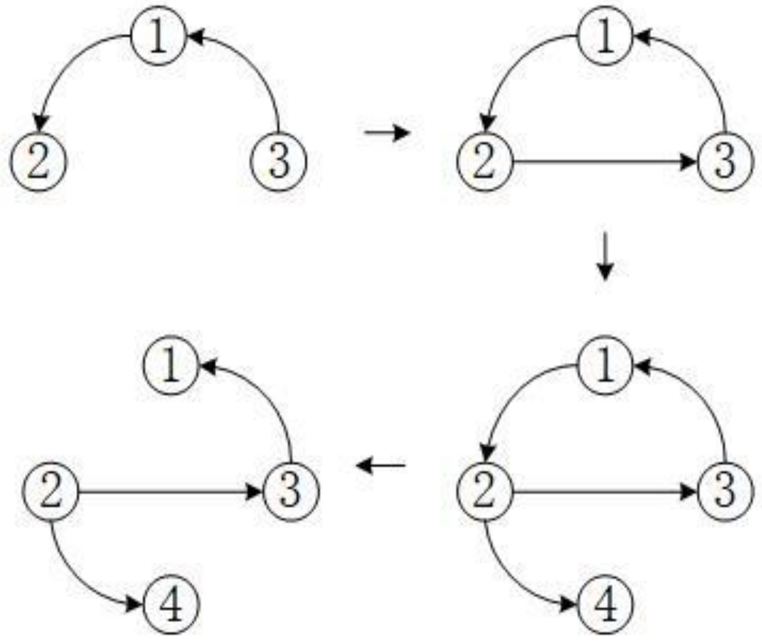
What is the Problem?

The PageRank algorithm works well on small static networks

But real-world networks are:

- dynamically changing
- of large scale.

e. g., according to data released in 2017, Facebook had 1.32 billion daily active users and these users sent tens of billions of messages every day.



An example of a dynamic network

How can we efficiently track the PageRank values of such real-world dynamic networks?

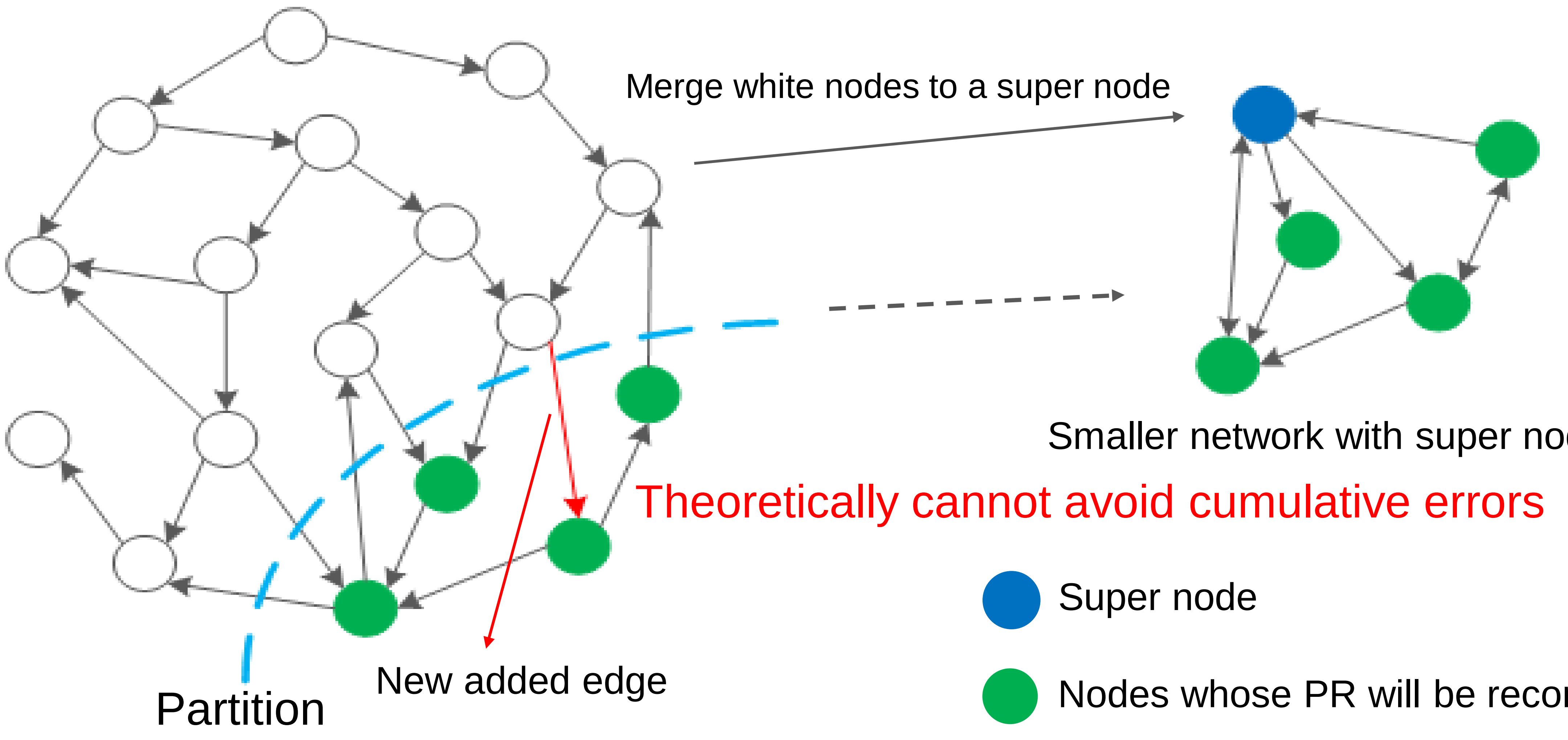
Relevant state of the art

Currently, there are two main categories of incremental PageRank algorithms:

- Partition and Aggregation algorithms
- Monte Carlo based algorithms.

Since Partition and Aggregation algorithms cannot avoid cumulative errors, our work focus on Monte Carlo based algorithms.

How do Partition and Aggregation algorithms work?


Merge white nodes to a super node
Smaller network with super node S
New added edge
Partition

Theoretically cannot avoid cumulative errors

Original Network

● Super node

● Nodes whose PR will be recomputed

○ Nodes whose PR will not be recomputed

How do Monte Carlo based algorithms work?

Approximating PageRank:

- First simulating exactly R random walks with reset probability ϵ starting from each node
- Then counting the number of times those random walks visits each node u, denoted as V_u
- Finally approximating the PageRank value of each node u by

$$\widetilde{\pi}_u = \frac{V_u}{nR/\epsilon}$$

where n is the number of nodes in a network.

How do Monte Carlo based algorithms work?

Updating PageRank:

- By storing all random walk segments, we can update PageRank without re-simulating all nR random walks.
- When an edge $e(u, w)$ is deleted or inserted, to update PageRank is to adjust those random walk segments accordingly.

Core problem: how many random walk segments are really affected, thus need to be adjusted?

Motivation and goals

Motivation:

Previous Monte Carlo based algorithms works only in a special case:

- No nodes are added or deleted
- No random walks revisit a same node (i.e. no loop in a network)

which however is not the truth for real-world networks.

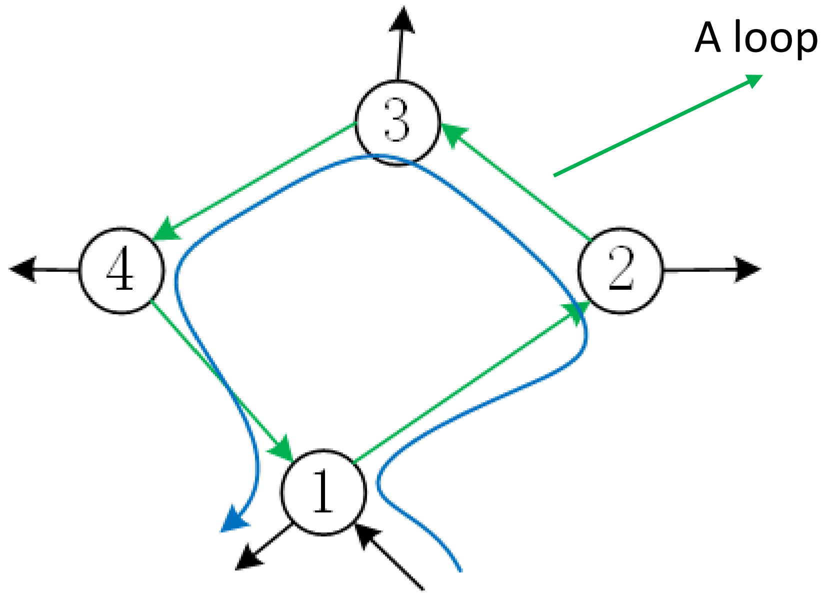
Our goals:

To develop an efficient Monte Carlo based PageRank tracking algorithm for real-world networks.

Difficulties

Loops in real-world networks make it difficult:

- (**main**) to determine how many random walk segments are affected?
- to properly adjust an affected random walk segments
- to manage all nR random walk segments efficiently.



A random walk visits node 1 twice

Key contribution: The Revisit Probability Model

For real-world networks, the main difficulty is to determine how many random walk segments are really affected by the edge modified.

To solve this issue, we proposed the revisit probability model.

Key contribution: The Revisit Probability Model

For a graph $G(t)$, we define the revisit probability of an edge $e(u,v)$ and a node u as follows:

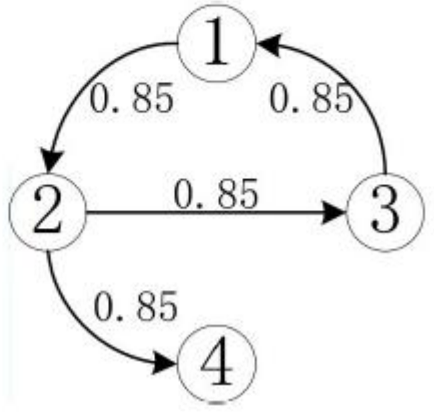
- $\boldsymbol{r_{uv}}$: The probability that a random walk passing through edge $e(u,v)$ revisits node u.
- $\boldsymbol{R_u}$: The expectation of its out edges' r_{uv} :

$$R_u(t) = \sum_{v \in \boldsymbol{out}_u(t)} \frac{1-\epsilon}{outdeg_u(t)} r_{uv}(t)$$

Key contribution: The Revisit Probability Model

the probability of all paths starting from $e(1,2)$ in $G(t)$ with $\epsilon = 0.15$

No.	path x	$\mathbb{P}(X = x)$	
1.	① → ②	0.15	0.15
2.	① → ② → ④	$0.85 \times \frac{1}{2}$	0.425
3.	① → ② → ③	$0.85 \times \frac{1}{2} \times 0.15$	0.06375
4.	① → ② → ③ → ①	$0.85 \times \frac{1}{2} \times 0.85$	0.36125
Sum.			1



graph G(t)

$r_{12}(t) = 0.36125$

$R\_1(t) = r\_12(t) * (1 - \epsilon) = 0.36215 * 0.85$

quite a high probability

It is complicated to compute it directly

An example of the revisit probability

Key contribution: The Revisit Probability Model

Denote $W_u(t)$ as the number of random walk segments visits node u, $V_u(t)$ as the total visit times. They can be expressed by:

$$V_u(t) = W_u(t)(1 + R_u^1(t) + R_u^2(t) + ...) = \frac{W_u(t)}{1 - R_u(t)}$$

So, for large and complex networks, we can estimate $R_u(t)$ as

$$\widetilde{R}_u(t) = 1 - \frac{W_u(t)}{V_u(t)}$$

Key contribution: The Revisit Probability Model

Observation 1 *When an edge $e(u, w)$ is modified at time $t + 1$, r_{uv} remains unchanged for any node $v \in \boldsymbol{out}_u(t)$ and $v \neq w$, where $\boldsymbol{out}_u(t)$ is the set of nodes that u pointing out to.*

Observation 2 *When an edge $e(u, w)$ is modified at time $t + 1$, walk count $W_u(t+1)$ remains unchanged but visit count $V_u(t+1)$ is affected, i.e., $W_u(t+1) = W_u(t)$ but $V_u(t + 1) \neq V_u(t)$.*

These two observations make it possible for accurate PageRank tracking

Key contribution: The Revisit Probability Model

Denote M_{t+1} as the actual number of random walk segments that need to be adjusted when adding an edge $e(u,w)$ to graph $G(t)$ at time t+1。 Through our model we finally get

$$M_{t+1} = \frac{W_u(t)}{outdeg_u(t+1) - outdeg_u(t) R_u(t)}$$

Result used by previous method is simply :

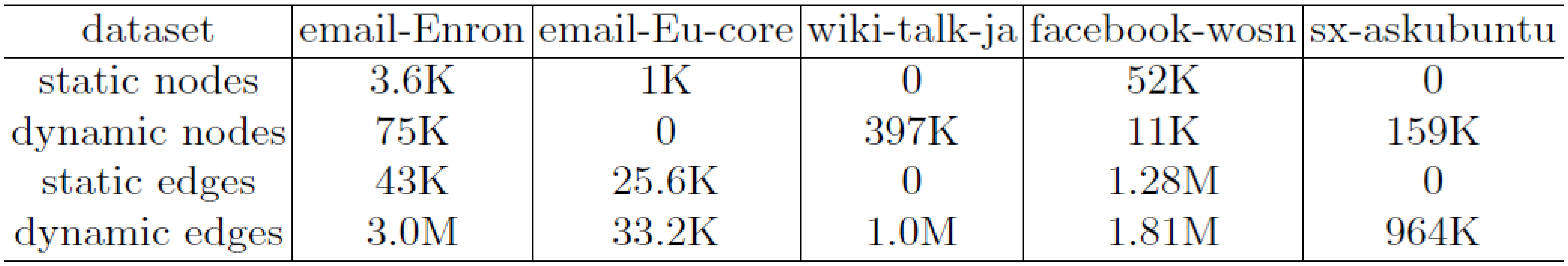
$$M_{t+1} = \frac{V_u(t)}{outdeg_u(t+1)}$$

If no random walk revisits a node, then $W_u(t) = V_u(t)$ and $R_u(t)$ =0, these two equations are the same.

Datasets

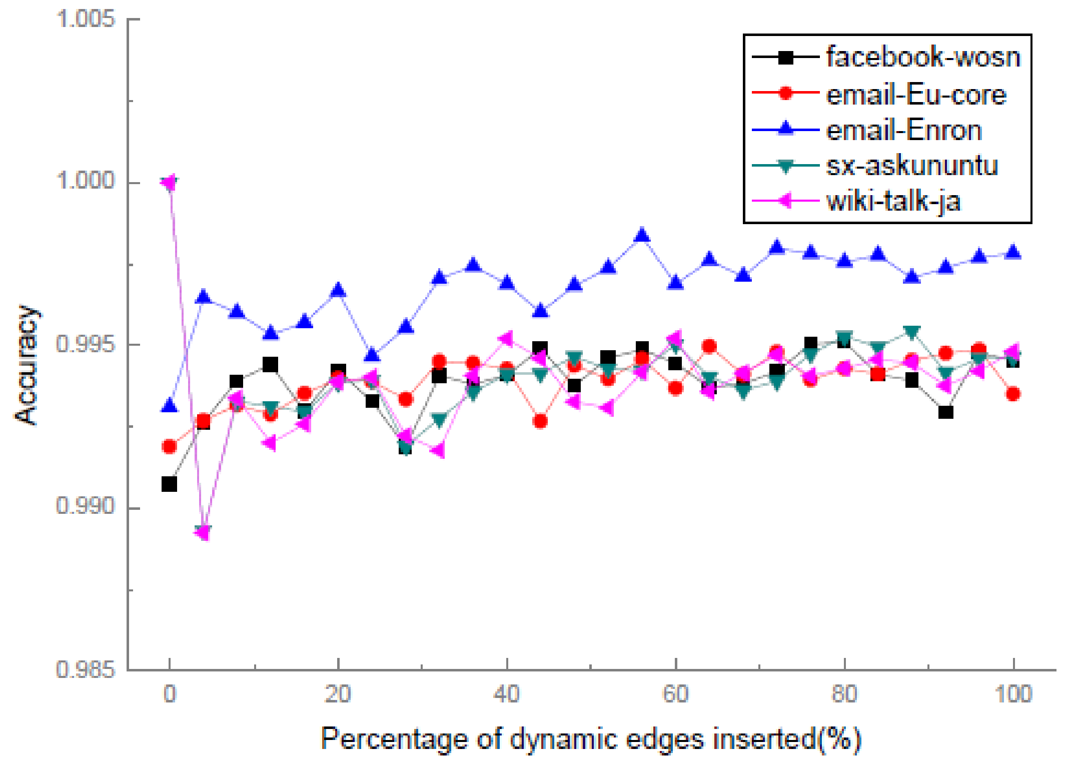
5 real-world dynamic networks with time-stamps, obtained from http://snap.stanford.edu/data/ and http://konect.cc/networks/, were used in our experiments

Table 2. Dynamic networks that we experimented with.

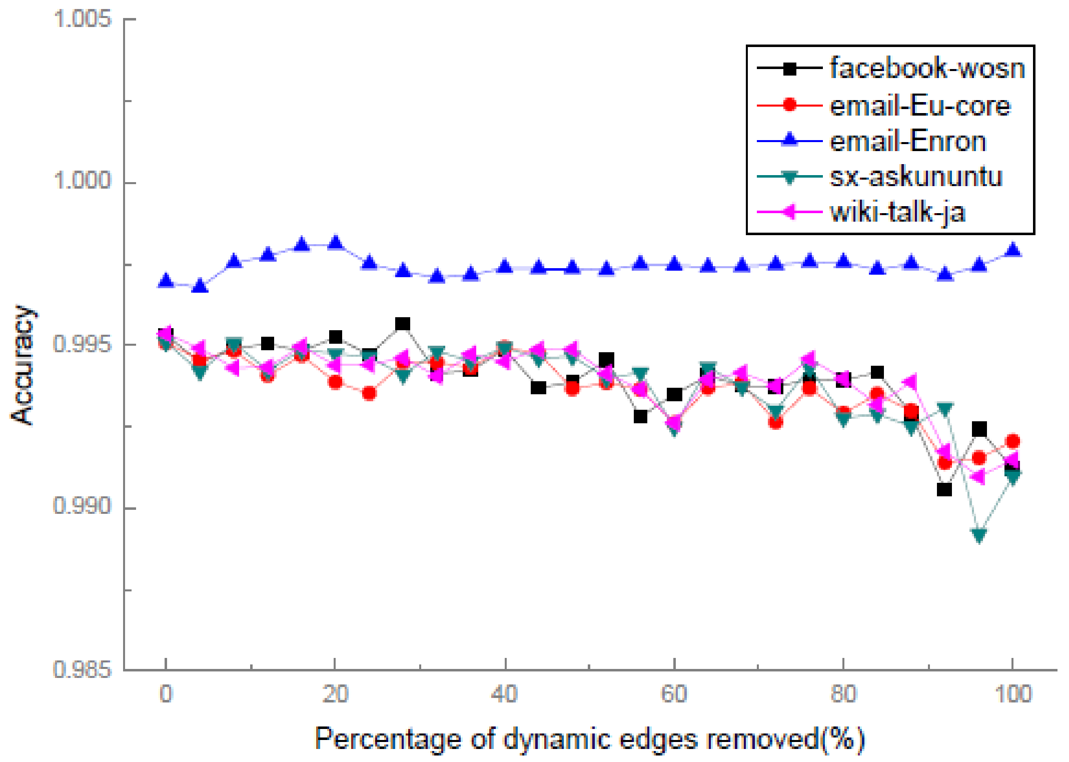


dataset	email-Enron	email-Eu-core	wiki-talk-ja	facebook-wosn	sx-askubuntu
static nodes	3.6K	1K	0	52K	0
dynamic nodes	75K	0	397K	11K	159K
static edges	43K	25.6K	0	1.28M	0
dynamic edges	3.0M	33.2K	1.0M	1.81M	964K

Experimental results: Accuracy



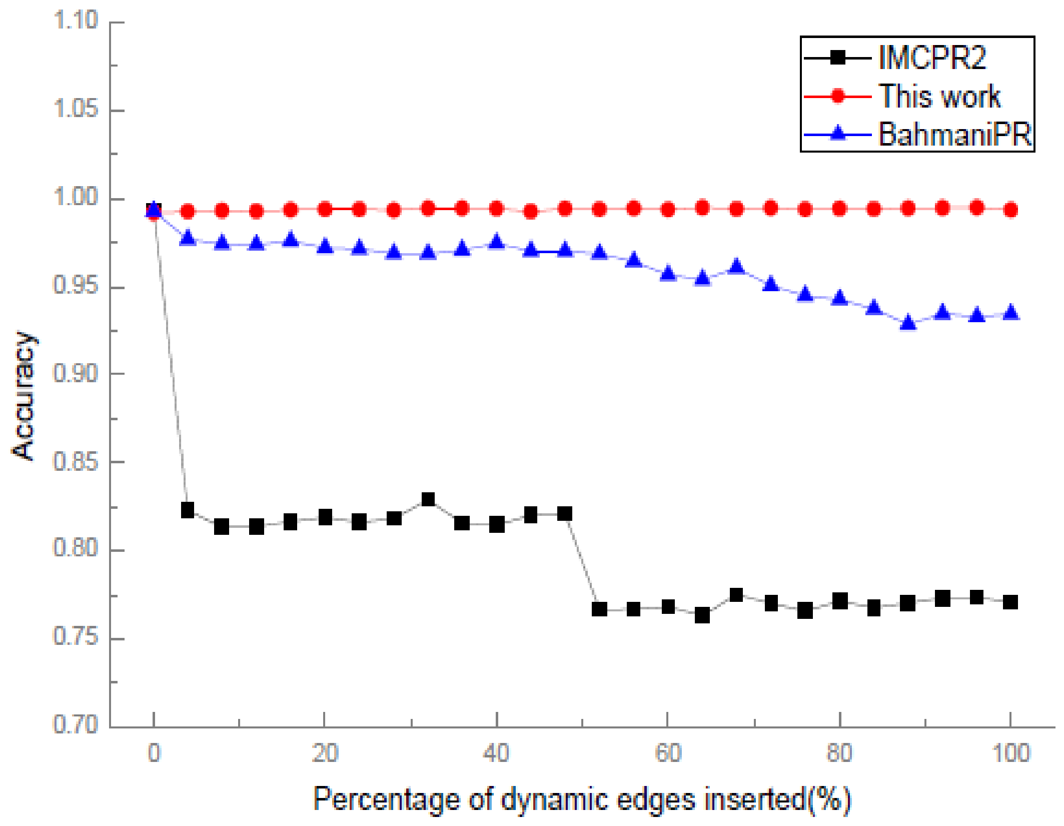
(a) adding edges.



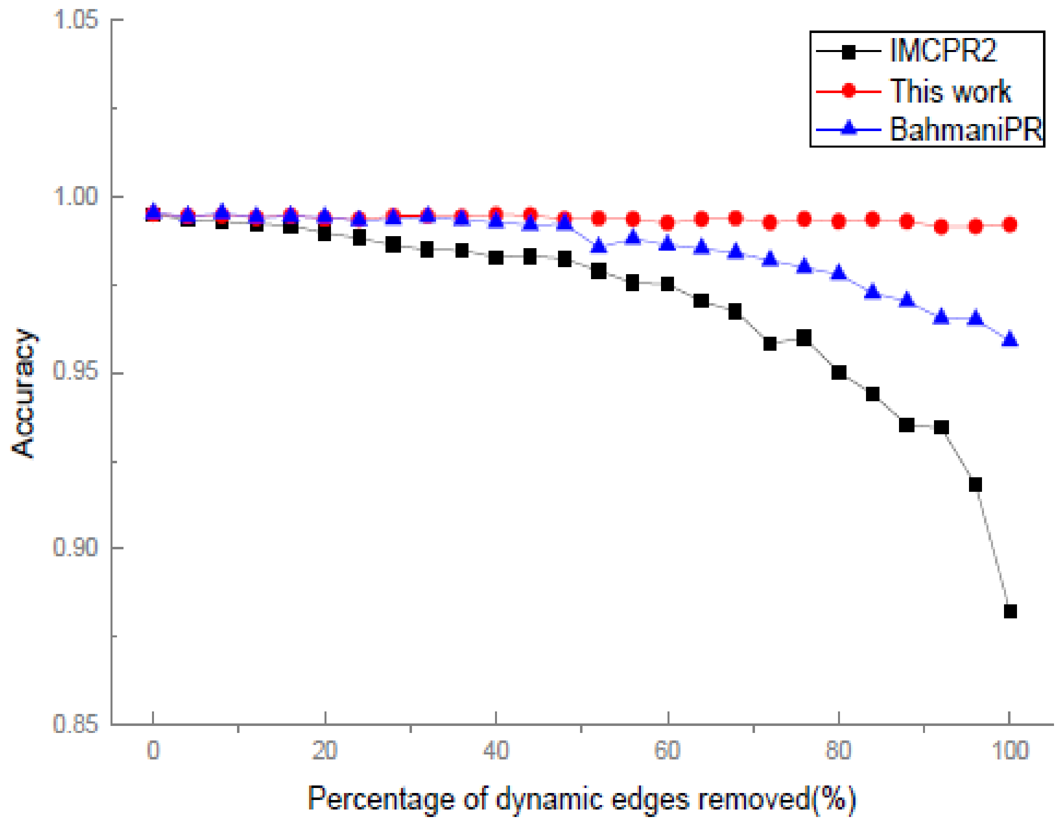
(b) removing edges.

PageRank tracking accuracy of this work on 5 real-world networks

Experimental results: Accuracy



(a) adding edges.

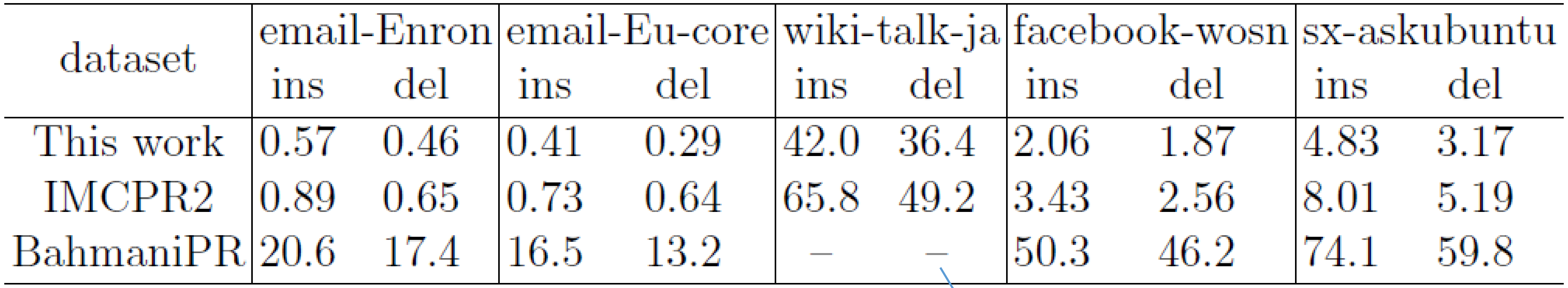


(b) removing edges.

Accuracy comparison of the algorithms on network email-Eu-core

Experimental results: Efficiency

Average update time (ms) for inserting or deleting a single edge.



dataset	email-Enron		email-Eu-core		wiki-talk-ja		facebook-wosn		sx-askubuntu	
	ins	del	ins	del	ins	del	ins	del	ins	del
This work	0.57	0.46	0.41	0.29	42.0	36.4	2.06	1.87	4.83	3.17
IMCPR2	0.89	0.65	0.73	0.64	65.8	49.2	3.43	2.56	8.01	5.19
BahmaniPR	20.6	17.4	16.5	13.2	–	–	50.3	46.2	74.1	59.8

– indicates that algorithm did not finish within 100000s

The proposed algorithm is about

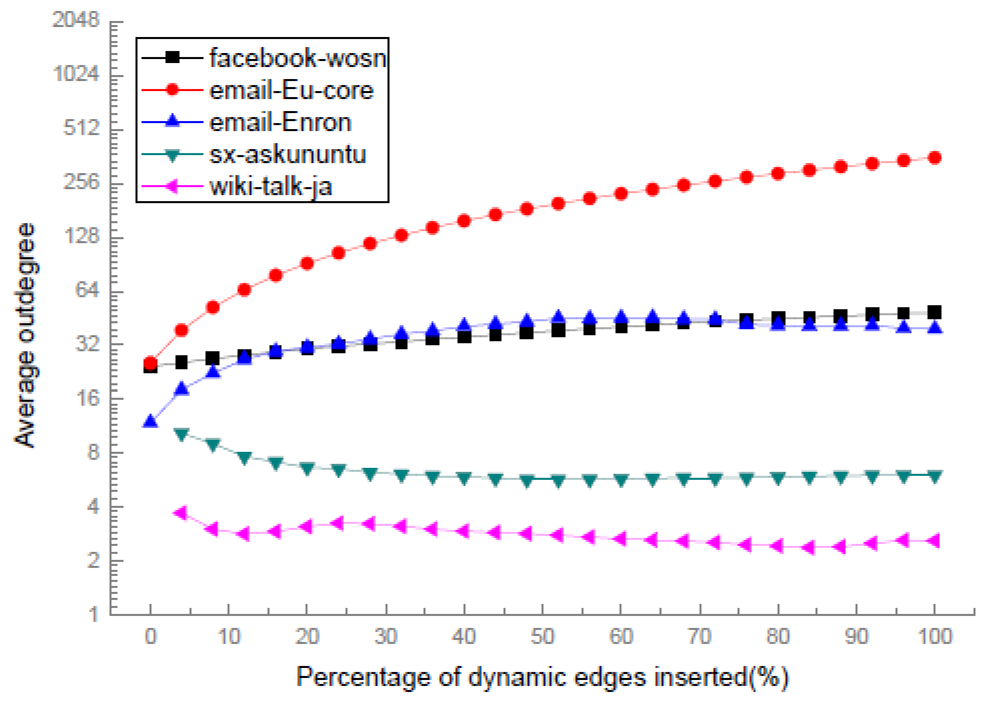
1.3 times faster than IMCPR2 algorithm and

30 times faster than BahmaniPR algorithm.

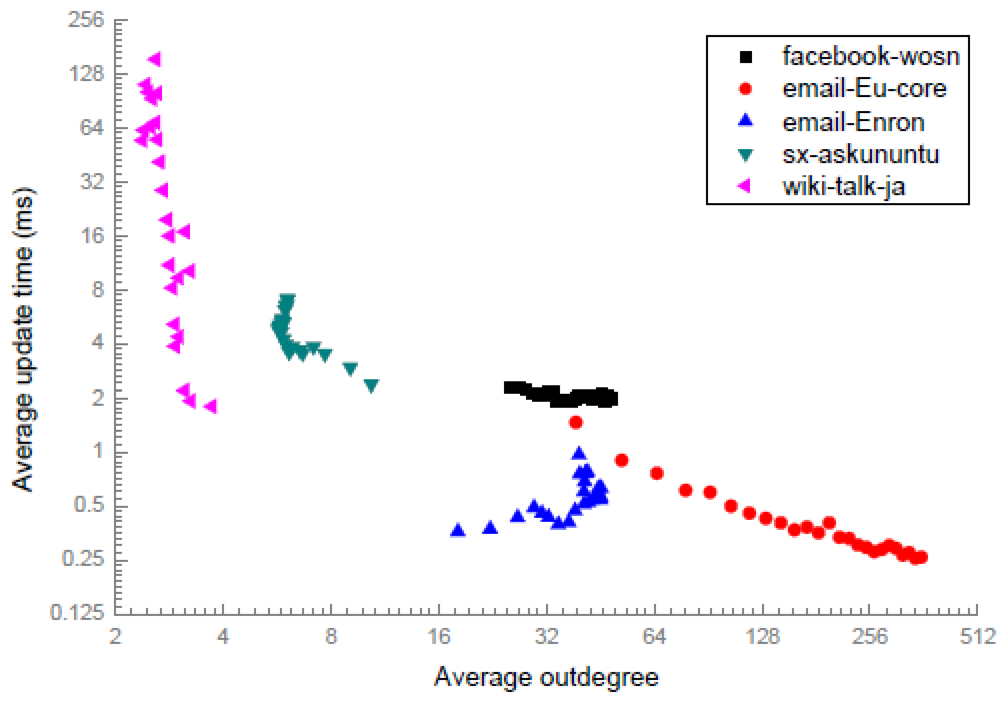
Experimental results: Scalability

The average update time is only inversely proportional to the average out degree.

That means, the proposed algorithm is scalable for large networks.



(a) average outdegree



(b) outdegree versus update time

Why is our approach better?

Accuracy:

- Our algorithm is developed under a more general case (real-world networks with loops)
- Our algorithm properly deals with all added or removed node in a dynamic network

Efficiency：

- Our algorithm saves only one copy of each random walk segments in a hash-table
- Our algorithm adjusts less random walks to update PageRank values

In the future

- Introduce this algorithm to some application aspects, such as event detection on dynamic networks and so on.
- Improve the proposed method to a distributed algorithm, to make it more efficient.
- Migrate the revisit probability model we proposed to some similar research fields, like personalized PageRank and random walk betweenness centrality

Thank You

Collaborators:

Ruimin Hu, Xiyue Gao and Nian Huai

Funding:

National Natural Science Foundation of China

National Key R&D Program of China