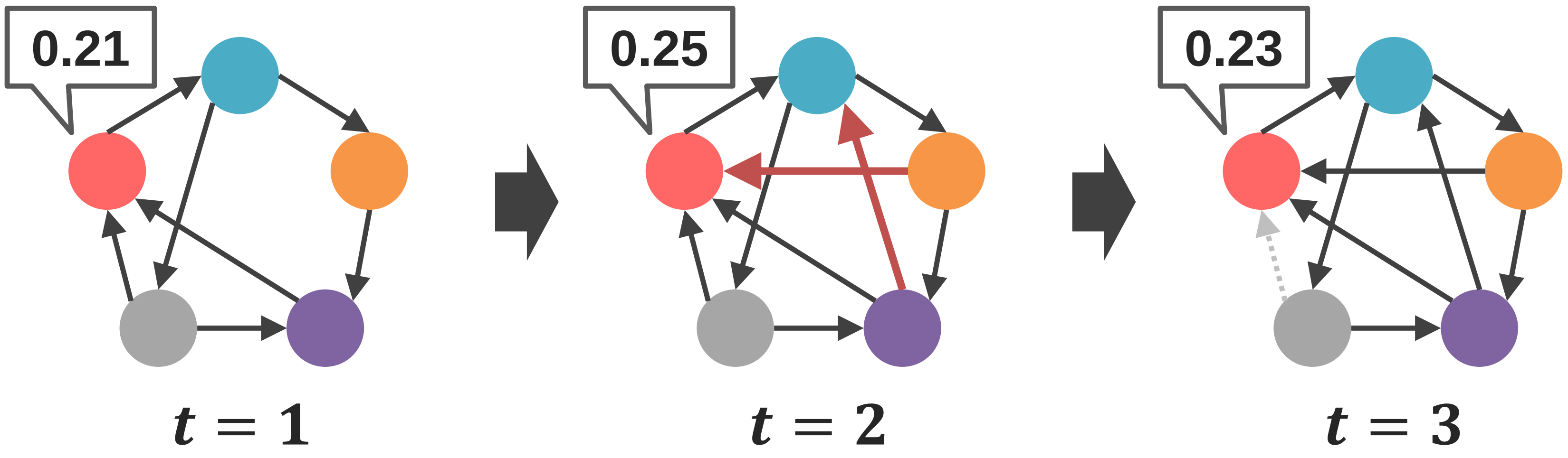
Efficient PageRank Tracking in Evolving Networks

Naoto Ohsaka (UTokyo)
Takanori Maehara (Shizuoka University)
Ken-ichi Kawarabayashi (NII)

ERATO Kawarabayashi Large Graph Project

Personalized PageRank Tracking



Applications

Search engine [Brin-Page. ’98]

Spam detection
[Chung-Toyoda-Kitsuregawa. ’09, ’10]

User recommendation
[Gupta-Goel-Lin-Sharma-Wang-Zadeh. ’13]

Growth of real networks

	Size	Speed
WWW	60T	600K pages / s
Twitter	300M	5K tweets / s
Google+	700M	+19 users / s

Existing Work for PageRank Tracking

	m random edge insertions	Scalability Update time < 0.1s Error ≈ 10^{-9}
Aggregation/Disaggregation [Chien et al. ’04]	$\mathcal{O}(\boldsymbol{m}\lvert\boldsymbol{S}\rvert \log 1/\epsilon)$	68M edges
Monte-Carlo [Bahmani et al. ’10]	$\mathcal{O}(m + \log m\, /\boldsymbol{\epsilon}^{2})$	68M edges
Power method naive method	$\mathcal{O}(\boldsymbol{m}^{2} \log 1/\epsilon)$	11M edges

Our Contribution

Propose a **simple**, **efficient**, & **accurate** method for Personalized PageRank tracking in evolving networks

	m random edge insertions	**Scalability** **Update time < 0.1s** **Error ≈ 10^{-9}**
This work	Ave. ⇩ Max. out-deg $\mathcal{O}(m + \Delta \log m / \epsilon)$	**3,700M** edges
Aggregation/Disaggregation [Chien et al. '04]	$\mathcal{O}(m\|S\| \log 1/\epsilon)$	68M edges
Monte-Carlo [Bahmani et al. '10]	$\mathcal{O}(m + \log m / \epsilon^2)$	68M edges
Power method naive method	$\mathcal{O}(m^2 \log 1/\epsilon)$	11M edges

Definition of Personalized PageRank

[Brin-Page. Comput. Networks ISDN Syst.'98] [Jeh-Widom. WWW'03]

- Linear equation

A solution x of

Preference vector

$$\boldsymbol{x} = \boldsymbol{\alpha} \boldsymbol{P} \boldsymbol{x} + (\mathbf{1} - \boldsymbol{\alpha}) \boldsymbol{b}$$

Transition matrix

Decay factor = 0.85

Random-walk interpretation

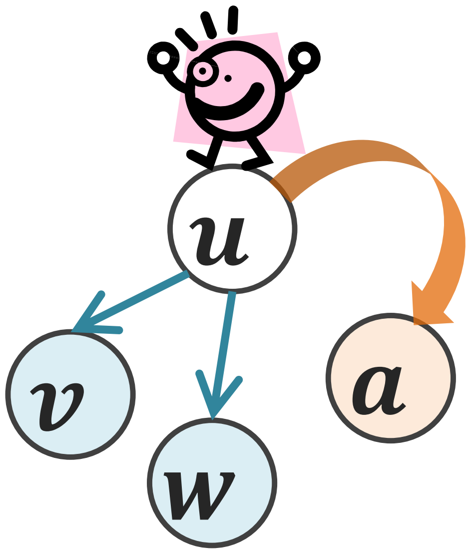
Stationary distribution

- Random walk modeling web browsing

Moves to a random out-neighbor *w.p.* α

Jumps to a random vertex *w.p.* $1 - \alpha$

(biased by b)



Computing PageRank in Static Graphs

- Solving eq. $x = \alpha P x + (1 - \alpha) b$

  Power method $x^{(\nu)} = \alpha P x^{(\nu-1)} + (1 - \alpha) b$

  Gauss-Seidel [Del Corso-Gullí-Romani. Internet Math.'05]

  LU/QR factorization [Fujiwara-Nakatsuji-Onizuka-Kitsuregawa. VLDB'12]

  Krylov subspace method [Maehara-Akiba-Iwata-Kawarabayashi. VLDB'14]

- Estimating the frequency x_v of visiting v

  Monte-Carlo simulation

Tracking PageRank in Evolving Graphs

- Aggregation/disaggregation
  [Chien-Dwork-Kumar-Simon-Sivakumar. Internet Math.'04]

  Apply the power method to a subgraph

  Still large☹

- Monte-Carlo
  [Bahmani-Chowdhury. VLDB'10]

  Maintain & update random-walk segments

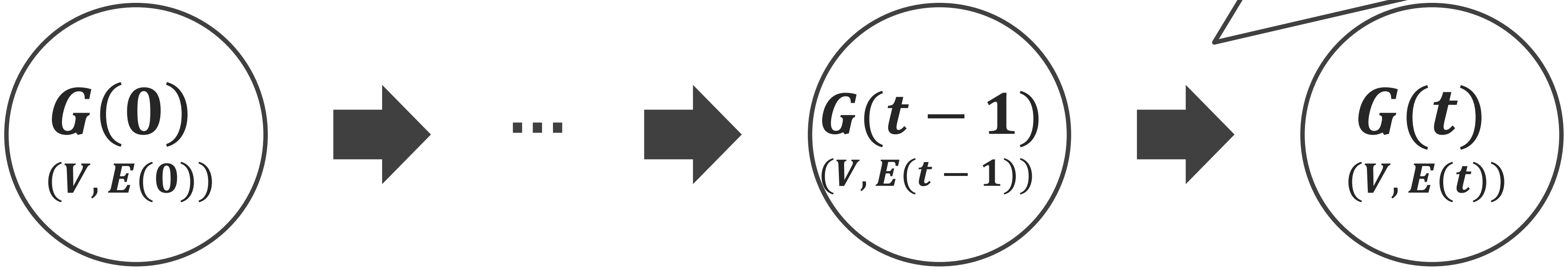
  $\Omega(1/\epsilon^2)$ Too many☹

Proposed Method

Problem Setting

Given at time t:
Edges **inserted** to／**deleted** from

Given $G(0), \alpha, b$ at time 0



Problem at time 0
Compute approx.
PPR $x(0)$ of $G(0)$

$$\boxed{\|\boldsymbol{x}(\boldsymbol{0}) - \boldsymbol{x}^*(\boldsymbol{0})\|_\infty < \boldsymbol{\epsilon}}$$

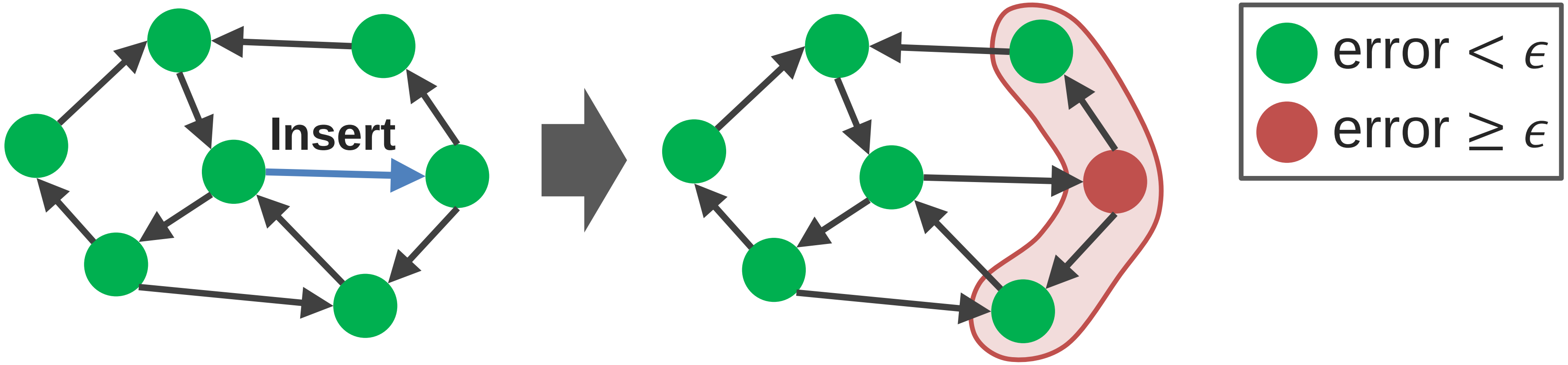
Problem at time t
Compute approx.
PPR $x(t)$ of $G(t)$

The Idea

Solving eq. $x(t) = \alpha P(t)x(t) + (1-\alpha)b$

1. $x(t-1)$ is a **GOOD** initial solution for $x(t)$

2. Improving approximate solution **locally**



- We use the **Gauss-Southwell** method ☺ [Southwell. '40,'46]

a.k.a. *Bookmark coloring algorithm* [Berkhin. Internet Math.'06]

Local algorithm

[Spielman-Teng. SIAM J. Comput.'13] [Andersen-Chung-Lang. FOCS'06]

Proposed Method

Gauss-Southwell Method [Southwell. '40,'46]

- ν^{th} **solution** $x^{(\nu)}$
- ν^{th} **residual** $r^{(\nu)} = (1-\alpha)b - (I - \alpha P)x^{(\nu)}$

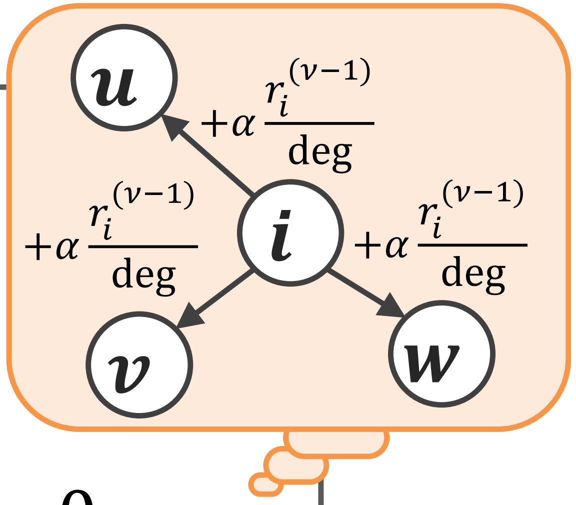
  Goal: $r^{(\nu)} \rightarrow \mathbf{0}$

$\nu = 1,2,3,\ldots$

  $i \leftarrow$ a vertex with largest $\left|r_i^{(\nu-1)}\right|$

  If $\left|r_i^{(\nu-1)}\right| < \epsilon$ **terminate**

  Update $x^{(\nu-1)}$ & $r^{(\nu-1)}$ **locally** so that $r_i^{(\nu)} = 0$



 # iterations

Stops within $\frac{\|r^{(0)}\|}{(1-\alpha)\epsilon}$ iter.

✓ $\|r^{(\nu)}\| \leq \|r^{(\nu-1)}\| - (1-\alpha)\epsilon$

Accuracy

$\left\|\boldsymbol{x}^* - \boldsymbol{x}^{(\nu)}\right\|_\infty \leq \frac{\boldsymbol{\epsilon}}{\mathbf{1}-\boldsymbol{\alpha}}$

✓ $\left|r_i^{(\nu)}\right| < \epsilon$

Overview

At time t:

$x(t)^{(0)} = x(t-1)$

$r(t)^{(0)} = r(t-1) + \alpha\big(P(t) - P(t-1)\big)x(t-1)$

Apply the Gauss-Southwell method

Proposed Method

Performance Analysis

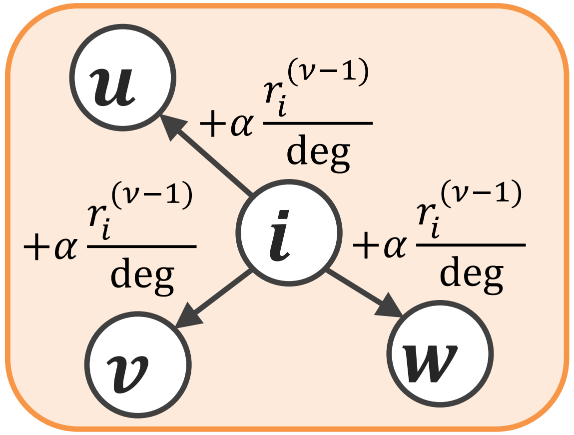
At time t:

$$x(t)^{(0)} = x(t-1)$$

$$r(t)^{(0)} = r(t-1) + \overbrace{\alpha\big(P(t) - P(t-1)\big)x(t-1)}^{\textbf{Increase of } \|r\|_1}$$

Apply the Gauss-Southwell method

Computation time of $\boxed{} = \mathcal{O}(\underset{\text{Max. out-deg } \Uparrow}{\Delta} \times \#\text{iters.})$



depends on $\|\overbrace{\qquad}\|_1$

How small?

Performance Analysis: Any Change

Consider any change including full construction

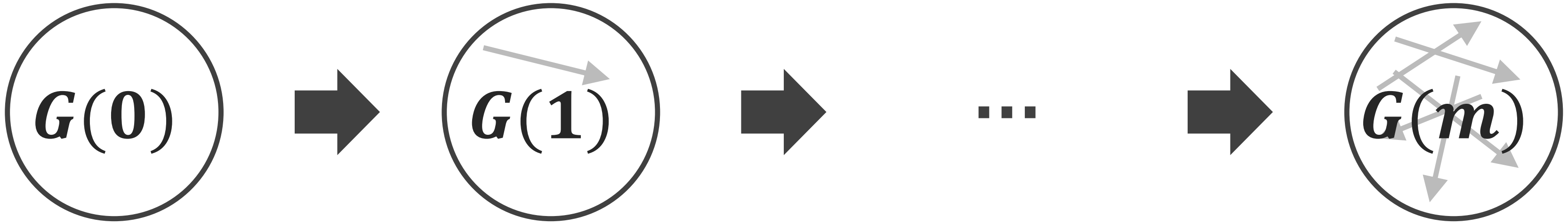
$$G(t-1) \Rightarrow G(t)$$

$$\text{increase of } \|r\|_1 \leq 2\alpha$$

Same as **static computation**

Performance Analysis: Random Edge Insertion

A single-edge is randomly inserted for each time



$E(0) = \emptyset$ Insert an edge $|E(m)| = m$

Lemma 3 in [Bahmani-Chowdhury. VLDB'10]

Monte-Carlo

[Bahmani-Chowdhury. VLDB'10]

$\mathbf{E}[\#\ \text{updated seg.}] = O(R \log m)$

$R = \Omega(1/\epsilon^2)$ is total # seg.

Our method

$\mathbf{E}[\text{increase of } \|r\|_1] \leq 2\alpha/t$

Performance Analysis: Results

Our result for random edge insertion (Prop. 6 in the paper)

If m edges are randomly and sequentially inserted,

expected total #iter. is $\mathcal{O}(\log m/\epsilon)$

⇨ expected total time is $\mathcal{O}(m + \Delta \log m/\epsilon)$

Our result for any change (Prop. 5 in the paper)

#iter. for any change is amortized $\mathcal{O}(1/\epsilon)$

⇨ Time is amortized $\mathcal{O}(\Delta/\epsilon)$

Setting: Single-edge Insertion

$G(0)$ → $G(1)$ → … → $G(10^5)$

Except the last 100,000 edges

Add an edge

chronologically
or
randomly

The whole graph

- Parameter settings
  - $\alpha = 0.85$
  - b has 100 non-zero elements
  - $\epsilon = 10^{-9}$

Efficiency Comparison: Time for an Edge Insertion

	web-Google [SNAP] $\|V\|$=1M $\|E\|$=**5M**	Wikipedia [KONECT] $\|V\|$= 2M $\|E\|$=**40M**	twitter-2010 [LAW] $\|V\|$= 142M $\|E\|$=**1,500M**	uk-2007-05 [LAW] $\|V\|$= 105M $\|E\|$=**3,700M**
This work	**7** μs	**77** μs	**29,383** μs	**2** μs
Aggregation/Disaggregation [Chien et al. '04]	320 μs	40,336 μs	>100,000 μs	>100,000 μs
Monte-Carlo [Bahmani et al. '10]	444 μs	9,196 μs	>100,000 μs	>100,000 μs
Warm start power method	80,994 μs	>100,000 μs	>100,000 μs	>100,000 μs
From scratch power method	>100,000 μs	>100,000 μs	>100,000 μs	>100,000 μs

[KONECT] The Koblenz Network Collection http://konect.uni-koblenz.de/networks/
[LAW] Laboratory for Web Algorithmics http://law.di.unimi.it/datasets.php
[SNAP] Stanford Large Network Dataset Collection http://snap.stanford.edu/data/

Environment: Intel Xeon E5-2690 2.90GHz CPU with 256GB memory

Experiments

Accuracy Comparison: Transition of Ave. L_1 Error

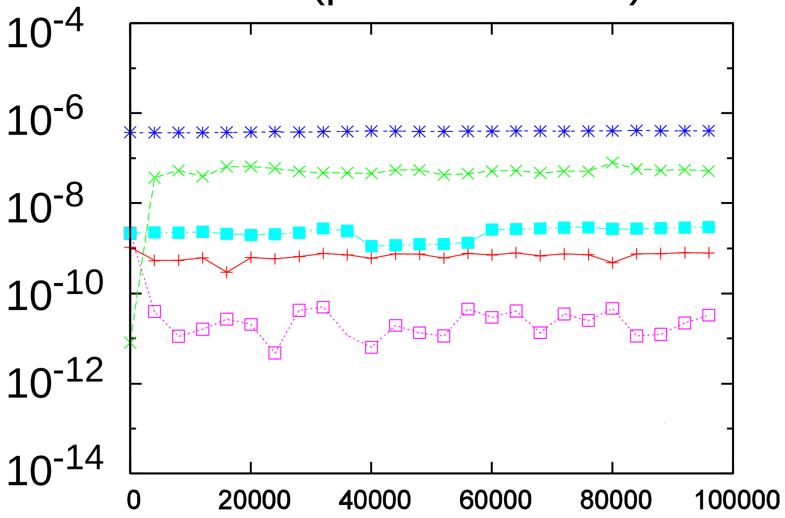
This work

Aggregation/Disaggregation [Chien et al.'04]

Monte-Carlo [Bahmani et al.'10]

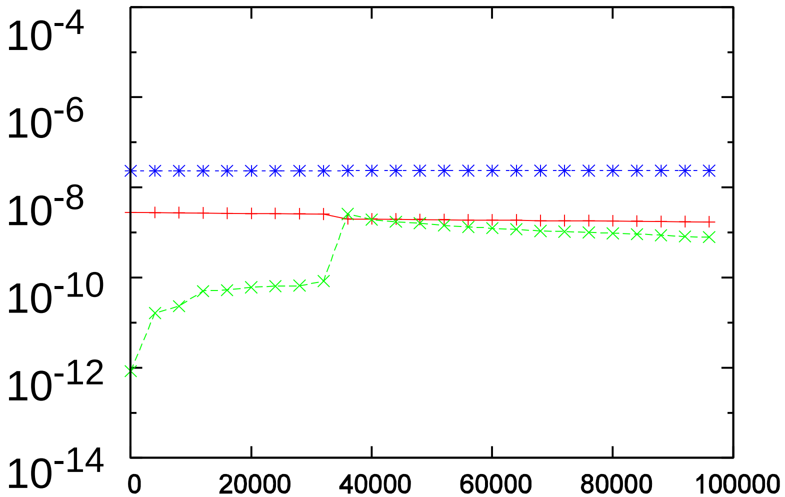
Warm start (power method)

From scratch (power method)



soc-Epinions1 [SNAP]

$|V|$=76K $|E|$=509K



Wikipedia [KONECT]

$|V|$=2M $|E|$=40M

Comparable ($\sim 10^{-9}$) to naive methods

Environment: Intel Xeon E5-2690 2.90GHz CPU with 256GB memory

Experiments

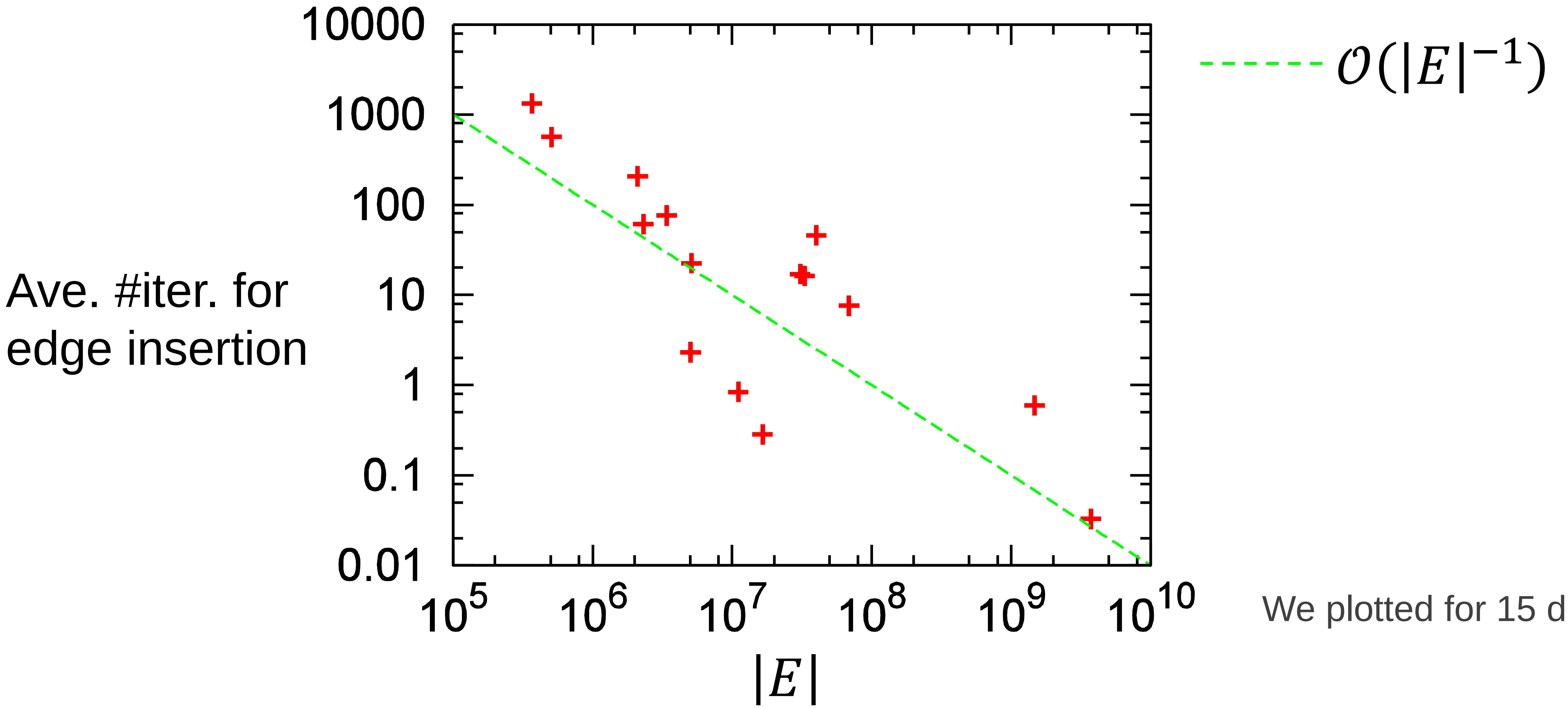
Evaluation: Time & #Iter. for a Single-edge Insertion

Dataset [Source]	$\lvert V \rvert$	$\lvert E \rvert$	Max. Out-deg Δ	Ave. Time	Ave. #Iter.
wiki-Talk [SNAP]	2M	5M	100,022	589.6 μs	2.3
web-Google [SNAP]	1M	5M	3,444	7.2 μs	22.6
as-Skitter [SNAP]	2M	11M	35,387	288.4 μs	0.8
Flickr$^{\text{Time}}$ [KONECT]	2M	33M	26,367	95.3 μs	16.2
Wikipedia$^{\text{Time}}$ [KONECT]	2M	40M	6,975	76.8 μs	46.0
soc-LiveJournal1 [SNAP]	5M	68M	20,292	17.9 μs	7.6
twitter-2010 [LAW]	42M	1,500M	2,997,469	29,382.8 μs	0.7
uk-2007-05 [LAW]	105M	3,700M	15,402	2.3 μs	0.0

[KONECT] The Koblenz Network Collection http://konect.uni-koblenz.de/networks/
[LAW] Laboratory for Web Algorithmics http://law.di.unimi.it/datasets.php
[SNAP] Stanford Large Network Dataset Collection http://snap.stanford.edu/data/

Environment: Intel Xeon E5-2690 2.90GHz CPU with 256GB memory

Evaluation: Relationship between $|E|$ & #Iter.



We plotted for 15 datasets

Matches our theoretical result

Environment: Intel Xeon E5-2690 2.90GHz CPU with 256GB memory

Evaluation: Time & #Iter. for a Single-edge Insertion

Dataset [Source]	$\lvert V\rvert$	$\lvert E\rvert$	Max. Out-deg Δ	Ave. Time	Ave. #Iter.
wiki-Talk [SNAP]	2M	5M	100,022	589.6 μs	2.3
web-Google [SNAP]	1M	5M	3,444	7.2 μs	22.6
as-Skitter [SNAP]	2M	11M	35,387	288.4 μs	0.8
Flickr$^{\text{Time}}$ [KONECT]	2M	33M	26,367	95.3 μs	16.2
Wikipedia$^{\text{Time}}$ [KONECT]	2M	40M	6,975	76.8 μs	46.0
soc-LiveJournal1 [SNAP]	5M	68M	20,292	17.9 μs	7.6
twitter-2010 [LAW]	42M	1,500M	**2,997,469**	**29,382.8** μs	0.7
uk-2007-05 [LAW]	105M	3,700M	**15,402**	**2.3** μs	0.0

[KONECT] The Koblenz Network Collection http://konect.uni-koblenz.de/networks/
[LAW] Laboratory for Web Algorithmics http://law.di.unimi.it/datasets.php
[SNAP] Stanford Large Network Dataset Collection http://snap.stanford.edu/data/

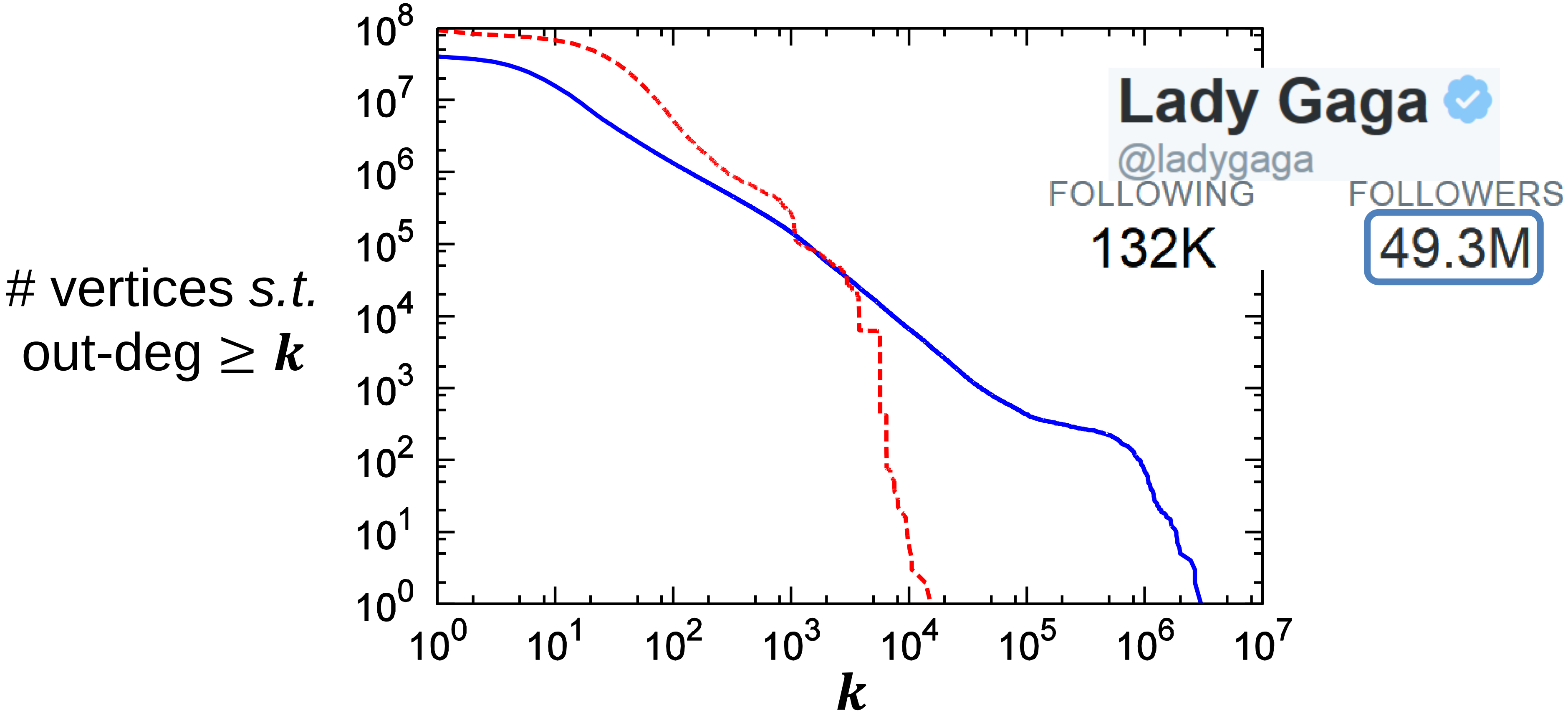
What Happens !?

Environment: Intel Xeon E5-2690 2.90GHz CPU with 256GB memory

Discussion: What is the Difference?

twitter-2010 (u, v) says "v follows u"

uk-2007-05 (u, v) says "Page u links to v"



Celebrities cause the performance degradation !!

Summary

Proposed an efficient & accurate method for
Personalized PageRank tracking in evolving networks

Theoretically

Ave. $\mathcal{O}(m + \Delta \log m / \epsilon)$
for m **edge insertions**

Experimentally

Scales to a graph w/ **3.7B edges**

Future Work

- Further Speed-up based on our observation
- Handle dangling nodes