Towards Efficient SimRank Computation over Large Networks

Weiren Yu[1,2], Xuemin Lin[1], Wenjie Zhang[1]

[1] University of New South Wales
[2] NICTA, Australia

Outline

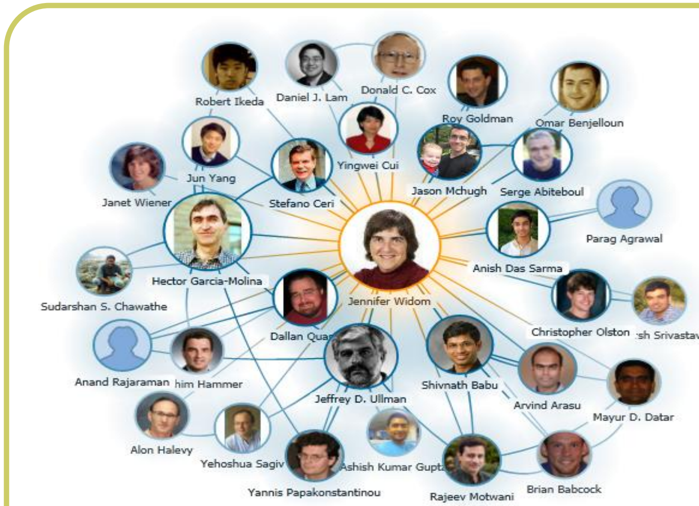
- **SimRank overview**
- Existing computing method
- Our approaches
  - Partial Sums Sharing
  - Exponential SimRank
- Empirical evaluations
- Conclusions

SimRank Overview

- Similarity Computation plays a vital role in our lives.

Recommender System

Citation Graph



Collaboration Network

SimRank Overview

- SimRank
  - An appealing link-based similarity measure (KDD '02)
  - Basic philosophy

    Two vertices are similar if they are referenced by similar vertices.

- Two Forms
  - Original form (KDD '02)

damping factor

$$s(a,a) = 1$$

similarity btw. nodes a and b

$$s(a,b) = \frac{C}{|\mathcal{I}(a)|\,|\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s(i,j)$$

in-neighbor set of node b

  - Matrix form (EDBT '10)

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n$$

Existing Computing Methods

- State-of-the-art Method (VLDB J. '10)
  - Partial Sum Memoization

memoize

$$\boxed{Partial^{s_k}_{\mathcal{I}(a)}(j)} = \sum_{i \in \mathcal{I}(a)} s_k(i,j), \quad (j \in \mathcal{I}(b))$$

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \boxed{Partial^{s_k}_{\mathcal{I}(a)}(j)}$$

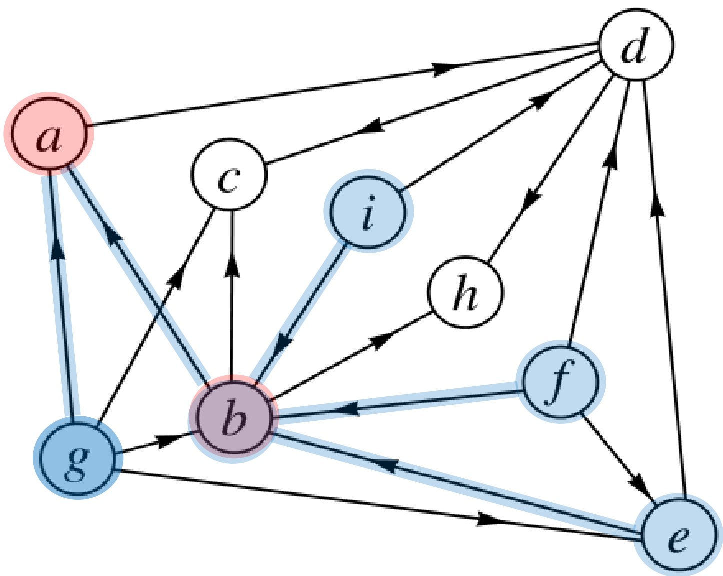
reuse

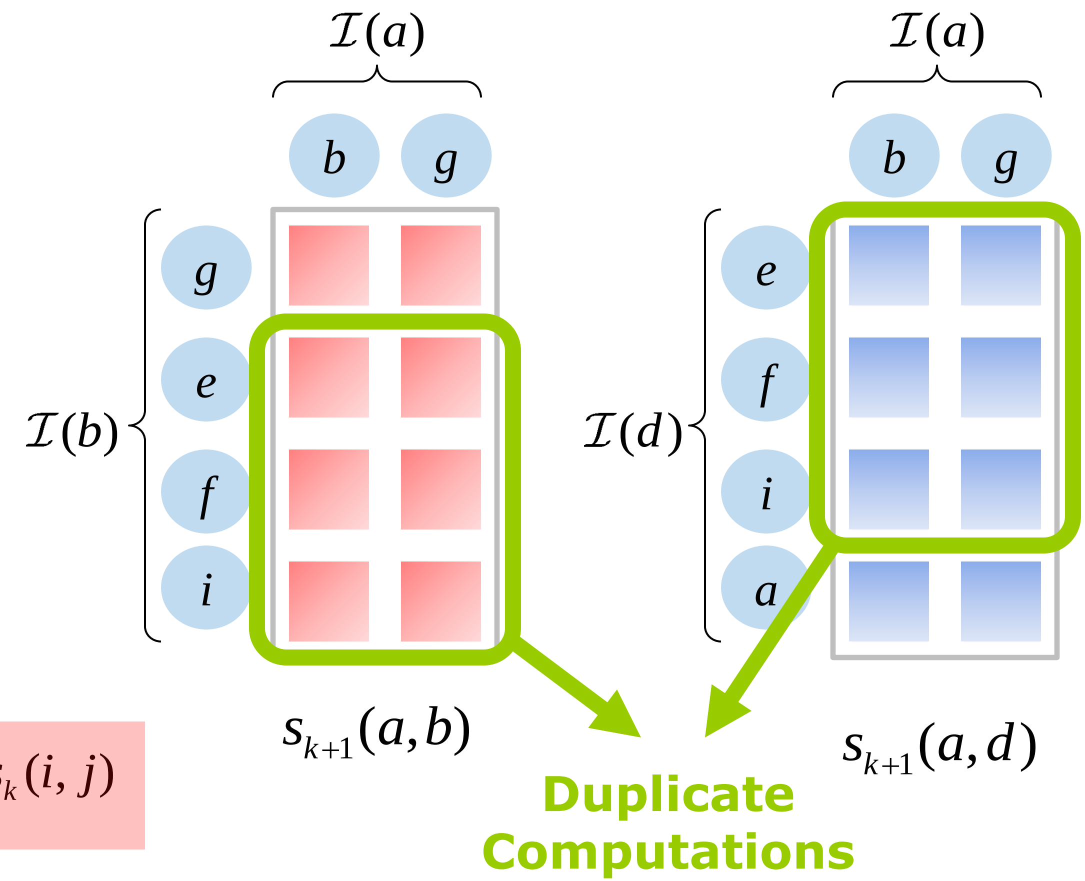
- Two Limitations
  - High computation time: O(Kdn^2)
  - Low convergence rate:

$$|s_k(a,b) - s(a,b)| \le C^{k+1}$$

Naïve vs. VLDB J.'10

- Example: Compute $s(a,b)$, $s(a,d)$





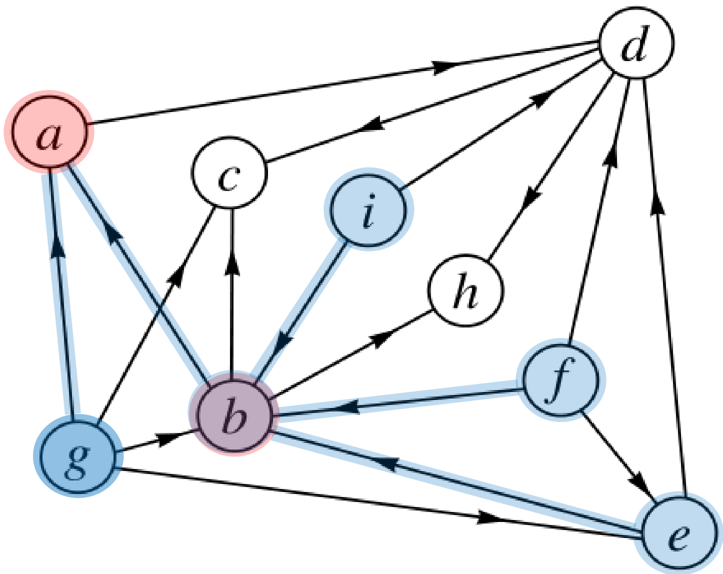
Naïve

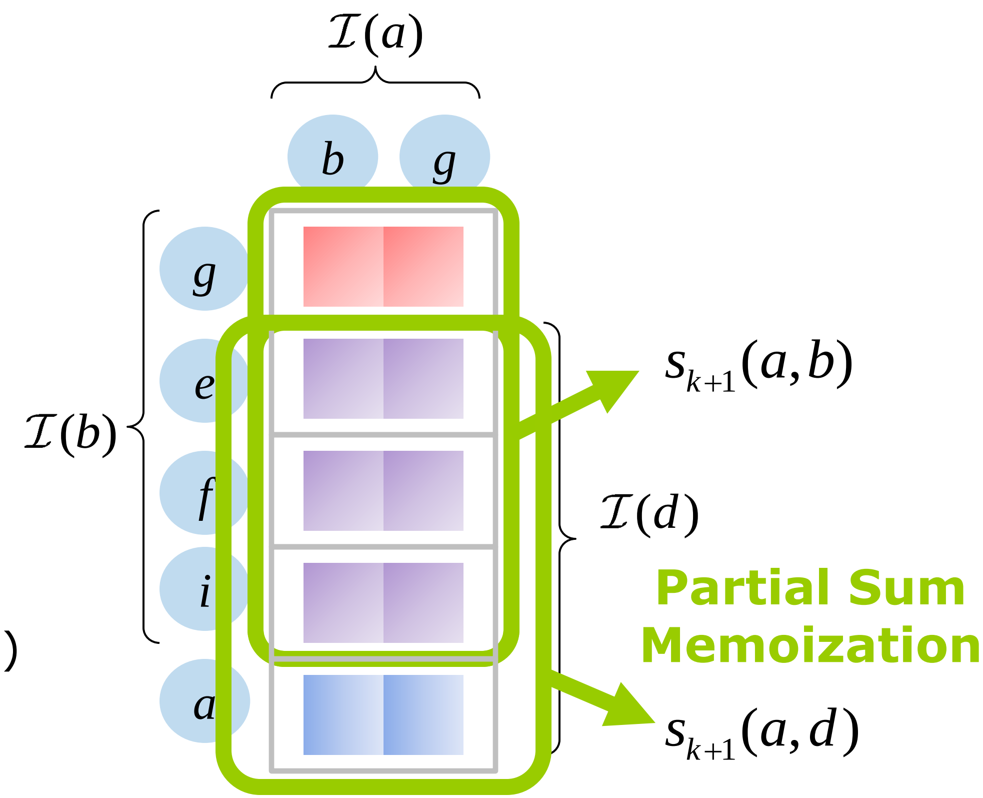
$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s_k(i,j)$$

$$s_{k+1}(a,d) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(d)|} \sum_{j \in \mathcal{I}(d)} \sum_{i \in \mathcal{I}(a)} s_k(i,j)$$

Naïve vs. VLDB J.'10

- Example: Compute $s(a,b)$, $s(a,d)$





Partial Sums Memoization (VLDB J.'10)

$$\forall j, \quad \boxed{Partial^{s_k}_{\mathcal{I}(a)}(j)} = \sum_{i \in \mathcal{I}(a)} s_k(i,j)$$

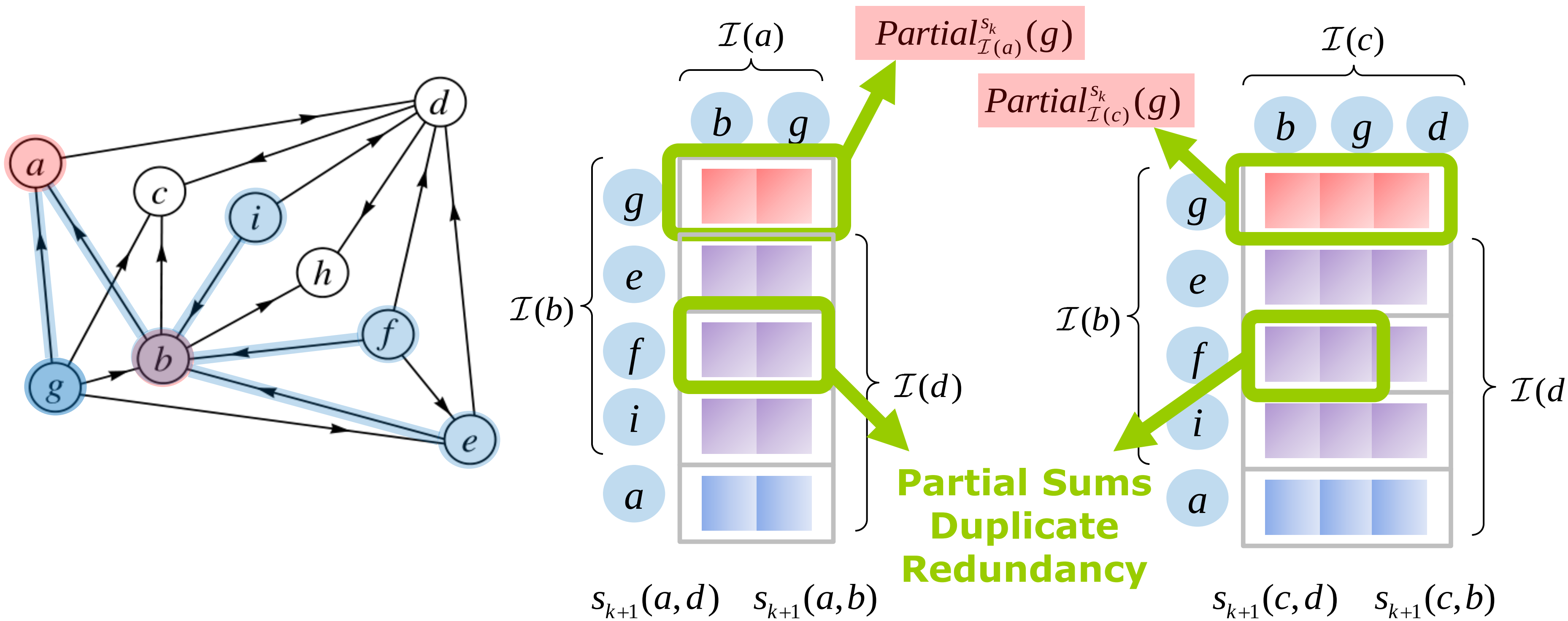
$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \boxed{Partial^{s_k}_{\mathcal{I}(a)}(j)}$$

$$s_{k+1}(a,d) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(d)|} \sum_{j \in \mathcal{I}(d)} \boxed{Partial^{s_k}_{\mathcal{I}(a)}(j)}$$

reuse

VLDB J.'10 Limitations

- Example: Compute $s(a,b)$, $s(a,d)$, $s(c,b)$, $s(c,d)$



$$\forall j, \quad Partial^{s_k}_{\mathcal{I}(a)}(j) = \sum_{i \in \mathcal{I}(a)} s_k(i,j)$$

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} Partial^{s_k}_{\mathcal{I}(a)}(j)$$

$$\forall j, \quad Partial^{s_k}_{\mathcal{I}(c)}(j) = \sum_{i \in \mathcal{I}(c)} s_k(i,j)$$

$$s_{k+1}(c,b) = \frac{C}{|\mathcal{I}(c)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} Partial^{s_k}_{\mathcal{I}(c)}(j)$$

Motivation

- Prior Work (VLDB J. '10)
  - High time complexity: $O(Kdn^2)$
    - Duplicate computation among partial sums memoization
  - Slow (geometric) convergence rate
    - Require $K = \lceil \log_C \epsilon \rceil$ iterations to guarantee accuracy ϵ

- Our Contributions
  - Propose an adaptive clustering strategy
    to reduce the time from $O(Kdn^2)$ to $O(Kd'n^2)$, where $d' \leq d$.
  - Introduce a new notion of SimRank
    to accelerate convergence from geometric to exponential rate.

UNSW

Eliminating Partial Sums Redundancy

- Main idea
  - share common sub-summations among partial sums

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j\in\mathcal{I}(b)} \sum_{\Delta\in\mathscr{P}(\mathcal{I}(a))} Partial_{\Delta}^{s_k}(j)$$

NP-hard to find "best" partition for $\mathcal{I}(a)$

fine-grained partial sums

- Example

Existing Approach (3 additions)

$$Partial_{\mathcal{I}(a)}^{s_k}(g) = s_k(b,g) + s_k(g,g)$$

$$Partial_{\mathcal{I}(c)}^{s_k}(g) = s_k(b,g) + s_k(g,g) + s_k(d,g)$$

Duplicate Computation !!

Our Approach (only 2 additions)

Partition $\mathcal{I}(c) = \mathcal{I}(a) \cup \{d\}$

$$Partial_{\mathcal{I}(a)}^{s_k}(g) = s_k(b,g) + s_k(g,g)$$

$$Partial_{\mathcal{I}(c)}^{s_k}(g) = Partial_{\mathcal{I}(a)}^{s_k}(g) + s_k(d,g)$$

Heuristic for Finding Partitions

- Find: partitions for each in-neighbor set
  Minimize: the total cost of all partial sums

- Main Idea
  - Calculate *transition cost* btw. in-neighbor sets

$$\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)} \triangleq \min\{|\mathcal{I}(a) \ominus \mathcal{I}(b)|, |\mathcal{I}(b)| - 1\}$$

  - Construct a weighted digraph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ s.t.

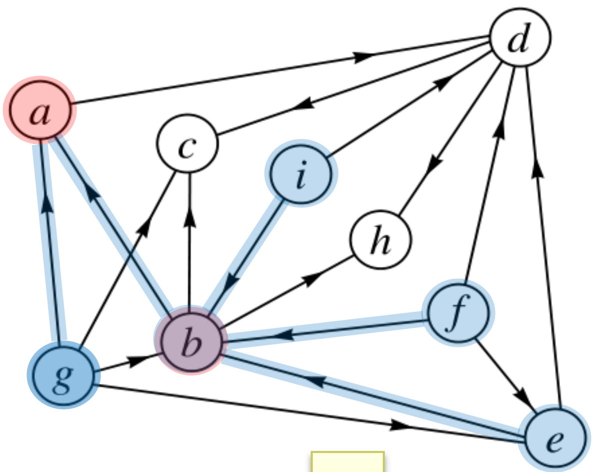
$$\mathscr{V} = \{\mathcal{I}(a) \mid a \in \mathcal{V}\} \cup \{\varnothing\}$$

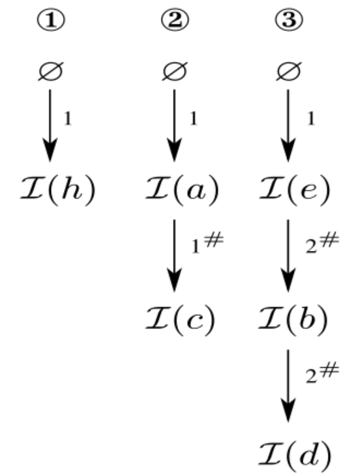
$$(\mathcal{I}(a), \mathcal{I}(b)) \in \mathscr{E} \text{ if } |\mathcal{I}(a)| \leq |\mathcal{I}(b)|$$

$$\text{weight of } (\mathcal{I}(a), \mathcal{I}(b)) = \mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)}$$

  - Find a MST of $\mathscr{G}$ to minimize the total transition cost

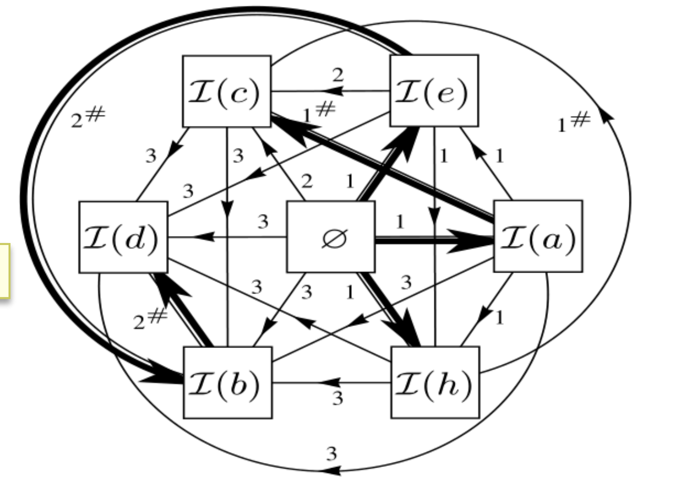
Example





(d) Partial Sums Order

(4)



(c) Minimum Spanning Tree $\mathscr{T}$ of $\mathscr{G}$

(1)

(3)

vertex	$\mathcal{I}(\star)$
a	$\{b, g\}$
e	$\{f, g\}$
h	$\{b, d\}$
c	$\{b, d, g\}$
b	$\{f, g, e, i\}$
d	$\{f, a, e, i\}$

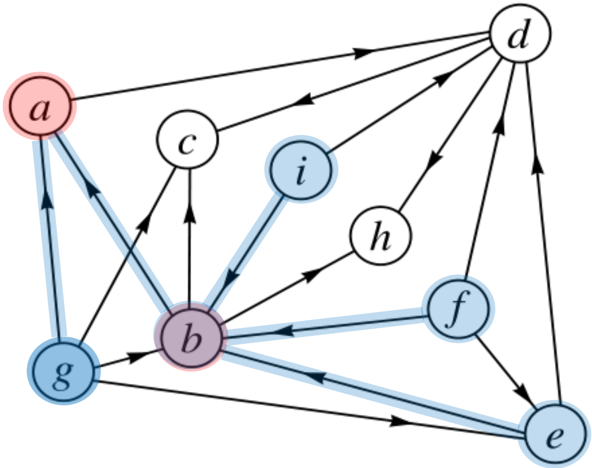
(a) In-neighbors in $\mathcal{G}$

(2)

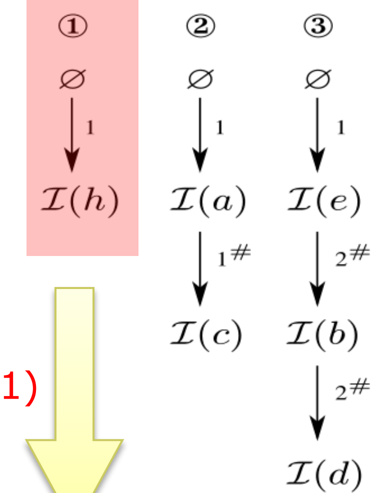
	$\mathcal{I}(a)$	$\mathcal{I}(e)$	$\mathcal{I}(h)$	$\mathcal{I}(c)$	$\mathcal{I}(b)$	$\mathcal{I}(d)$
∅	1	1	1	2	3	3
$\mathcal{I}(a)$		1	1	$1^{\#}$	3	3
$\mathcal{I}(e)$			1	2	$2^{\#}$	3
$\mathcal{I}(h)$				$1^{\#}$	3	3
$\mathcal{I}(c)$					3	3
$\mathcal{I}(b)$						$2^{\#}$

(b) Transition Costs (Edge Weights) in $\mathscr{G}$

Example



Partial Sums Order



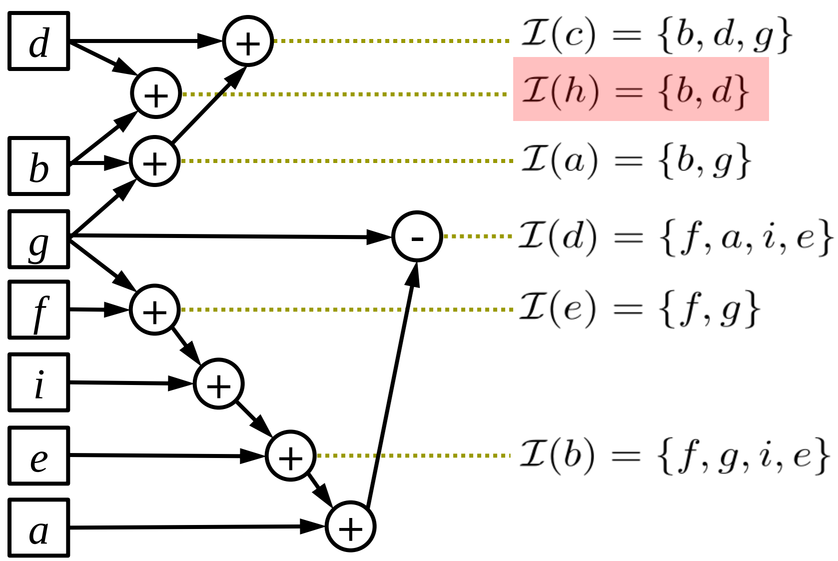
(1)

Partitions of $\mathcal{I}(\star)$ in $\mathcal{G}$

	$\mathscr{P}(\star)$
$\mathcal{I}(a)$	$\{\{b, g\}\}$
$\mathcal{I}(e)$	$\{\{f, g\}\}$
$\mathcal{I}(h)$	$\{\{b, d\}\}$
$\mathcal{I}(c)$	$\{\mathcal{I}(a), \{d\}\}$
$\mathcal{I}(b)$	$\{\mathcal{I}(e), \{e, i\}\}$
$\mathcal{I}(d)$	$\{\mathcal{I}(b) \setminus \{g\}, \{a\}\}$

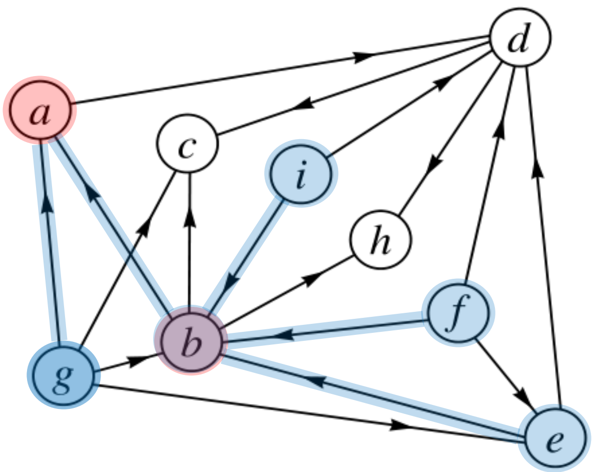
In-neighbors in $\mathcal{G}$

vertex	$\mathcal{I}(\star)$
a	$\{b, g\}$
e	$\{f, g\}$
h	$\{b, d\}$
c	$\{b, d, g\}$
b	$\{f, g, e, i\}$
d	$\{f, a, e, i\}$

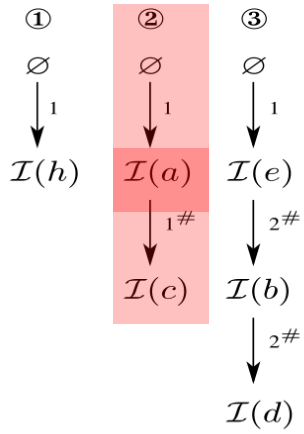


(2)

Outer Partial Sums Sharing



Partial Sums Order



Partitions of $\mathcal{I}(\star)$ in $\mathcal{G}$

	$\mathscr{P}(\star)$
$\mathcal{I}(a)$	$\{\{b, g\}\}$
$\mathcal{I}(e)$	$\{\{f, g\}\}$
$\mathcal{I}(h)$	$\{\{b, d\}\}$
$\mathcal{I}(c)$	$\{\mathcal{I}(a), \{d\}\}$
$\mathcal{I}(b)$	$\{\mathcal{I}(e), \{e, i\}\}$
$\mathcal{I}(d)$	$\{\mathcal{I}(b)\backslash\{g\}, \{a\}\}$

- (Inner) partial sums sharing

$$Partial^{s_k}_{\mathcal{I}(a)}(\star) = s_k(b, \star) + s_k(g, \star)$$

$$Partial^{s_k}_{\mathcal{I}(c)}(\star) = Partial^{s_k}_{\mathcal{I}(a)}(\star) + s_k(d, \star)$$

- Outer partial sums sharing

$$OuterPartial^{\mathcal{I}(\star), s_k}_{\mathcal{I}(a)} = \sum_{y \in \{b,g\}} Partial^{s_k}_{\mathcal{I}(\star)}(y)$$

$$OuterPartial^{\mathcal{I}(\star), s_k}_{\mathcal{I}(c)} = OuterPartial^{\mathcal{I}(\star), s_k}_{\mathcal{I}(a)} + Partial^{s_k}_{\mathcal{I}(\star)}(d)$$

$$s_{k+1}(a, \star) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(\star)|} \; OuterPartial^{\mathcal{I}(\star), s_k}_{\mathcal{I}(a)}$$

Exponential SimRank

- Existing Approach (VLDB J. '10)

$$\left\|\mathbf{S}_k - \mathbf{S}\right\|_{\max} \le C^{k+1}$$

Geometric Rate

For $C = 0.8$, to guarantee the accuracy $\epsilon = 0.0001$, there are

$$K = \lceil \log_{0.8} 0.0001 \rceil = 41 \text{ iterations.}$$

- Our Approach

$$\left\|\hat{\mathrm{S}}_k - \hat{\mathrm{S}}\right\|_{\max} \le \frac{C^{k+1}}{(k+1)!}$$

Exponential Rate

For $C = 0.8$, $\epsilon = 0.0001$, we need only 7 iterations.

Exponential SimRank

- Key Observation

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n \Rightarrow \mathbf{S} = (1 - C) \cdot \sum_{i=0}^{\infty} \underbrace{C^i}_{\text{Geometric Sum}} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i$$

The effect of C^i is to *reduce* the contribution of *long* paths relative to *short* ones

- Main Idea
  - Accelerate convergence by replacing the *geometric sum* of SimRank with the *exponential sum.*

$$\underbrace{\frac{d\hat{\mathbf{S}}(t)}{dt} = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T}_{\text{Differential Equation}}, \quad \underbrace{\hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n}_{\text{Initial Condition}}. \Leftarrow \hat{\mathbf{S}} = \underbrace{e^{-C}}_{\text{Normalized Factor}} \cdot \sum_{i=0}^{\infty} \underbrace{\frac{C^i}{i!}}_{\text{Exponential Sum}} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i$$

SimRank Differential Equation

- Recursive form for Differential SimRank

$$\frac{d\hat{\mathbf{S}}(t)}{dt} = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T, \quad \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n.$$

- Conventional computing method
  - Euler iteration: Set $t_k = k \cdot h$,

$$\hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + h \cdot \mathbf{Q} \cdot \hat{\mathbf{S}}_k \cdot \mathbf{Q}^T, \quad \hat{\mathbf{S}}_0 = \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n.$$

  - Disadvantage: Hard to determine the value of *h*.
- Our approach

$$\begin{cases} \mathbf{T}_{k+1} = \mathbf{Q} \cdot \mathbf{T}_k \cdot \mathbf{Q}^T \\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C} \cdot \frac{C^{k+1}}{(k+1)!} \cdot \mathbf{T}_{k+1} \end{cases} \text{with} \begin{cases} \mathbf{T}_0 = \mathbf{I}_n \\ \hat{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n \end{cases}$$

SimRank Differential Equation

- Accuracy Guarantee $\begin{cases} \mathbf{T}_{k+1} = \mathbf{Q} \cdot \mathbf{T}_k \cdot \mathbf{Q}^T \\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C} \cdot \frac{C^{k+1}}{(k+1)!} \cdot \mathbf{T}_{k+1} \end{cases}$ with $\begin{cases} \mathbf{T}_0 = \mathbf{I}_n \\ \hat{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n \end{cases}$

$$\|\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\|_{\max} \leq \frac{C^{k+1}}{(k+1)!}$$

- #-Iterations
  - Use Lambert W function

$$K' \geq \left\lceil \frac{\ln \epsilon'}{W(\frac{1}{e \cdot C} \cdot \ln \epsilon')} \right\rceil, \ with\ \epsilon' = (\sqrt{2\pi} \cdot \epsilon)^{-1}$$

  - Use Log function (for $0 < \epsilon < \frac{1}{\sqrt{2\pi}} e^{-C \cdot e^2}$)

$$K' \geq \left\lceil \frac{-\ln(\sqrt{2\pi} \cdot \epsilon)}{\eta - \ln(\eta)} \right\rceil \ with\ \eta = \ln(-\tfrac{1}{e \cdot C} \cdot \ln(\sqrt{2\pi} \cdot \epsilon)).$$

- Example ($C = 0.8$, $\epsilon = 0.0001$)

$$\eta = \ln(-\tfrac{1}{e \cdot 0.8} \cdot \ln(\sqrt{2\pi} \cdot 0.0001)) = 1.3384,$$

$$K' \geq \left\lceil \frac{-\ln(\sqrt{2\pi} \cdot 0.0001)}{1.3384 - \ln(1.3384)} \right\rceil = \left\lceil \frac{8.2914}{1.0469} \right\rceil = 7.$$

Comparison

- Geometric vs. Exponential SimRank

	(Geometric) SimRank	Exponential SimRank
Closed form	$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1-C) \cdot \mathbf{I}_n$	$\frac{d\hat{\mathbf{S}}(t)}{dt} = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T, \qquad \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n.$
Series form	$\mathbf{S} = (1-C) \cdot \sum_{i=0}^{\infty} C^i \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i$	$\hat{\mathbf{S}} = e^{-C} \cdot \sum_{i=0}^{\infty} \frac{C^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i$
Iterative form	$\mathbf{S}_0 = \mathbf{I}_n$ $\mathbf{S}_{k+1} = C \cdot (\mathbf{Q} \cdot \mathbf{S}_k \cdot \mathbf{Q}^T) + (1-C) \cdot \mathbf{I}_n$	$\begin{cases} \mathbf{T}_{k+1} = \mathbf{Q} \cdot \mathbf{T}_k \cdot \mathbf{Q}^T \\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C} \cdot \frac{C^{k+1}}{(k+1)!} \cdot \mathbf{T}_{k+1} \end{cases}$ with $\begin{cases} \mathbf{T}_0 = \mathbf{I}_n \\ \hat{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n \end{cases}$
Accuracy	$\Vert\mathbf{S}_k - \mathbf{S}\Vert_{\max} \le C^{k+1}$	$\Vert\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\Vert_{\max} \le \frac{C^{k+1}}{(k+1)!}$

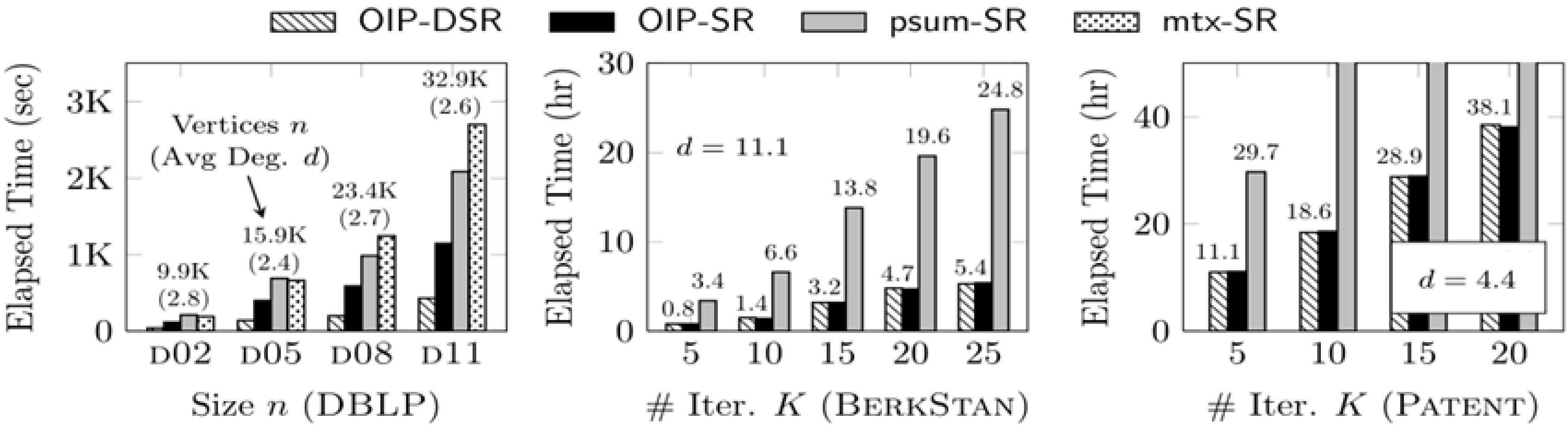
Experimental Settings

- Datasets
  - Real graph: BERKSTAN, PATENT, DBLP (D02, D05, D08, D11)
  - Synthetic data: SYN100K (via GTGraph generator)

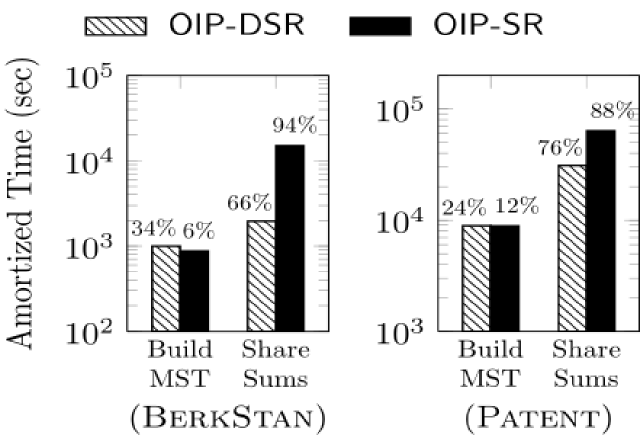
- Compared Algorithms
  - OIP-DSR: Differential SimRank + Partial Sums Sharing
  - OIP-SR: Conventional SimRank + Partial Sums Sharing
  - psum-SR [VLDB J. ’10]: Without Partial Sums Sharing
  - mtx-SR [EDBT ’10]: Matrix-based SimRank via SVD

- Evaluations
  - Efficiency: CPU time, memory space, convergence rate
  - Effectiveness: relative order preservation of OIP-DSR

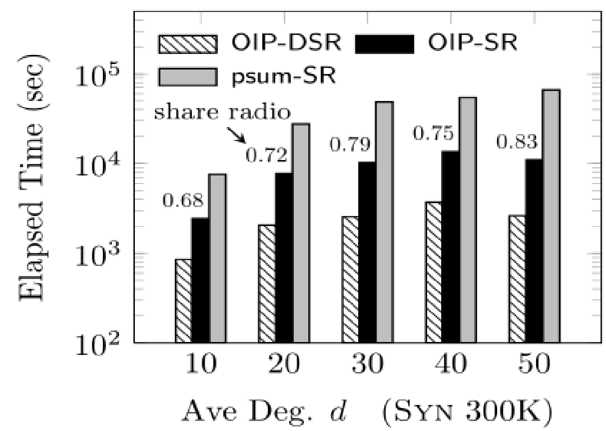
Time Efficiency



(a) Time Efficiency on Real Datasets

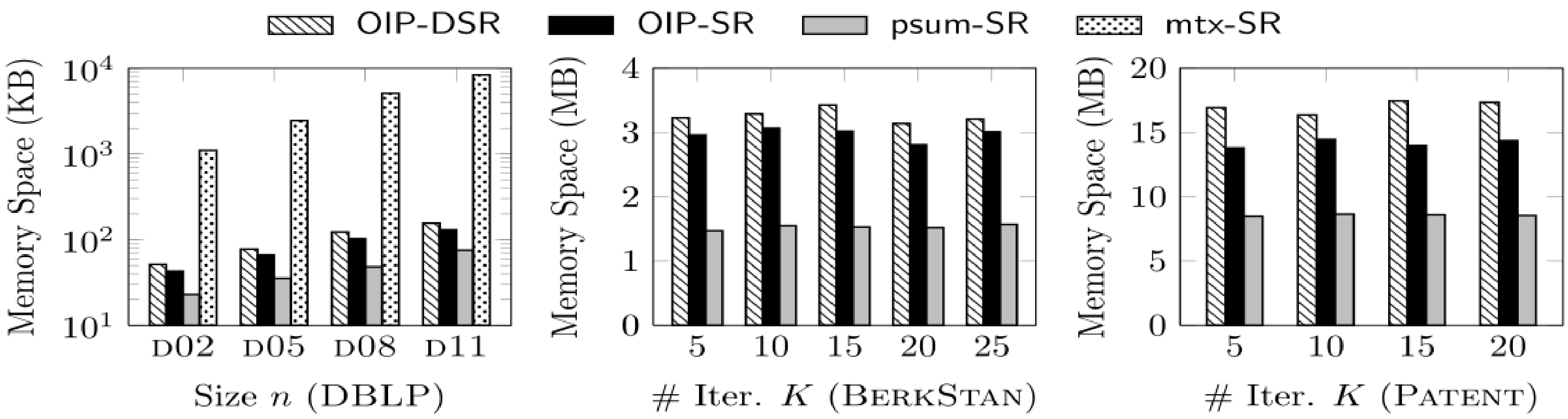


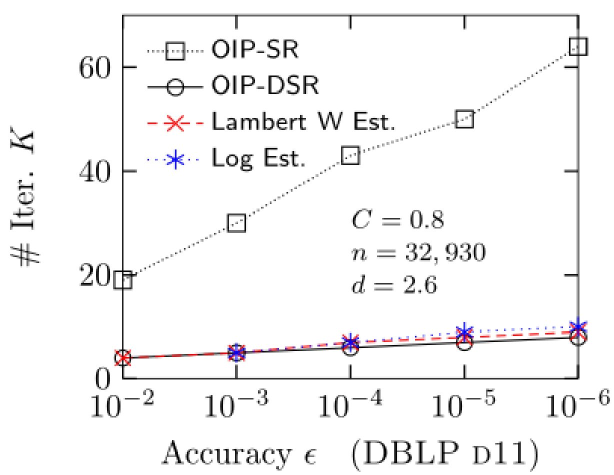
(b) Amortized Time on Real Data



(c) Effect of Density

Space & Convergence Rate



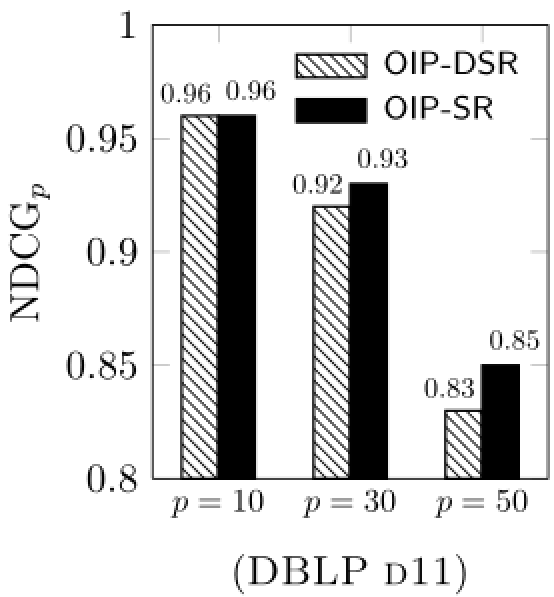


(e) Convergence Rate

Err ϵ	OIP-SR	OIP-DSR	LamW Est.	Log Est.
10^{-2}	19	4	4	-
10^{-3}	30	5	5	5
10^{-4}	43	6	7	7
10^{-5}	50	7	8	9
10^{-6}	64	8	9	10

(f) Lam W & Log Bound on K

Relative Order Preservation



(g) Relative Ordering

#	Co-authors	#	Co-authors
1	Hongjun Lu	16	Aoying Zhou
2	Lu Qin	17	Xiang Lian
3	Xuemin Lin	18	Cheqing Jin
4	Wei Wang	19	Baichen Chen
5	Lei Chen	20	Byron Choi
6	Lijun Chang	21	Wenfei Fan
7	Yiping Ke	22	Rong-Hua Li
8	Haifeng Jiang	23	**Hong Cheng ▼**
9	Philip S. Yu	24	**Jun Gao ▲**
10	Gabriel Pui Cheong Fung	25	Xiaofang Zhou
11	James Cheng	26	Ke Yi
12	Weifa Liang	27	Yufei Tao
13	Ying Zhang	28	Nan Tang
14	Bolin Ding	29	Jinsoo Lee
15	Haixun Wang	30	Kam-Fai Wong

(h) Top-30 Co-authors of "Jeffrey Xu Yu"

Conclusions

- Two efficient methods are proposed to speed up the computation of SimRank on large graphs.
  - A novel clustering approach to eliminate duplicate computations among the partial summations.
  - A differential SimRank model for achieving an exponential convergence rate.
- Empirical evaluations to show the superiority of our methods by one order of magnitude.

Thank you!

Q/A