Fast Personalized PageRank On MapReduce

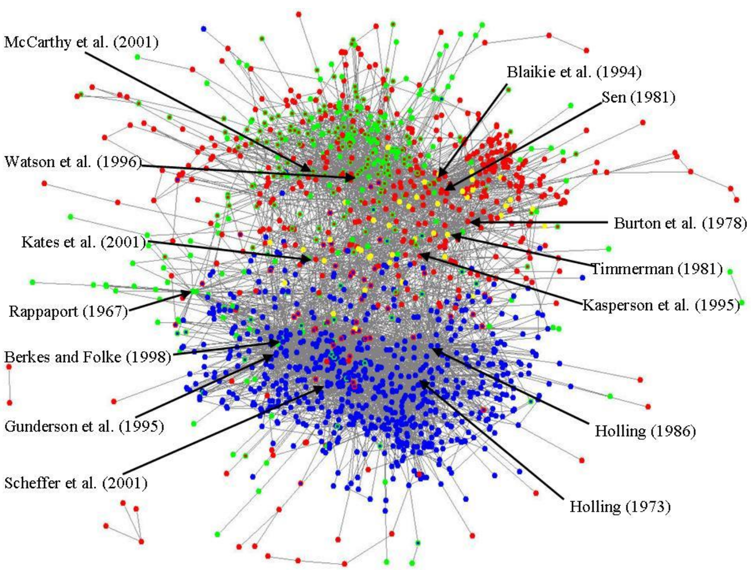
Authors: Bahman Bahmani, Kaushik Chakrabart, Dong Xin

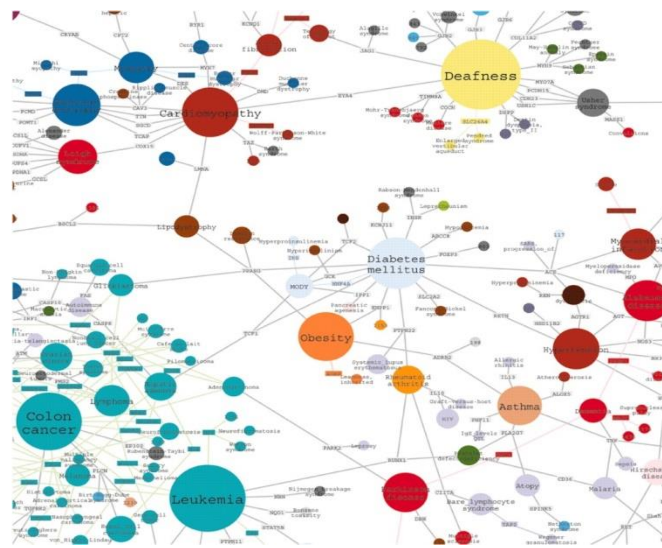
In SIGMOD 2011

Presenter: Adams Wei Yu

March 2015, CMU

Graph data is Ubiquitous





Basic Problem in Graphs: How do we measure the proximity (similarity) between two nodes?

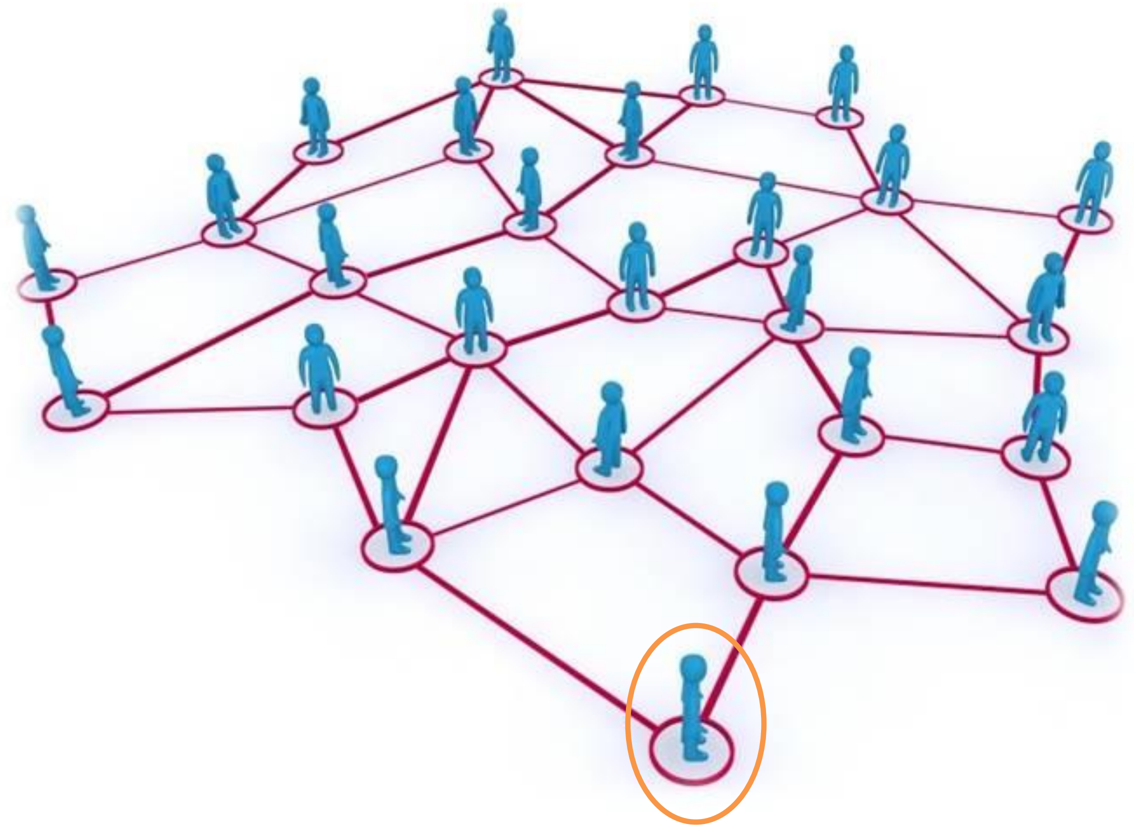
Typical proximity measure

- Shortest distance
- Common neighbor set

Typical proximity measure

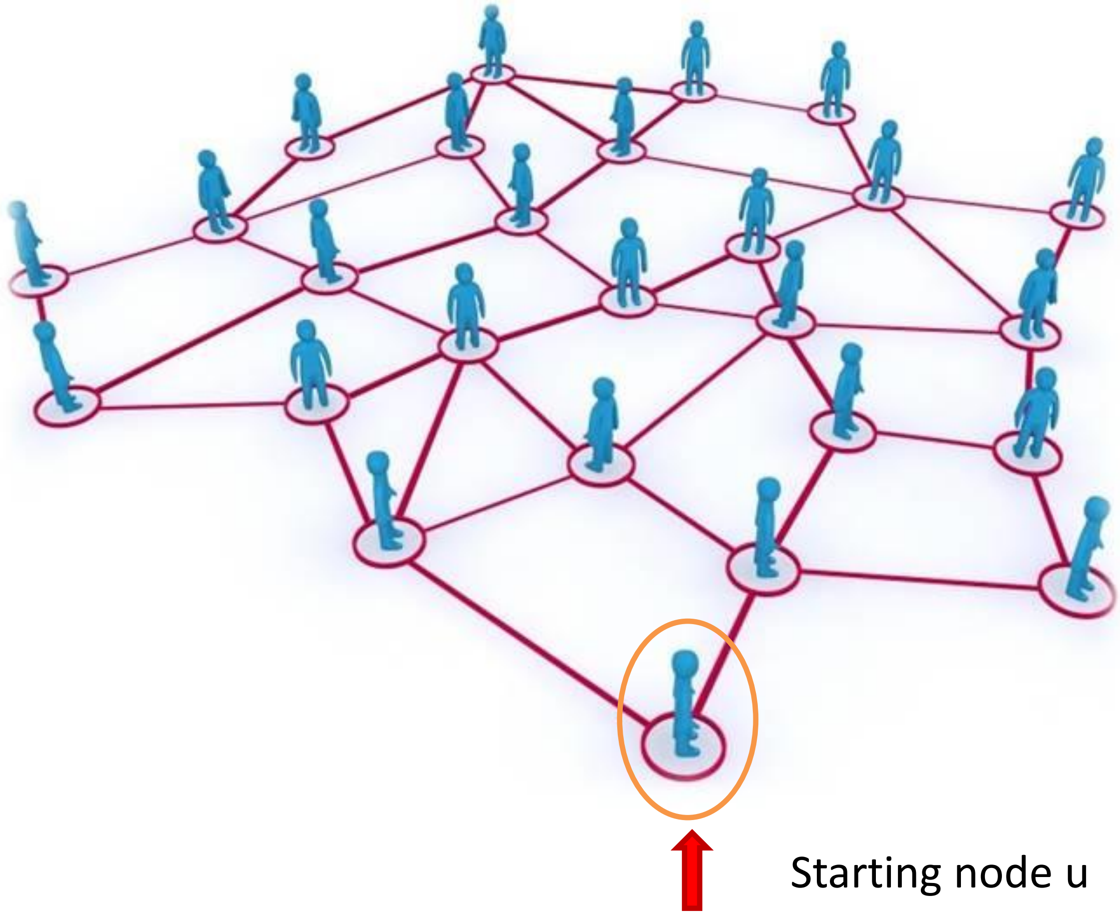
- Shortest distance
- Common neighbor set
- **Personalized PageRank (PPR), a.k.a Random Walk with Restart**
  - Considers all the possible paths from node to node
  - Captures the global structure of the graph

Personalized PageRank



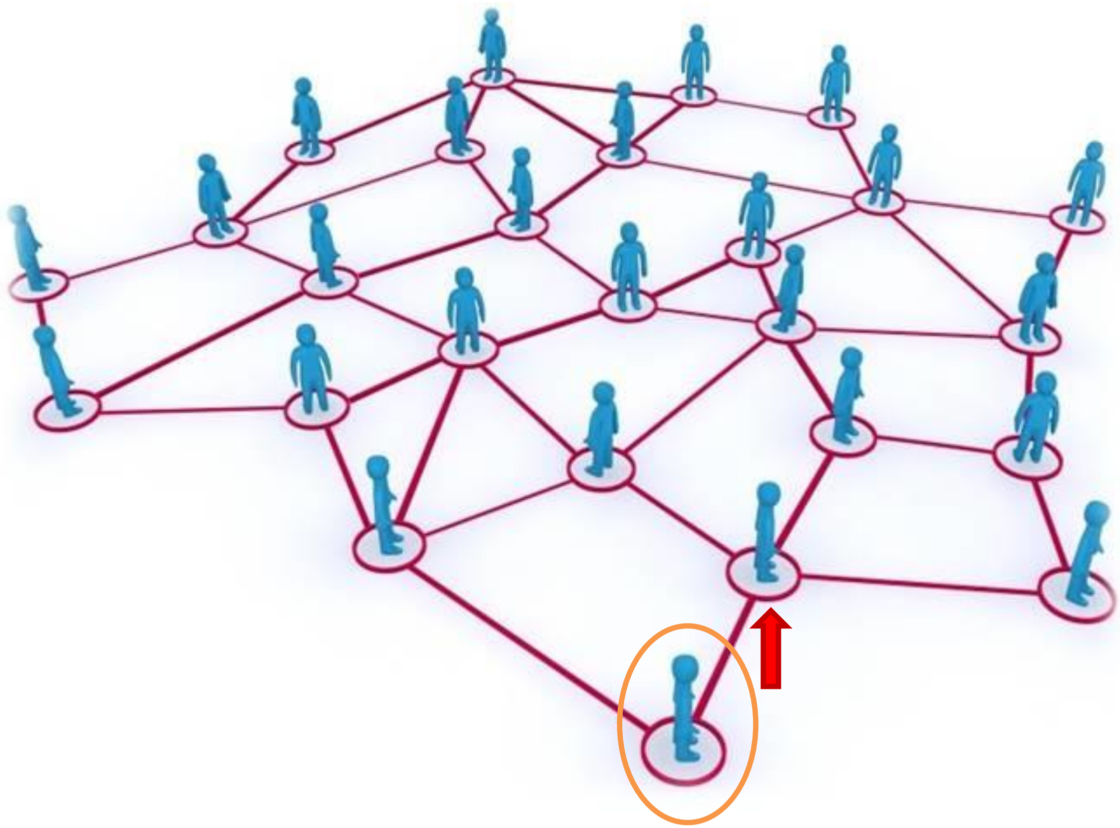
Starting node u

Personalized PageRank



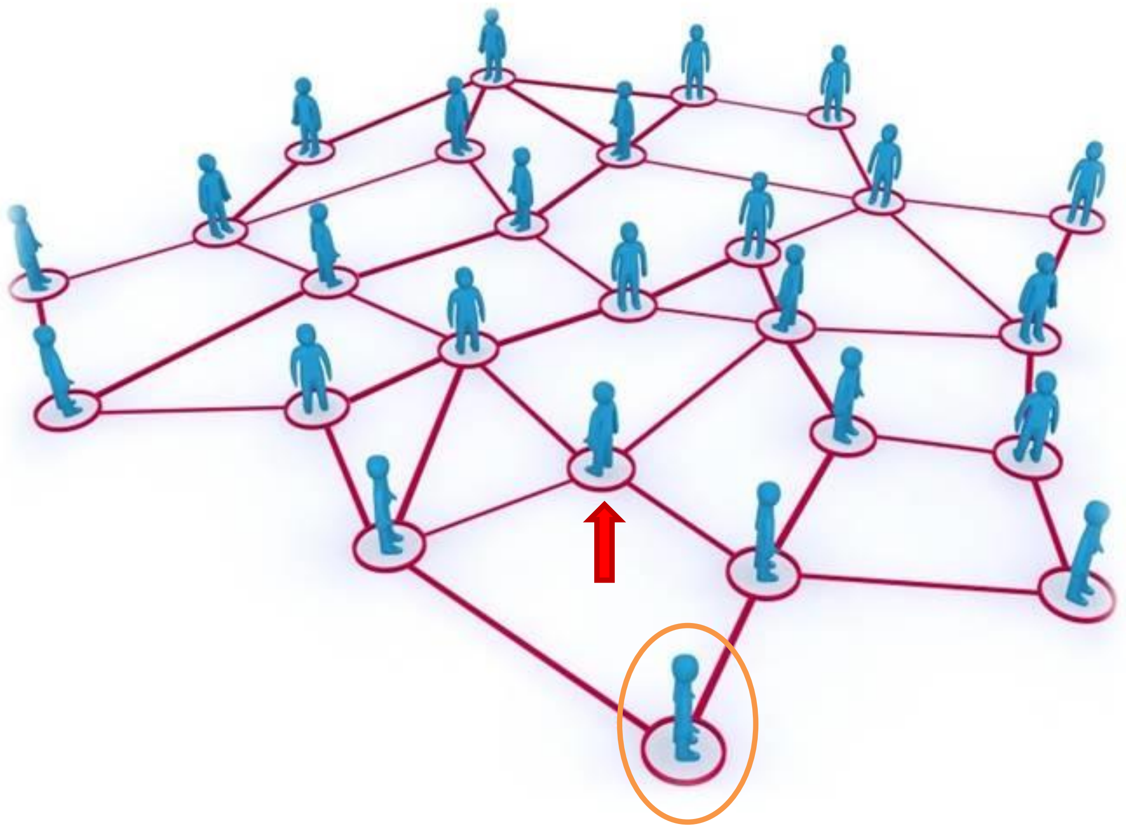
Starting node u

Personalized PageRank



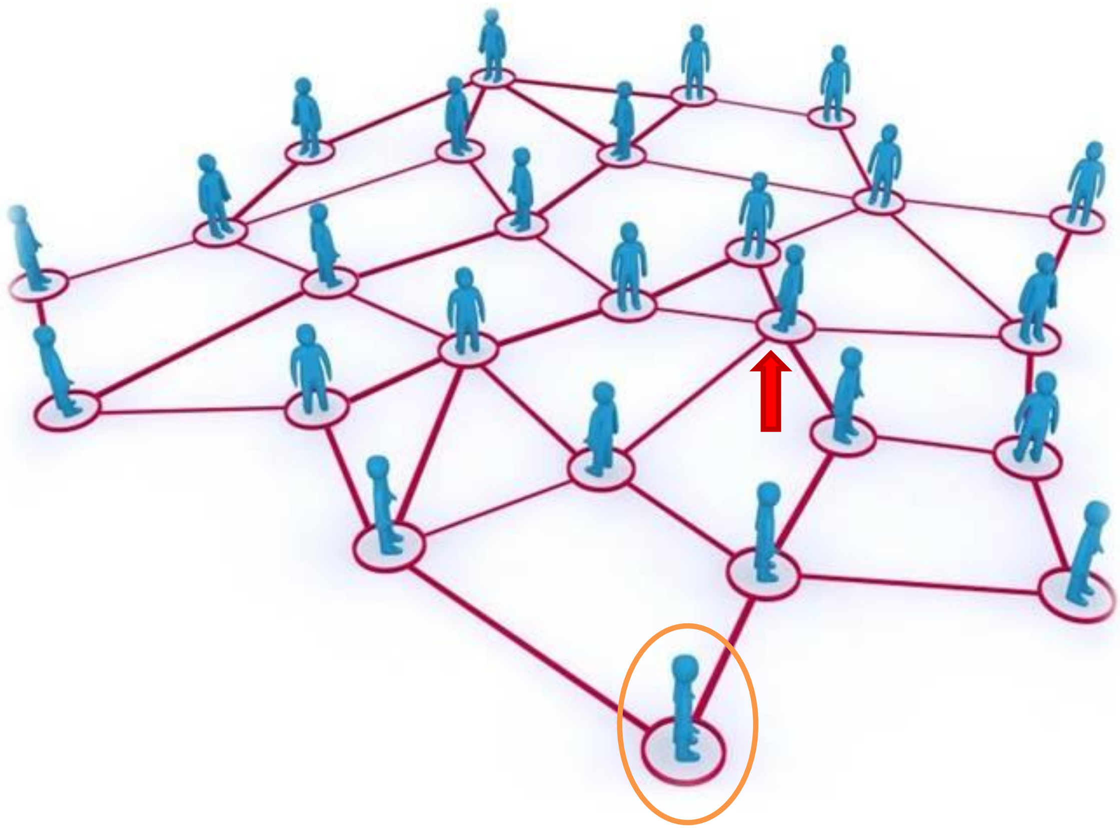
Starting node u

Personalized PageRank



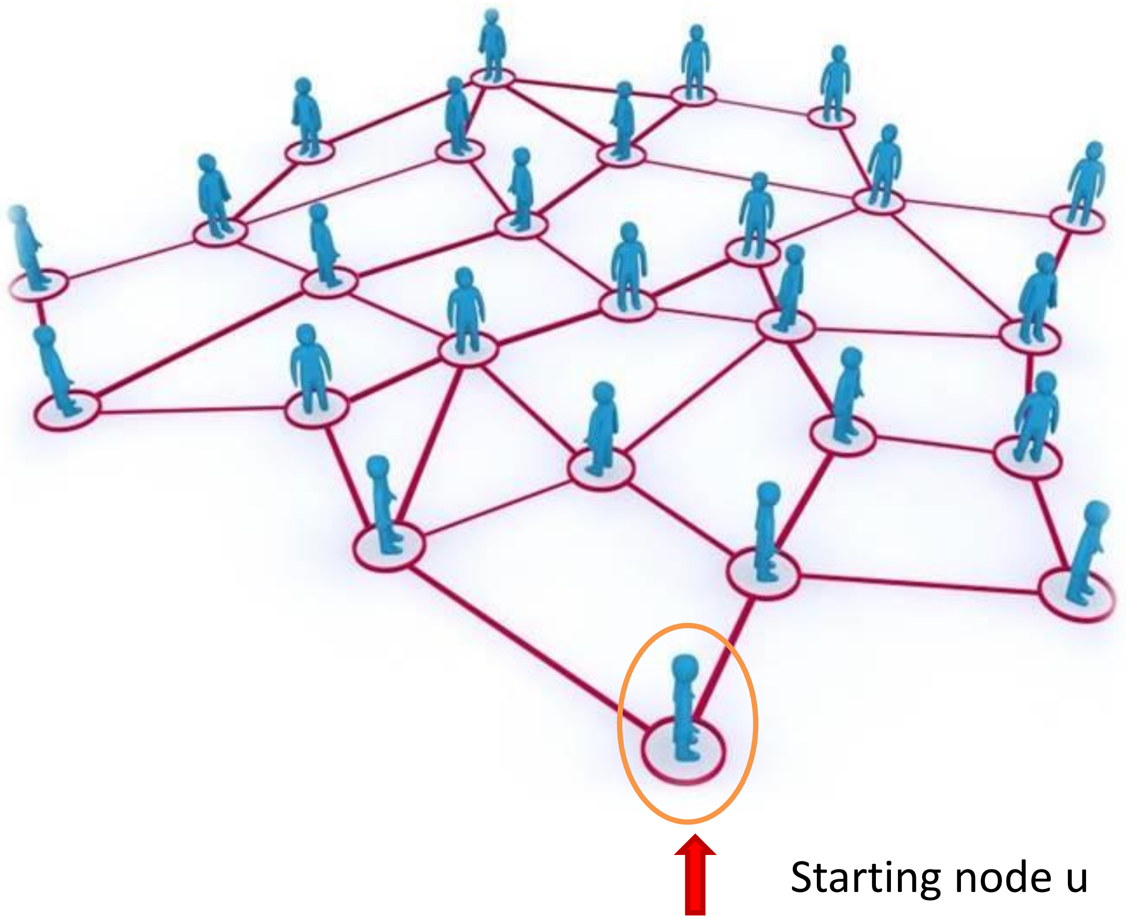
Starting node u

Personalized PageRank



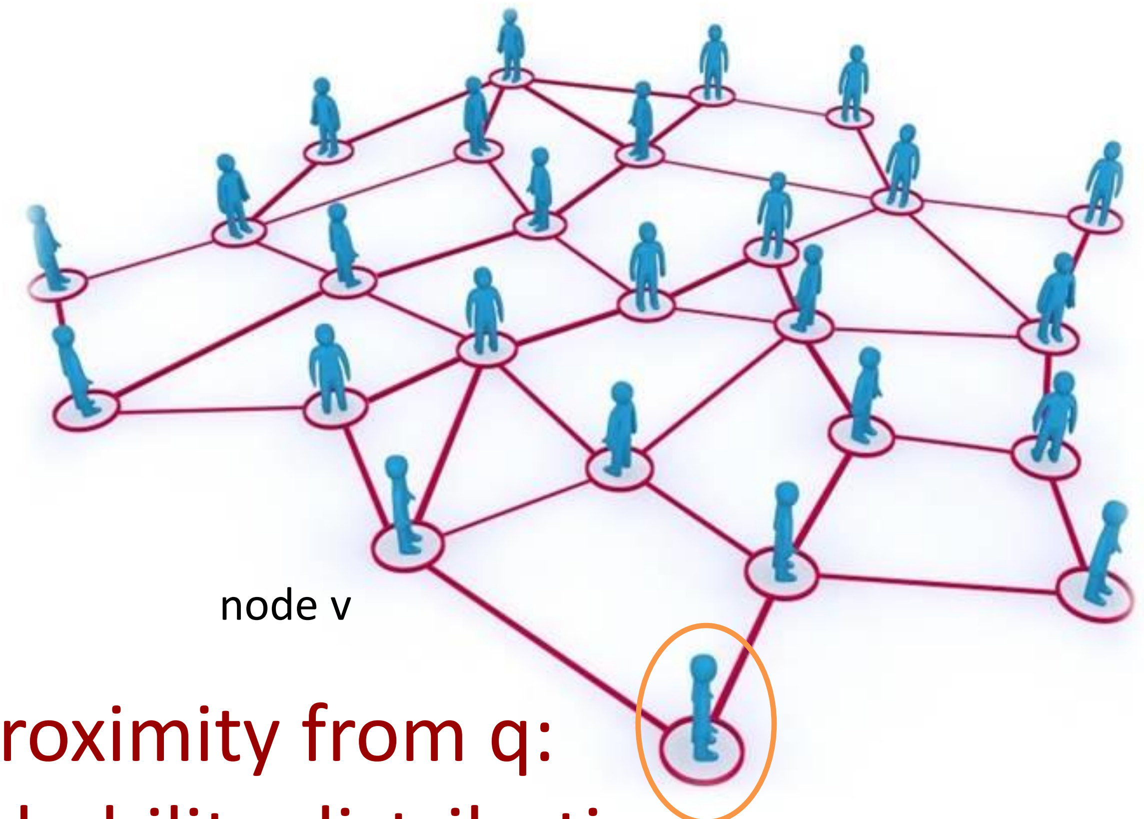
Starting node u

Personalized PageRank



Starting node u

Personalized PageRank



PPR proximity from q:
Stable probability distribution

Personalized PageRank

restart probability

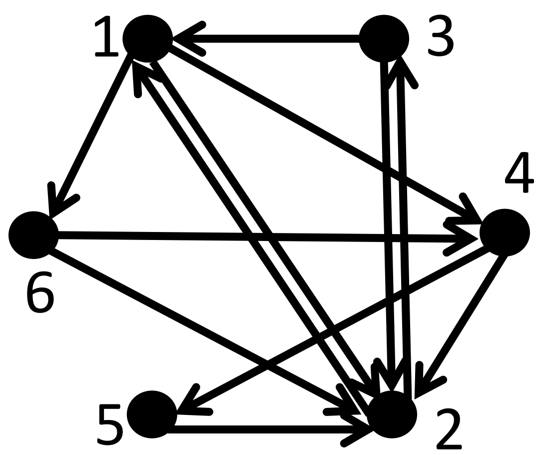
starting vector

PPR vector

$$\pi_u(v) = \epsilon\delta_u(v) + (1-\epsilon)\sum_{\{w|(w,v)\in E\}} \pi_u(w)\alpha_{w,v}$$

column normalized adjacent matrix

PPR Proximity Matrix



1	2	3	4	5	6
0.32	0.24	0.24	0.19	0.20	0.18
0.28	0.39	0.29	0.31	0.33	0.30
0.12	0.17	0.27	0.13	0.14	0.13
0.13	0.10	0.10	0.23	0.08	0.14
0.06	0.04	0.04	0.10	0.18	0.06
0.09	0.07	0.07	0.05	0.06	0.20

Application of PPR

- Citation analysis [Jeh, KDD 02]
- Link prediction [Liben Nowell, CIKM 03].
- Graph clustering [Andersen, FOCS 06].
- Recommendation system [Konstas, SIGIR 09].
- Top-k search [Fujiwara, VLDB 12]
- Reverse Top-k Search [Yu, VLDB 14]
- ……

How to Compute the PPR?

1	2	3	4	5	6
0.32	0.24	0.24	0.19	0.20	0.18
0.28	0.39	0.29	0.31	0.33	0.30
0.12	0.17	0.27	0.13	0.14	0.13
0.13	0.10	0.10	0.23	0.08	0.14
0.06	0.04	0.04	0.10	0.18	0.06
0.09	0.07	0.07	0.05	0.06	0.20

?

MapReduce for PPR Computation

- Reducer: Compute random neighbor of u

MapReduce for PPR Computation

- Reducer: Compute random neighbor of u
  - G = ⟨u, v, weight⟩. Reducer uses u as the key, groups all ⟨u, v, weight⟩ triples (for all v), and generating one random neighbor v for u and outputs ⟨u, v⟩.

MapReduce for PPR Computation

- Reducer: Compute random neighbor of u
  - G = ⟨u, v, weight⟩. Reducer uses u as the key, groups all ⟨u, v, weight⟩ triples (for all v), and generating one random neighbor v for u and outputs ⟨u, v⟩.
- Combiner: Extend N = ⟨u, v⟩ to ⟨u, v, t⟩.

MapReduce for PPR Computation

- Reducer: Compute random neighbor of u
  - G = ⟨u, v, weight⟩. Reducer uses u as the key, groups all ⟨u, v, weight⟩ triples (for all v), and generating one random neighbor v for u and outputs ⟨u, v⟩.
- Combiner: Extend N = ⟨u, v⟩ to ⟨u, v, t⟩.
  - Find a random neighbor t of v by joining N and G on condition N.v = G.u from the graph G and output ⟨u, v, t⟩.

Computation of PPR

- Power Iteration:

$$\pi_u^{(i)}(v) = \epsilon\delta_u(v) + (1-\epsilon) \sum_{\{w|(w,v)\in E\}} \pi_u^{(i-1)}(w)\alpha_{w,v}$$

Computation of PPR

- Power Iteration:

$$\pi_u^{(i)}(v) = \epsilon\delta_u(v) + (1-\epsilon) \sum_{\{w|(w,v)\in E\}} \pi_u^{(i-1)}(w)\alpha_{w,v}$$

- Basic MapReduce:
  - Given a graph $G = \langle u, v, \alpha_{u,v} \rangle$, the initialization of $\overrightarrow{\pi_u}^{(0)}$ is a Reducer of graph on key u.
  - Update of $\overrightarrow{\pi_u}^{(i)}$ can be implemented as combiner joining $\overrightarrow{\pi_u}^{(i-1)}$ and G.

Computation of PPR

- Power Iteration:

$$\pi_u^{(i)}(v) = \epsilon\delta_u(v) + (1-\epsilon) \sum_{\{w|(w,v)\in E\}} \pi_u^{(i-1)}(w)\alpha_{w,v}$$

- Basic MapReduce:
  - Given a graph $G = \langle u, v, \alpha_{u,v} \rangle$, the initialization of $\overrightarrow{\pi_u}^{(0)}$ is a Reducer of graph on key u.
  - Update of $\overrightarrow{\pi_u}^{(i)}$ can be implemented as combiner joining $\overrightarrow{\pi_u}^{(i-1)}$ and G.

Space: O(n^2)!

We may not need the exact PPR for most tasks!

Computation of PPR

- Monte Carlo Simulation

  Simulate R random walks starting from u,
  the portion of visits to v is approximately $\pi_u(v)$

Computation of PPR

- Monte Carlo Simulation

  Simulate R random walks starting from u,
  the portion of visits to v is approximately $\pi_u(v)$

- Basic MapReduce:
  - A Reducer to initialize R random walks from u.
  - A sequence of Combiner iterations to extend each random walk until it restarts at u.
  - A final Reducer to aggregate the frequencies of visits to every node v in all R walks (for each source node), and approximate the PPR values.

Computation of PPR

- Monte Carlo Simulation

  Simulate R random walks starting from u, the portion of visits to v is approximately $\pi_u(v)$

- Basic MapReduce:
  - A Reducer to initialize R random walks from u.
  - A sequence of Combiner iterations to extend each random walk until it restarts at u.
  - A final Reducer to aggregate the frequencies of visits to every node v in all R walks (for each source node), and approximate the PPR values.

Needs to execute the Combiner many times for long walks!

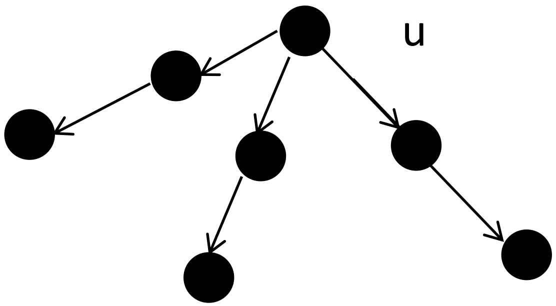
Computation of PPR

- Monte Carlo Simulation

Given a graph G and a length λ, outputs one random walk of length λ starting from each node in the graph.

Basic Idea

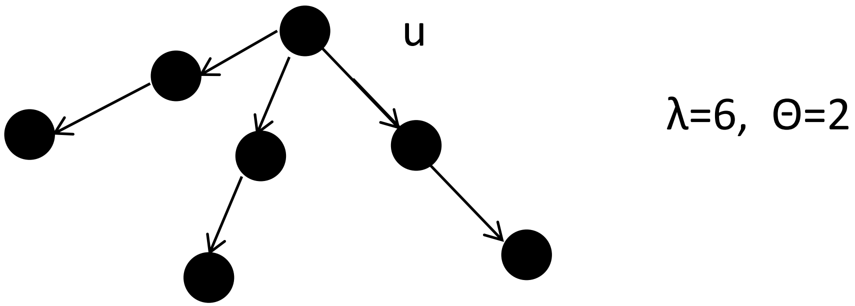
- Preprocessing: Sample $\eta=\lambda/\theta$ short segments of length θ out of each node.
- Online Computation: Merge these segments to form the longer walk.



$\lambda=6$, $\Theta=2$

Basic Idea

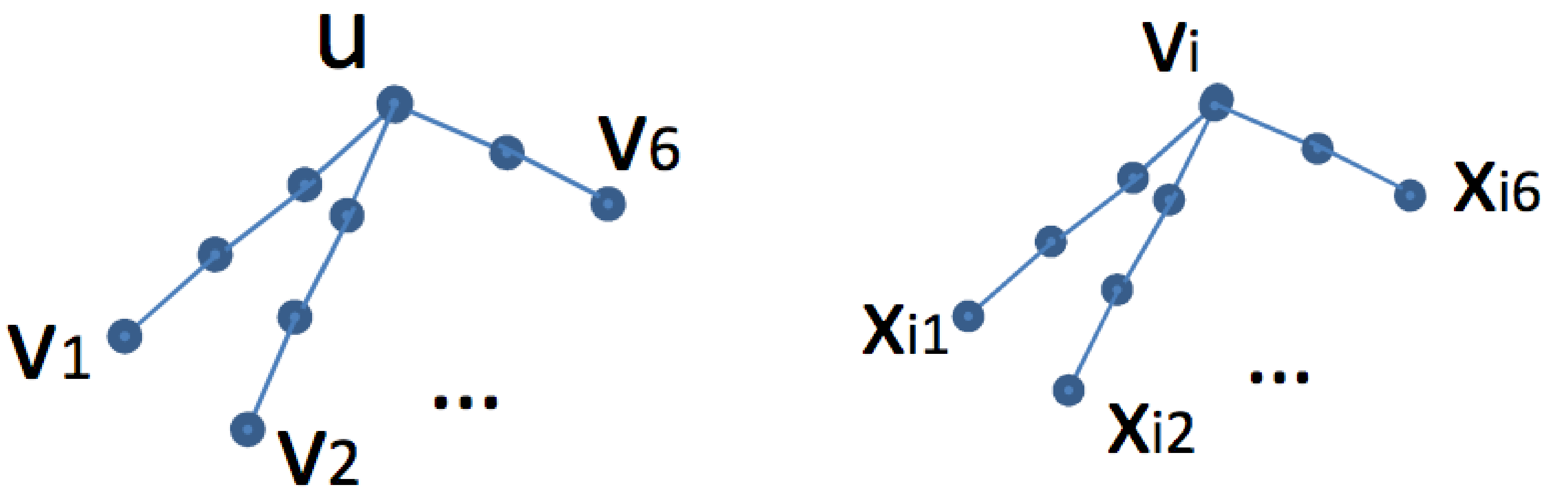
- Preprocessing: Sample $\eta=\lambda/\theta$ short segments of length θ out of each node.
- Online Computation: Merge these segments to form the longer walk.



1. How to ensure it is a REAL random walk?
2. How to make it efficient?

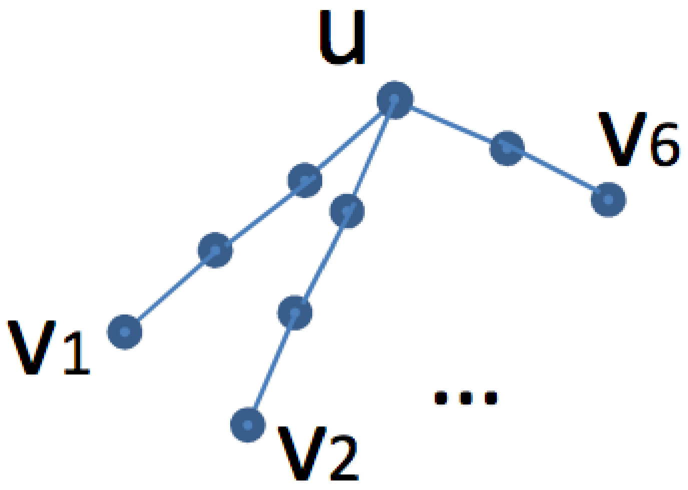
Doubling Algorithm: Initialization

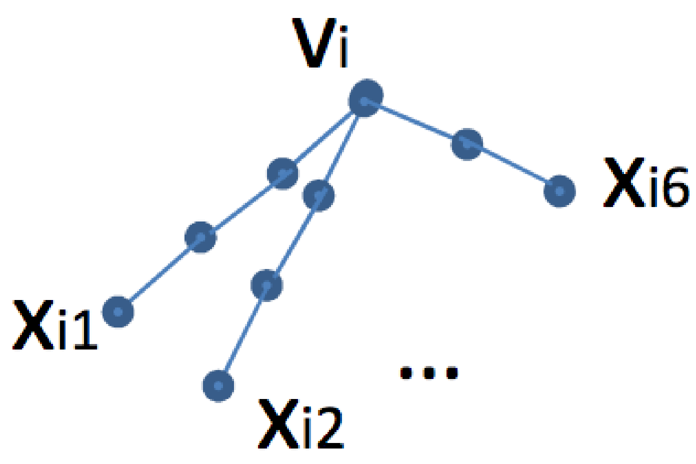
- Assume $\lambda = 17$, $\theta = 3$.
- For each node u, generate $\eta = \lceil 17/3 \rceil = 6$ segments S[u, i] ($1 \le i \le 6$), i is its ID.
- S[u, 6] is of length 2 while the other segments S[u, i] ($i < 6$) are of length 3.



Doubling Algorithm: Initialization

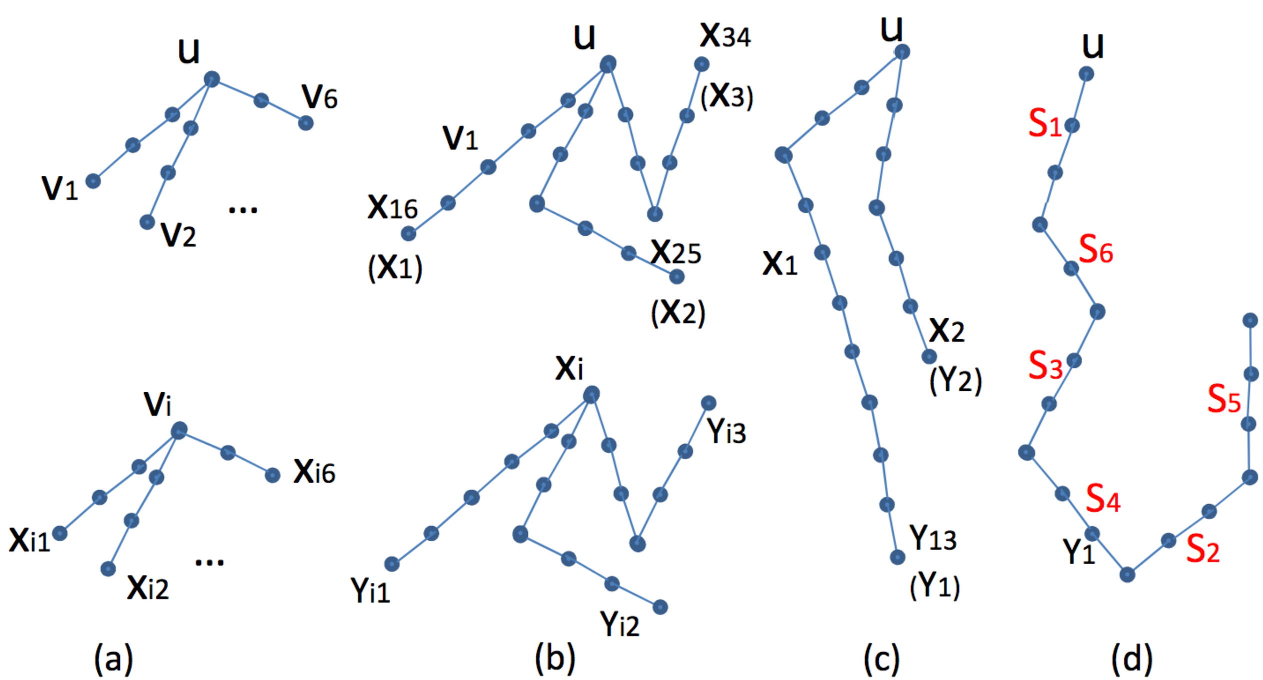
- S[u, i]: i-th segment of u
- W[u, i, η]: RW from u, η is the maximum ID at the current iteration.
- In the beginning, η = 6, and W [u, i, η] = S[u, i] for i = 1, . . . , 6.





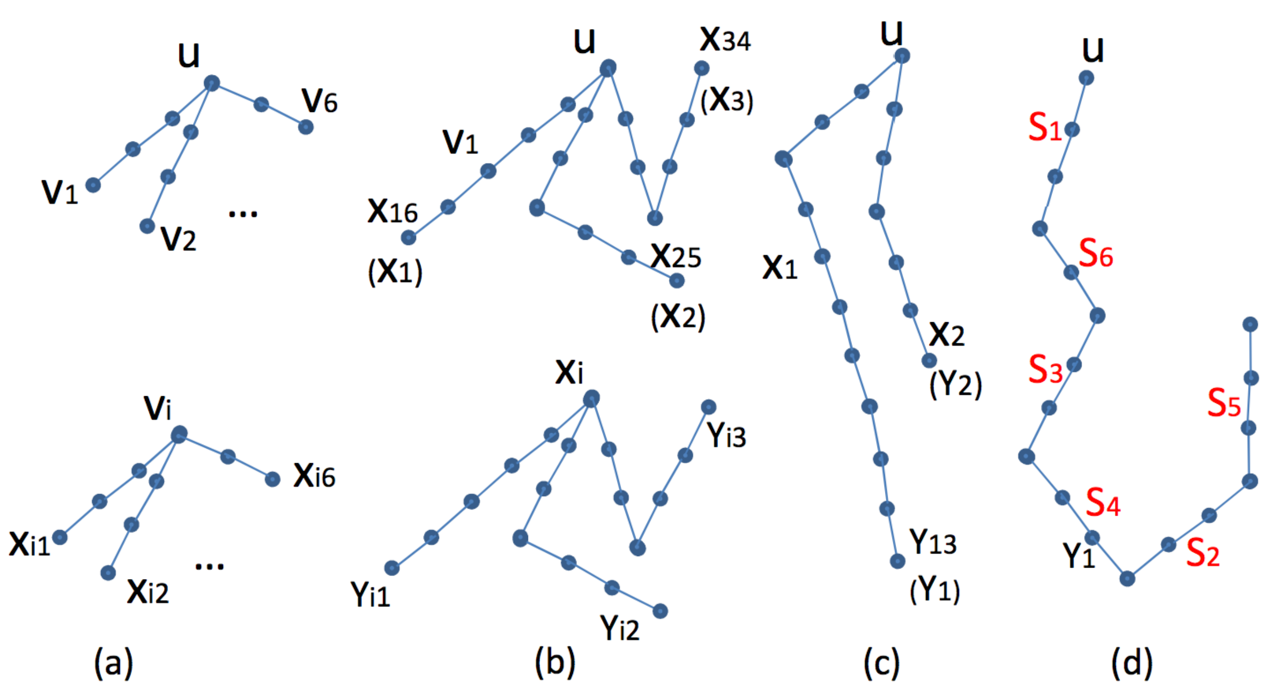
Doubling Algorithm: Merge

- Appends W2 to W1 if:
  - W1.LastNode = W2.FirstNode
  - W1.ID < W2.ID
  - W1.ID + W2.ID = η + 1. (ensure each segment is a proper random walk)



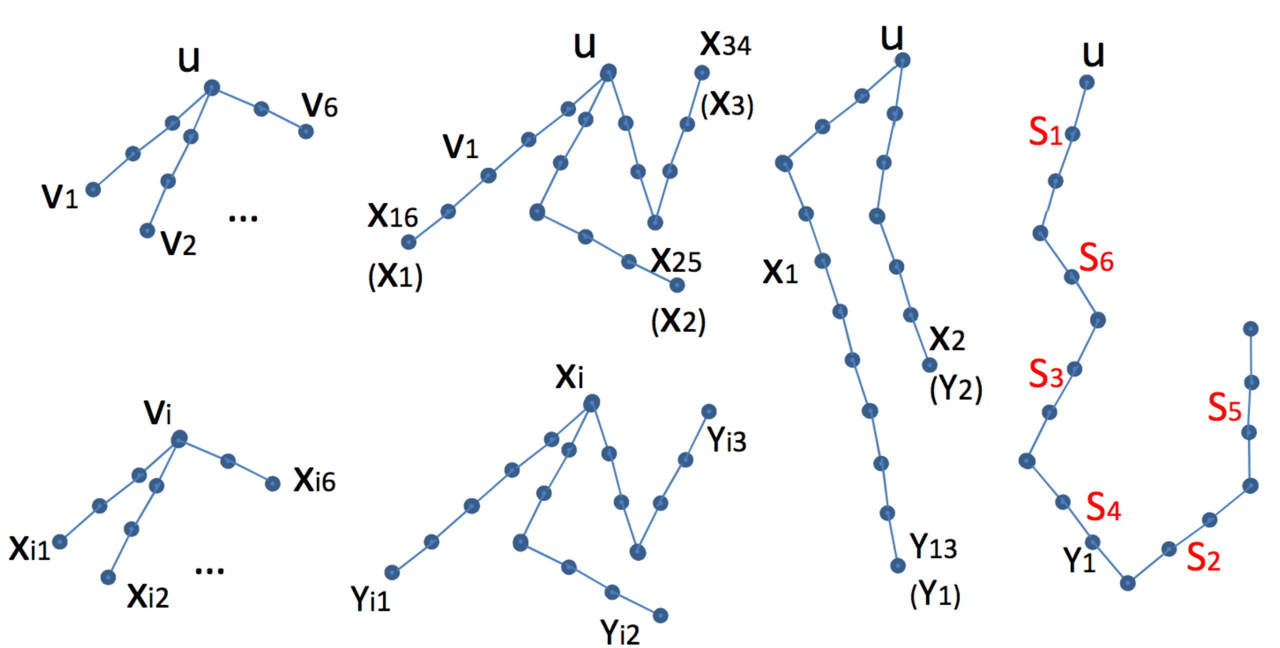
Doubling Algorithm: Merge

- For a node u, we merge W[u,i,6] with W[vi,7−i,6] for i = 1,2,3, and get 3 new segments:
  - W[u,1,3] that ends at x1, W[u,2,3] that ends at x2, and W[u,3,3] that ends at x3



Doubling Algorithm: Merge

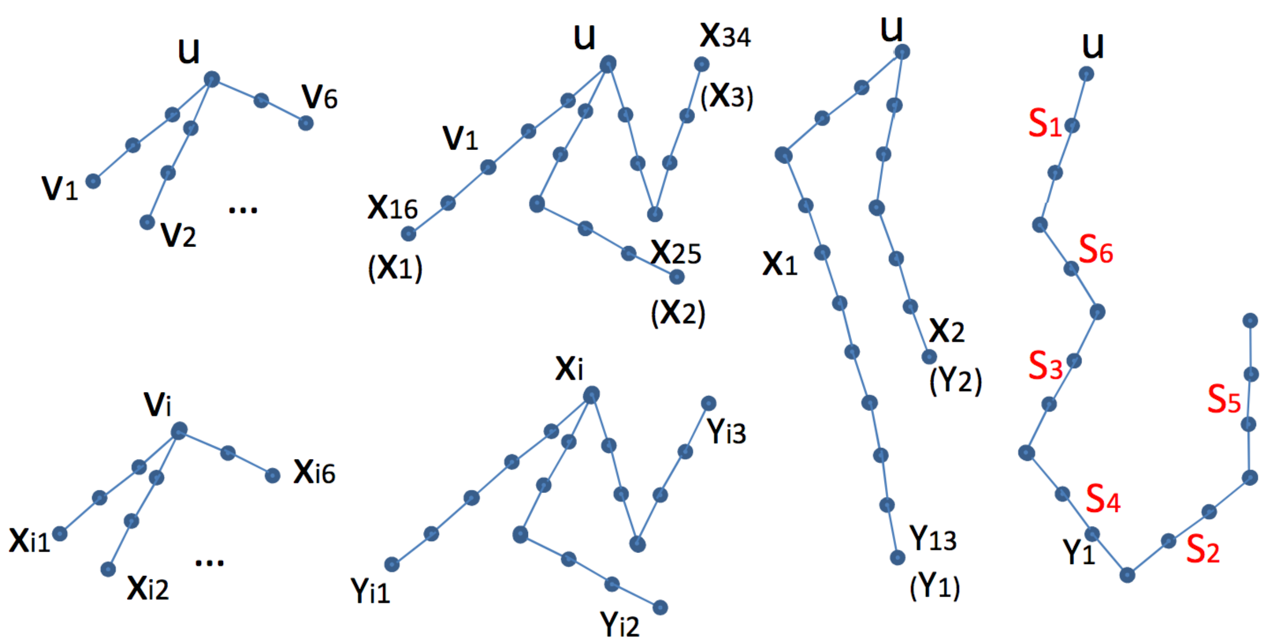
- Continue: η = 3, and we will merge W[u,1,3] with W[x1,3,3]. See Fig (c).
- Finally: merge W[u,1,2] with W[y1,2,2], and get W[u,1,1], which has length λ = 17.



(a) (b) (c) (d)

Doubling Algorithm: Merge

Shrinkage fast: One merging iteration will reduce the maximum ID from η to $\lfloor \frac{\eta+1}{2} \rfloor$!



(a) (b) (c) (d)

Optimality of the Algorithm

- Theorem: The Doubling algorithm with parameters λ, θ finishes in $\theta + \lceil \log_2 \lceil \frac{\lambda}{\theta} \rceil \rceil$ MapReduce iterations.

$$\boxed{1 + \lceil \log_2 \lambda \rceil \quad \text{If } \theta = 1}$$

Optimal for in the class of Natural Algorithms(NA)!

NA only allows Extend and Merge operations.

Computing the PPR

- Choose different λ, θ and R for different nodes.
- Repeat Doubling algorithm.
- Collect and count number of visits.
  - C(u,v)+ = number of visits to v in Wi[u]

I/O cost of the Algorithm

$$O(m \max_{1 \leq i \leq R}\{\theta_i\} + n \sum_{i=1}^{R}(\lambda_i \theta_i + \lambda_i \log_2 \frac{\lambda_i}{\theta_i}))$$

Experiments

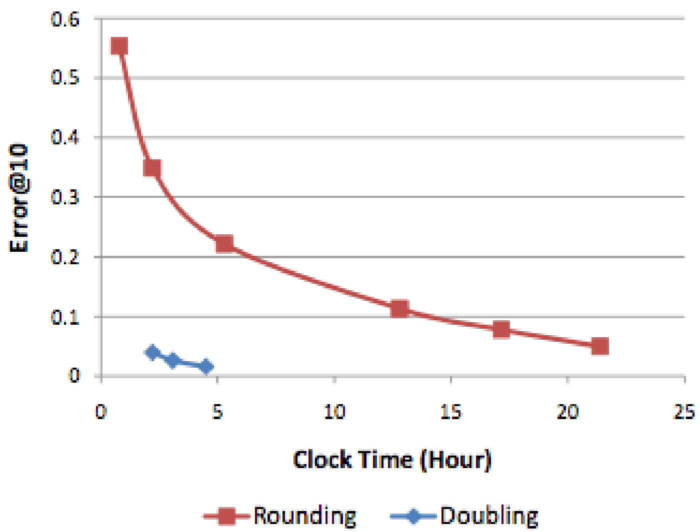


Figure 2: $\overline{Err}(.,10)$ **vs. Clock Time**

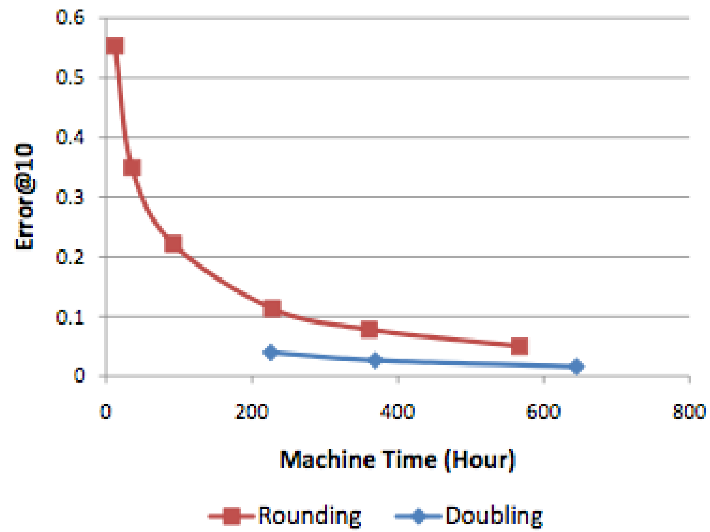


Figure 3: $\overline{Err}(.,10)$ **vs. Machine Time**

Graph Size: 112M nodes and 513M edges

Take-home Message

- A new PPR computation method on MapReduce is proposed.
- Theoretically sound
  - Optimal in the class of natural algorithms.
  - Manageable I/O cost.
- Empirically works well.

Thanks!