FAST-PPR: Personalized PageRank Estimation for Large Graphs

Peter Lofgren (Stanford)
Joint work with Siddhartha Banerjee (Stanford),
Ashish Goel (Stanford), and C. Seshadhri (Sandia)
Motivation: Personalized Search
Motivation: Personalized Search

Re-ranked by PPR

<table>
<thead>
<tr>
<th>PPR</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.75236E-4</td>
<td>Adam Messinger</td>
<td>CTO @twitter</td>
</tr>
<tr>
<td>3.65838E-4</td>
<td>Adam D'Angelo</td>
<td>CEO of Quora</td>
</tr>
<tr>
<td>2.2774E-5</td>
<td>Adam Satariano</td>
<td>Technology Reporter, Bloomberg News</td>
</tr>
<tr>
<td>1.907E-5</td>
<td>Adam Steltzner</td>
<td>Rocket scientist, intermittent gardener</td>
</tr>
<tr>
<td>8.789E-6</td>
<td>Adam Rugel</td>
<td>Hello</td>
</tr>
<tr>
<td>8.342E-6</td>
<td>AdamSerwer</td>
<td>Reporter @msnbc. I like cats and nerd stuff. I also fight crime. Mostly loitering. <a href="mailto:adam.serwer@nbculi.com">adam.serwer@nbculi.com</a> tinyletter.com/adserwer</td>
</tr>
</tbody>
</table>
Result Preview

Running Time on Twitter-2010

- Fast-PPR: 2 sec
- Monte-Carlo: 6 min
- Local-Update: 1.2 hour
Personalized PageRank

Given source $s$, target $t$, and ‘teleport probability’ $\alpha$

- Start a random walk from $s$.
- At each step, stop with probability $\alpha$, else continue.

Then Personalized PageRank from $s$ to $t$ is given by:

$$\pi_s(t) = \mathbb{P} \left[ \text{Walk from } s \text{ stops at } t \right]$$

- Equivalent to eigenvector/stationary distribution definitions
- FAST-PPR allows arbitrary starting set, e.g.
  random $s \in V \implies$ Global PageRank
  random $s \in S \implies$ personalize to $S$
Goal
Given $\alpha$, start node $s$, single target node $t$, threshold $\delta$, estimate

$$\pi_s(t)$$

when $\pi_s(t) > \delta$

- Natural primitive for personalized search
- Want only $\pi_s(t)$, not entire $\pi_s$ vector.
- Since average of $\pi_s$ is $\frac{1}{n}$, we want $\delta \sim \frac{1}{n}$. For real-time, running time must be $\ll \frac{1}{\delta}$
Previous Algorithm: Monte-Carlo

[Avrachenkov, et al 2007]

Sample $O\left(\frac{1}{\delta}\right)$ random walks from $s$, and return estimate

$$\hat{\pi}(s, t) = \text{Fraction of walks ending at } t.$$

Running time:

$$\Theta\left(\frac{1}{\delta}\right)$$
Previous Algorithm: Local Update

- Works from target $t$ backwards along edges, updating Personalized PageRank estimates locally.

- Tight average running time $O\left(\frac{\bar{d}}{\delta}\right)$ where $\bar{d} = \frac{|E|}{|V|}$
Main Result

Theorem 1. Given $s$, $t$, and $\delta$, if $\pi_s(t) > \delta$ then estimate $\hat{\pi}_s(t)$ satisfies $|\pi_s(t) - \hat{\pi}_s(t)| \leq 0.1 \pi_s(t)$ with probability 0.9. Furthermore, the average running time is

$$O\left(\frac{1}{\sqrt{\delta}} \sqrt{\bar{d}}\right)$$

where $\bar{d}$ is the average in-degree of the graph.

We also prove a lowerbound of $\Omega\left(\frac{1}{\sqrt{\delta}}\right)$. 
Analogy: Bidirectional Search

\[ d^l \]

\[ 2d^{l/2} = 2\sqrt{d^l} \]
Bidirectional PageRank Algorithm

Forward Work
(Random Walks)

Reverse Work
(Frontier Discovery)
Main Idea

Decomposition

\[ \pi(s, t) = \sum_{u \in F_t} \Pr[\text{Walk from } s \text{ first hits } u \in F_t] \pi(u, t) \]
Experimental Setup

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># Nodes</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP-2011</td>
<td>undirected</td>
<td>1.0M</td>
<td>6.7M</td>
</tr>
<tr>
<td>Pokec</td>
<td>directed</td>
<td>1.6M</td>
<td>30.6M</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>undirected</td>
<td>4.8M</td>
<td>69M</td>
</tr>
<tr>
<td>Orkut</td>
<td>undirected</td>
<td>3.1M</td>
<td>117M</td>
</tr>
<tr>
<td>Twitter-2010</td>
<td>directed</td>
<td>42M</td>
<td>1.5B</td>
</tr>
<tr>
<td>UK-2007-05</td>
<td>directed</td>
<td>106M</td>
<td>3.7B</td>
</tr>
</tbody>
</table>

- All three algorithms can trade-off accuracy and runtime. Use parameters with similar empirical relative error (10%), then compare running time.
Empirical Running Time

Running Time (Targets sampled by PageRank)

- FAST-PPR
- Local Update
- Monte-Carlo

Time per Query (ms)

- Dblp
- Pokec
- LJ
- Orkut
- Twitter
- UK-Web

Log Scale
Summary

- Fast-PPR estimates $\pi_s(t)$ 10x faster.
- Provable $O \left( \sqrt{\frac{d}{\delta}} \right)$ average time; previous best $\Omega \left( \frac{1}{\delta} \right)$
- $O \left( \frac{\bar{d}}{\sqrt{\delta}} \right)$ storage per node $\Rightarrow O \left( \frac{1}{\sqrt{\delta}} \right)$ worst-case time

Future Work

- Close the gap between running time and lower bound.
- Build a scalable personalized search system (in progress).
Thank You

- Paper available on Arxiv
- Code available at cs.stanford.edu/~plofgren
\[ T_t = \{ v \in V : \hat{\pi}(v, t) > \sqrt{\delta} \} \]

Frontier \( F_t = \bigcup_{v \in T_t} N^{in}(v) \)
Future Work

(In progress) Create a distributed implementation.

Create a scalable personalized search method.

Find an algorithm with average runtime $O \left( \frac{1}{\sqrt{\delta}} \right)$ or prove a lower bound of $\Omega \left( \sqrt{\frac{d}{\delta}} \right)$. 
Algorithm (Simple Version)

1. Use Local Update to compute estimates $\hat{\pi}(v, t)$ to accuracy $O(\sqrt{\delta})$.

2. Define

   Target Set $\hat{T}_t = \{v \in V : \hat{\pi}(v, t) > \sqrt{\delta}\}$

   Frontier $\hat{F}_t = \{u \in V \setminus \hat{T}_t : (u, v) \in E$ for some $v \in \hat{T}_t\}$
Algorithm (Simple Version)

3. Take $O\left(\frac{1}{\sqrt{\delta}}\right)$ Random Walks $\{W_i\}$, terminating each early if it hits $\hat{F}_t$. Define

$$X_i = \begin{cases} \hat{\pi}(u, t), & \text{if } W_i \text{ hits } u \in \hat{F}_t \\ 0, & \text{if } W_i \text{ does not hit } \hat{F}_t \end{cases}$$

4. Return empirical mean$\{X_i\}$. 
Average Running Time
For a uniformly random target node $t$, the average per-query running time is

$$O \left( \frac{1}{\sqrt{\delta}} (\bar{d} + 1) \right).$$

Improved Implementation:

$$O \left( \frac{1}{\sqrt{\delta}} \sqrt{\bar{d}} \right).$$
Local Update Algorithm

\[
\Pr [\text{end at } t \text{ from } u] = \sum_{v \in \text{Out}(u)} \Pr [\text{transition } u \rightarrow v] \Pr [\text{end at } t \text{ from } v]
\]

\[
\pi(u, v) = \sum_{v \in \text{Out}(u)} \frac{1 - \alpha}{d_{\text{out}}(u)} \pi(v, t)
\]
Local Update Algorithm

**After 0 iterations**
- **x**: \(p=0.00, s=0.00\)
- **y**: \(p=0.00, s=0.00\)
- **z**: \(p=0.00, s=0.00\)
- **v**: \(p=0.20, s=0.20\)

**After 1 iteration**
- **x**: \(p=0.00, s=0.00\)
- **y**: \(p=0.00, s=0.00\)
- **z**: \(p=0.16, s=0.16\)
- **v**: \(p=0.00, s=0.20\)
Local Update Algorithm

After 2 iterations:

- \( x \) with \( p=0.06 \) and \( s=0.06 \)
- \( z \) with \( p=0.00 \) and \( s=0.16 \)
- \( v \) with \( p=0.13 \) and \( s=0.33 \)

After 3 iterations:

- \( x \) with \( p=0.06 \) and \( s=0.06 \)
- \( z \) with \( p=0.10 \) and \( s=0.26 \)
- \( v \) with \( p=0.00 \) and \( s=0.33 \)
Algorithm 2 FRONTIER($t$, $\varepsilon_r$)

Inputs: target node $t$, reverse threshold $\varepsilon_r$, multiplicative factor $\beta$ (default: $1/6$)

1: Define additive error $\varepsilon_{inv} = \beta \varepsilon_r$
2: Initialize (sparse) estimate-vector $\hat{\pi}_t^{-1}$ and (sparse) residual-vector $r_t$ as: 
   \[ \begin{cases} 
   \hat{\pi}_t^{-1}(u) = r_t(u) = 0 & \text{if } u \neq t \\
   \hat{\pi}_t^{-1}(t) = r_t(t) = \alpha 
   \end{cases} \]
3: Initialize target-set $\hat{T}_t = \{ t \}$, frontier-set $\hat{F}_t = \{ \}$
4: while $\exists w \in V$ s.t. $r_t(w) > \alpha \varepsilon_{inv}$ do
5: for $u \in \mathcal{N}^{in}(w)$ do
6: $\Delta = (1 - \alpha) \frac{r_t(w)}{d_{out}(u)}$
7: $\hat{\pi}_t^{-1}(u) = \hat{\pi}_t^{-1}(u) + \Delta, r_t(u) = r_t(u) + \Delta$
8: if $\hat{\pi}_t^{-1}(u) > \varepsilon_r$ then
9: $\hat{T}_t = \hat{T}_t \cup \{ u \}, \hat{F}_t = \hat{F}_t \cup \mathcal{N}^{in}(u)$
10: end if
11: end for
12: $r_t(w) = 0$
13: end while
14: $\hat{F}_t = \hat{F}_t \setminus \hat{T}_t$
15: Run DE-BIAS($\hat{\pi}_t^{-1}, \hat{T}_t, \varepsilon_{inv}$)
16: return $\hat{T}_t, \hat{F}_t, (\hat{\pi}_t^{-1}(w))_{w \in \hat{F}_t \cup \hat{T}_t}$
Worst-Case Running Time

In worst case, $t$ has $\Omega(n)$ in-neighbors, and our algorithm takes $\Omega(n)$ time. If we allow ourselves precomputation time

$$O\left(n\sqrt{\frac{d\log(n)}{\delta}}\right)$$

and storage

$$O\left(n\sqrt{\frac{d\log(n)}{\delta}}\right),$$

we can guarantee a worst-case runtime of

$$O\left(\sqrt{\frac{d\log(n)}{\delta}}\right).$$
Problem Statement

Given

\(G\) Directed Graph

\(\alpha\) Teleport Probability

\(s\) start node

\(t\) target node

\(\delta\) Significance Threshold

\(c\) Approximation Ratio

Determine if

\[ \pi(s, t) \geq \delta \]

guaranteeing correct response (w. h. p.) when \(\pi(s, t) > c\delta\) or \(\pi(s, t) < \frac{1}{c}\delta\).
# Experiments

## Datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># Nodes</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP-2011</td>
<td>undirected</td>
<td>986K</td>
<td>6.7M</td>
</tr>
<tr>
<td>Pokec</td>
<td>directed</td>
<td>1.6M</td>
<td>30.6M</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>undirected</td>
<td>4.8M</td>
<td>69M</td>
</tr>
<tr>
<td>Orkut</td>
<td>undirected</td>
<td>3.1M</td>
<td>117M</td>
</tr>
<tr>
<td>Twitter-2010</td>
<td>directed</td>
<td>42M</td>
<td>1.5B</td>
</tr>
<tr>
<td>UK-2007-05</td>
<td>directed</td>
<td>106M</td>
<td>3.7B</td>
</tr>
</tbody>
</table>

\[
\delta = \frac{4}{n_1^\frac{1}{5}} \\
\alpha = \frac{1}{5} \\
s \sim \text{Uniform}, t \sim \text{Global PageRank}
\]
Outline

• Introduction to Personalized PageRank
• Previous Algorithms
  • Monte-Carlo
  • Local-Update
• New Bidirectional Algorithm
• Correctness Proof
• Future Work
Pr[end at t from u] = \sum_{v \in \text{Out}(u) \cap T_t} \Pr[\text{transition } u \to v] \Pr[\text{end at } t \text{ from } v] \\
+ \sum_{v \in \text{Out}(u) \setminus T_t} \Pr[\text{transition } u \to v] \Pr[\text{end at } t \text{ from } v]
Motivation: Personalized Search

Peter Norvig
Motivation: Friend Recommendation
Find all people named "John"

John Taggart
Stanford, California
Lives in Stanford, California
23 mutual friends including Brendan Tracey and Ted Sanders

Qifeng Chen (John Chen)
Research Assistant at Stanford University
Studies Computer Science at Stanford University '17
Lives in Stanford, California
In a relationship: Male
5 mutual friends including Sida Wang and Jeffrey Wang

John Bartel
Palo Alto, California
Read Ender's Game, Dune and The Kingkiller Chronicle
Listens to Mumford and Sons, The Lumineers and Bon Iver
Lives in Palo Alto, California
5 mutual friends including Nick Arnosti and Joshua Horowitz
Motivation: Personalized Search

How should we sort results with no mutual friends?
Correctness

Now assume imperfect Local Update estimates: $\forall u \in V$, 

$$|\hat{\pi}(u, t) - \pi(u, t)| < \beta \sqrt{\delta}$$

where $\beta$ is a constant like $\frac{1}{6}$.

Problem: if $\pi(u, t) < \beta \sqrt{\delta}$ for all $u \in F_t$, we might have $\hat{\pi}(u, t) = 0$.

However, for $u \in T_t$, $\pi(u, t) > \sqrt{\delta}$ so

$$|\hat{\pi}(u, t) - \pi(u, t)| < \frac{\beta}{1 - \beta} \pi(u, t)$$

so we only want to use $\hat{\pi}(u, t)$ for $u \in T_t$. 
Definition

Personalized PageRank:

$$\pi = (1 - \alpha)A\pi + \alpha p$$

where $p$ is a personalized distribution over nodes.

- Global PageRank results from $p = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$

- Our method handles arbitrary $p$, but for concreteness, assume $p = e_s$ for some vertex $s$. 