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

PPR proximity from q:
Stable probability distribution

Starting node u

node v
Personalized PageRank

\[ \pi_u(v) = \varepsilon \delta_u(v) + (1 - \varepsilon) \sum_{\{w | (w, v) \in E\}} \pi_u(w) \alpha_{w,v} \]

- PPR vector
- Restart probability
- Starting vector
- Column normalized adjacent matrix
PPR Proximity Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.32</td>
<td>0.24</td>
<td>0.24</td>
<td>0.19</td>
<td>0.20</td>
<td>0.18</td>
</tr>
<tr>
<td>2</td>
<td>0.28</td>
<td>0.39</td>
<td>0.29</td>
<td>0.31</td>
<td>0.33</td>
<td>0.30</td>
</tr>
<tr>
<td>3</td>
<td>0.12</td>
<td>0.17</td>
<td>0.27</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>0.13</td>
<td>0.10</td>
<td>0.10</td>
<td>0.23</td>
<td>0.08</td>
<td>0.14</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
<td>0.04</td>
<td>0.04</td>
<td>0.10</td>
<td>0.18</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>0.09</td>
<td>0.07</td>
<td>0.07</td>
<td>0.05</td>
<td>0.06</td>
<td>0.20</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.32</td>
<td>0.24</td>
<td>0.24</td>
<td>0.19</td>
<td>0.20</td>
<td>0.18</td>
</tr>
<tr>
<td>2</td>
<td>0.28</td>
<td>0.39</td>
<td>0.29</td>
<td>0.31</td>
<td>0.33</td>
<td>0.30</td>
</tr>
<tr>
<td>3</td>
<td>0.12</td>
<td>0.17</td>
<td>0.27</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
</tr>
<tr>
<td>4</td>
<td>0.13</td>
<td>0.10</td>
<td>0.10</td>
<td>0.23</td>
<td>0.08</td>
<td>0.14</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
<td>0.04</td>
<td>0.04</td>
<td>0.10</td>
<td>0.18</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>0.09</td>
<td>0.07</td>
<td>0.07</td>
<td>0.05</td>
<td>0.06</td>
<td>0.20</td>
</tr>
</tbody>
</table>

?
MapReduce for PPR Computation

- Reducer: Compute random neighbor of u
• Reducer: Compute random neighbor of u
  – $G = \langle u, v, \text{weight} \rangle$. Reducer uses $u$ as the key, groups all $\langle u, v, \text{weight} \rangle$ triples (for all $v$), and generating one random neighbor $v$ for $u$ and outputs $\langle u, v \rangle$. 
MapReduce for PPR Computation

- Reducer: Compute random neighbor of \( u \)
  - \( G = \langle u, v, \text{weight} \rangle \). Reducer uses \( u \) as the key, groups all \( \langle u, v, \text{weight} \rangle \) triples (for all \( v \)), and generating one random neighbor \( v \) for \( u \) and outputs \( \langle u, v \rangle \).

- Combiner: Extend \( N = \langle u, v \rangle \) to \( \langle u, v, t \rangle \).
MapReduce for PPR Computation

• Reducer: Compute random neighbor of u
  – $G = \langle u, v, \text{weight} \rangle$. Reducer uses $u$ as the key, groups all $\langle u, v, \text{weight} \rangle$ triples (for all $v$), and generating one random neighbor $v$ for $u$ and outputs $\langle u, v \rangle$.

• Combiner: Extend $N = \langle u, v \rangle$ to $\langle u, v, t \rangle$.
  – Find a random neighbor $t$ of $v$ by joining $N$ and $G$ on condition $N.v = G.u$ from the graph $G$ and output $\langle u, v, t \rangle$. 
Computation of PPR

• Power Iteration:

\[
\pi_u^{(i)}(v) = \epsilon \delta_u(v) + (1 - \epsilon) \sum_{\{w \mid (w, v) \in E\}} \pi_u^{(i-1)}(w) \alpha_{w,v}
\]
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 \( \pi_u^{(0)} \) is a Reducer of graph on key \( u \).
  
  – Update of \( \pi_u^{(i)} \) can be implemented as combiner joining \( \pi_u^{(i-1)} \) 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 \( \pi_u^{(0)} \) is a Reducer of graph on key \( u \).
  – Update of \( \pi_u^{(i)} \) can be implemented as combiner joining \( \pi_u^{(i-1)} \) 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.
• 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 $\lambda$, outputs one random walk of length $\lambda$ starting from each node in the graph.
Basic Idea

• Preprocessing: Sample $\eta=\lambda/\theta$ short segments of length $\theta$ out of each node.

• Online Computation: Merge these segments to form the longer walk.

![Diagram](attachment:image.png)

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

- Preprocessing: Sample $\eta=\lambda/\theta$ short segments of length $\theta$ 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 $S[u, i]$ ($1 \leq i \leq 6$), $i$ 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 $u$
- $W[u, i, \eta]$: RW from $u$, $\eta$ is the maximum ID at the current iteration.
- In the beginning, $\eta = 6$, and $W[u, i, \eta] = S[u, i]$ for $i = 1, \ldots, 6$. 

![Diagram of the algorithm](image)
Doubling Algorithm: Merge

- Appends W2 to W1 if:
  - $W1.\text{LastNode} = W2.\text{FirstNode}$
  - $W1.\text{ID} < W2.\text{ID}$
  - $W1.\text{ID} + W2.\text{ID} = \eta + 1$. (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 $x_1$, $W[u,2,3]$ that ends at $x_2$, and $W[u,3,3]$ that ends at $x_3$
Doubling Algorithm: Merge

- Continue: $\eta = 3$, 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 $\lambda = 17$. 

![Diagram showing the doubling algorithm steps](image-url)
Doubling Algorithm: Merge

Shrinkage fast: One merging iteration will reduce the maximum ID from $\eta$ to $\left\lfloor \frac{\eta+1}{2} \right\rfloor$!
• Theorem: The Doubling algorithm with parameters $\lambda$, $\theta$ finishes in $\theta + \lceil \log_2 \left( \frac{\lambda}{\theta} \right) \rceil$ MapReduce iterations.

\[ 1 + \lceil \log_2 \lambda \rceil \text{ if } \theta = 1 \]

Optimal for in the class of Natural Algorithms (NA)!
NA only allows Extend and Merge operations.
Computing the PPR

- Choose different $\lambda$, $\theta$ and $R$ for different nodes.
- Repeat Doubling algorithm.
- Collect and count number of visits.
  - $C(u, v)^+ = \text{number of visits to } v \text{ in } Wi[u]$
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

Graph Size: 112M nodes and 513M edges

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

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