Index Design for Dynamic Personalized PageRank

Amit Pathak
Soumen Chakrabarti
Manish Gupta

IIT Bombay
amit@cse.iitb.ac.in  soumen@cse.iitb.ac.in  manishg@cse.iitb.ac.in

Mar 1, 2008
Motivation

- Graph structured databases

- Find expert e from industry to review a submitted paper p
- 50–400× faster than whole-graph Pagerank
- 10–20% smaller index, > 94% accuracy
- Low index size, processing time and query time
Graph $G = (V, E)$ with edges $(u, v) \in E$

Conductance $C(v, u)$ such that $\sum_v C(v, u) = 1$

Teleport prob $1 - \alpha$ and vector $r$, $\sum_v r(v) = 1$

Personalized PageRank (PPR) for vector $r$ is

$$PPV_r = p_r = \alpha Cp_r + (1 - \alpha)r = (1 - \alpha)(I - \alpha C)^{-1}r$$

For node $v$, $r(v) = 1 \Rightarrow$ its PPV is $PPV_{\delta_v} = PPV_v$
Previous work

- ObjectRank: Connects word node $w$ to all entities where it is mentioned
- Precomputes and stores $PPV_w$ for all words $w$
- Preprocessing costs increase with increase in graph and vocabulary size
- 22000 CPU hours for 562000 words
Asynchronous Weight-Pushing Algorithm (BCA)

\[ p_r = (1 - \alpha) \left( \sum_{k \geq 0} \alpha^k C^k \right) r \]

1. \( q \leftarrow r, N_{H,r} \leftarrow 0, B_{H,r} \leftarrow 0 \)
2. \textbf{while} \( \|q\|_1 > \epsilon_{\text{push}} \) \textbf{do}
3. \quad \text{pick arg max}_u q(u) \{\text{delete-max}\}
4. \quad \hat{q} \leftarrow q(u), q(u) \leftarrow 0
5. \quad \textbf{if} \; u \in H
6. \quad \quad B_{H,r}(u) \leftarrow B_{H,r}(u) + \hat{q}
7. \quad \textbf{else}
8. \quad \quad N_{H,r}(u) \leftarrow N_{H,r}(u) + (1 - \alpha)\hat{q}
9. \quad \textbf{for} each out-neighbor \( v \) of \( u \) \textbf{do}
10. \quad \quad q(v) \leftarrow q(v) + \alpha C(v, u)\hat{q} \{\text{increase-key}\}
11. \textbf{return} \; N_{H,r} + \sum_{h \in H} B_{H,r}(h) \text{PPV}_h
Cost-benefit model

- $d$ is the dummy node; $w1, w2$ are word nodes
- $e1, e2, \cdots, e6$ are entity nodes
- Shaded area represents the work saved if $e6 \in$ hubset $H$
Modelling push algorithm performance

- Cost-benefit optimizer
- Estimate BCA running time
- Exact number of push steps
- PushActive\( (H, \delta_o, \epsilon_{push}) \)
- PathActive\( (H, \delta_o, \epsilon_{push}) \)
- Cohen’s Algorithm
- CohenActive\( (H, \delta_o, \epsilon_{push}) \) by
  \[ D = -\log \epsilon_{push} \]
Hubset Selection

- Benefit: Work saved by inclusion of node $u$ in $H$
- Cost: Space to save $PPV_u$
- Work saved for one query if $u \in H$ is estimated by a regression from CohenActive
- Work saved by $u$ over query workload is $\sum_w \tilde{f}(w) \text{WorkSaved}(H, \delta_w, u)$
Clipped $PPV_u$ storage space

- Full $PPV_h \forall h \in H$ takes huge space
- Clipping decreases size without much decrease in accuracy
- Cost-benefit optimizer needs clipped PPV size
- Estimate using power-law distribution of PPV size and modified Cohen
Hub inclusion policies

- **Large PageRank (LPR):** order by decreasing global PageRank with \( r = \tilde{1}/|V| \)
- **Naive one-shot or N1:** Chakrabarti ordered hubs \( u \) one-shot by only \( N_{H,\tilde{f}}(u) \)
- **LookAhead Progressive (LAP):** Include a fixed number of nodes with high benefit/cost into the hubset at each iteration.
Experiments

- CiteSeer corpus - 709K words, 1.1M entities, 3.7M edges
- Lucene text indices - 55, 139, 259, 378 MB resp
- 1.9M CiteSeer queries – 2.68 words/query
- Disjoint 100K train queries and 10K test queries
- Beats Chakrabarti wrt index size (10×) and query speed (10×).
  RAG, precision and τ accuracy (at rank 20) of 0.998, 0.95 and 0.94
By scaling $|H|$ at a small fraction of $|V|$, HUBRANK query times can be held independent of $|G|$.

Our index size is 9–13MB for 1994 graph.
Building PPV index

- Baseline: compute $PPV_h$ for each $h \in H$ independently using Power Iterations. Time $\propto |H|$.

- MPWH (Max PPR wrt $H$): First schedule nodes $h$ which block many “heavy” paths from other (pending) hubs; estimated by CohenActive.
Bibliography


