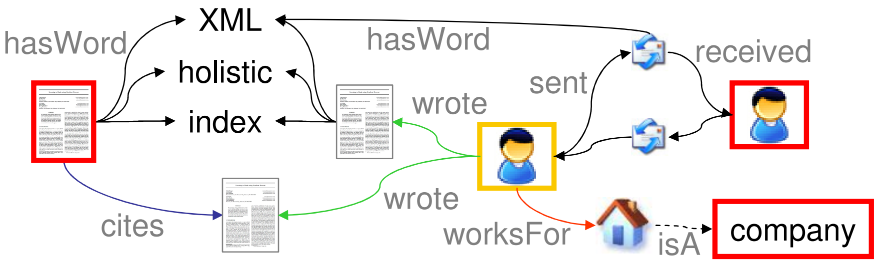
Dynamic Personalized PageRank in Entity-Relation Graphs

Soumen Chakrabarti
IIT Bombay
http://www.cse.iitb.ac.in/~soumen

Motivation: Desktop search tasks

"Find expert e from industry to review a submitted paper p"

- p shares important words with papers p' written by e
- p cites papers p' written by e
- e works for organization o is-a company
- e and I have exchanged many emails



Graph conductance queries

- Origin nodes spread activation to target nodes
- Personalized PageRank with teleport to origin nodes
- Parts of graph known only at query time, must compute PageRank dynamically
- Similar/related to many other graph search paradigms:
  - Resistive network, conductance from origin to target nodes
  - Random walk with restarts (Tong, Faloutsos, Pan)
  - Connection and centerpiece subgraphs (Faloutsos, McCurley, Tomkins, Tong)
- Naturally combines relevance and prestige
- Automatic “inverse document frequency” (IDF) effect: `"holistic"` connects to fewer papers than `"index"`

Notation

- Graph $G = (V, E)$, each edge (u, v) has a type $t(u, v)$
- E.g., "person wrote paper", "person works in company", "paper cites paper" etc.
- Edges often bidirectional to ensure activation spread, e.g., person wrote paper, paper written-by person
- Edge type t induces an edge weight $\beta(t)$
- From edge weight we get edge conductance $C(v, u) = C(u \to v) = \beta(t(u, v)) / \sum_{(u,w)\in E} \beta(t(u, w))$
- From each node, teleport with probability $1 - \alpha$
- In case of teleport, jump to node u with probability $r(u)$
- Overall PageRank equation $p_r = \alpha C p_r + (1 - \alpha) r$
- Teleport to single origin node o denoted $r = \delta_o$ and p_{δ_o} denoted PPV_o

ObjectRank

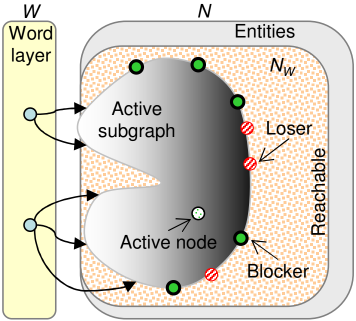
- Start with entity nodes, add query word nodes w
- Teleport to word nodes (set $r(w) > 0$) and compute p_r
- Too slow to do this for each query at query time
- Exploit linearity: $p_r = \alpha C p_r + (1-\alpha) r$ solves to $p_r = (1-\alpha)(\mathbb{I} - \alpha C)^{-1} r$, linear in r
- Therefore $p_{r_1} + p_{r_2} = p_{r_1 + r_2}$ and $p_{\gamma r} = \gamma p_r$
- Precompute and store $p_{\delta_w} = \text{PPV}_w$ for all w in vocabulary
- Given multiword query, average PPV_ws at query time

Limitations

- Long preprocessing time to compute all word PPVs
- Must truncate word PPVs arbitrarily to limit space

HubRank: ObjectRank with hubs

- Choose hub node subset $H \subset V$
- Precompute and store PPV_h for all $h \in H$
- Prepare entity graph N offline
- On query submission ...
  - Add word nodes W, link to N
  - Quickly identify query-specific active subgraph boundary (Active $\subset$ Reachable $\subset N$)
- Blockers are nodes in H whose PPVs have been precomputed and stored



- Losers are nodes too "far" from word nodes to influence word PPVs appreciably

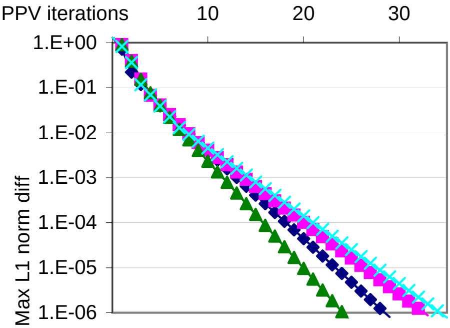
Estimating PPVs for active nodes

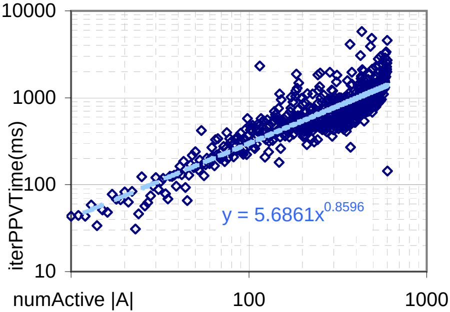
- Set $\widehat{\text{PPV}}_u = \delta_u$ for losers u
- Load approximate $\widehat{\text{PPV}}_u$ from cache for blockers u
- For active nodes u that are not blockers or losers, update

$$\widehat{\text{PPV}}_u \leftarrow \alpha \sum_{(u,v) \in E} C(v,u) \widehat{\text{PPV}}_v + (1-\alpha)\delta_u$$

  until convergence (using Decomposition Theorem)
- Can show PPV convergence similar to Jeh and Widom, even using fixed approximate $\widehat{\text{PPV}}_u$ for blockers and losers
- Add up word PPVs and report top-k entity nodes

HubRank query processing dynamics





- PPV convergence fast in practice
- Query time essentially decided by number of active nodes
- $\therefore$ critical to keep active subgraph small ...
- ... for typical and frequent queries

Performance issues in designing PPV cache

- Time to select a good $H \subset V$
- Time to precompute PPVs for nodes in H
- Space needed to store the hub cache
- Query processing time given the hub cache
- Query response accuracy wrt full ObjectRank computation

Hub selection in HubRank

Existing proposals

- Jeh and Widom: Keep large-PageRank nodes in H
- Berkhin: Teleport uniformly to personalized nodes, compute "H-relative PageRanks", pick best, update H, repeat

Not applicable because

- Teleports always go to word nodes
- $\therefore$ word nodes have large PageRank
- Too many word nodes, cannot include all in H
- If a query misses a single word PPV, it slows down drastically

Key insights

- Include judicious mix of word and entity nodes in H
- Exploit past query workload statistics to design H
- Limit PPV updates to query-specific active subgraph
- Dynamically degrade PPV resolution to save time

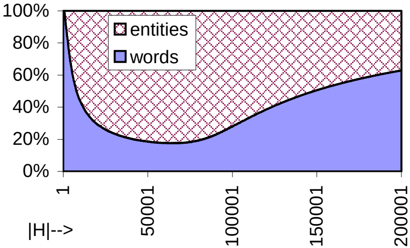
Summary of contributions

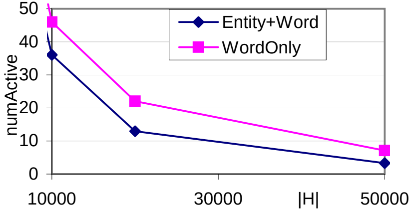
- Additional index space typically .1–1$\times$ basic text index
- Precomputation much faster (typically 52$\times$) than computing all word PPVs
- Query time much faster than query-time whole-graph PageRank (typically 35–450$\times$, gain grows with graph size)
- High ranking accuracy (precision $\approx$ .91)

Heuristic estimate of hub inclusion merit

initialize map $meritScore(u)$ for nodes $u \in W_0 \cup N$
for each query word $w \in W_0$ **do**
    attach node w to the preloaded entity graph
    let $frontier = \{w\}$ and $priority(w) = \widetilde{\Pr}(w)$
    create an empty set of visited nodes
    while $frontier \neq \varnothing$ **do**
        remove some u from $frontier$ and mark visited
        $meritScore(u)$ += $priority(u)$
        for each visited neighbor v **do**
            $meritScore(v)$ += $\alpha\, priority(u)\, C(u \rightarrow u)$
        for each unvisited neighbor v **do**
            let $priority(v) = \alpha\, priority(u)\, C(u \rightarrow v)$
            add v to $frontier$
sort word and entity nodes by decreasing $meritScore(u)$

Greedy merit order: Preliminary evaluation





- We pick a nontrivial mix of words and a large number of entities
- Unlike naive application of "large PageRank first" which picks words almost exclusively
- Allowing well-chosen entities into H significantly reduces active subgraph size

Replacing PPVs with fingerprints

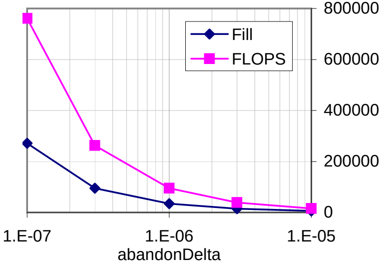
- Computing full-precision PPV_u is overkill if
  - All we care about is a top-k entity ranking
  - u is far from teleport origins (almost a loser)
- Idea (Fogaras *et al.*): Compute FP_u as follows:

1: sample walk length λ from $\Pr(\lambda) = \alpha^\lambda(1-\alpha)$
2: **repeat**
3: start at u, use C to take λ "random surfer" steps, ending in v
4: **until** `numWalks` walks completed
5: compute normalized histogram of end-node counts
6: store $\langle \widehat{\text{PPV}}_u(v), v\rangle$ records in decreasing $\hat{\text{PPV}}_u(v)$ order

- How to set `numWalks` for each $u \in H$?
- If $meritScore(u)$ is large, allocate it more `numWalks`

Loading FPs with dynamic clipping

- `numWalks` is based on aggregate query stats
- For a specific query, some $\widehat{\mathrm{PPV}}_u$ may be "too precise"
- When marking active subgraph, suppose activation score of u is s
- Recall $\widehat{\mathrm{PPV}}_u$ is stored as $\langle \widehat{\mathrm{PPV}}_u(v), v \rangle$ records in decreasing $\widehat{\mathrm{PPV}}_u(v)$ order
- While loading $\widehat{\mathrm{PPV}}_u$, if $s\,\widehat{\mathrm{PPV}}_u(v) < \delta_{\mathrm{abandon}}$, quit



- Use sparse vectors for Jeh-Widom updates
- Fill is the number of nonzero PPV elements over the active subgraph upon convergence
- Dramatic reduction in fill and flops

Accuracy indicators

For a fixed query, let S_k be the true top-k sequence and $\hat{S}_k$ be the sequence returned by the system

Precision at k: $\dfrac{|S_k \cap \hat{S}_k|}{k}$

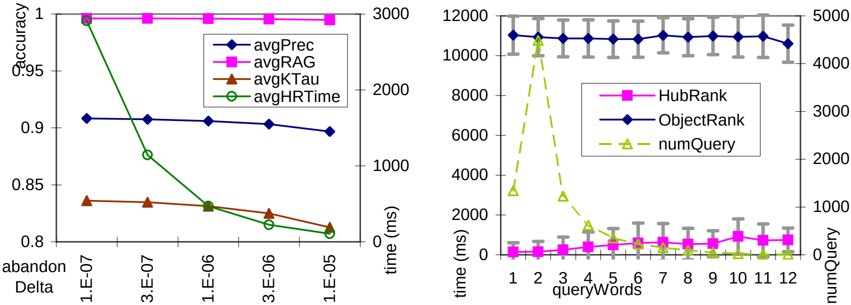
Relative aggregate goodness (RAG) at k: Let true score of $v \in S_k \cup \hat{S}_k$ be $S_k(v)$, then RAG is

$$\frac{\sum_{v \in \hat{S}_k} S_k(v)}{\sum_{v \in S_k} S_k(v)}$$

Kendall's τ: Let system score of v be $\hat{S}_k(v)$. Pair $v, w \in S_k \cup \hat{S}_k$ is *concordant* if $(S_k(v) - S_k(w))(\hat{S}_k(v) - \hat{S}_k(w)) > 0$, *discordant* if < 0, *exactTie* if $S_k(v) = S_k(w)$, *approxTie* if $\hat{S}_k(v) = \hat{S}_k(w)$.

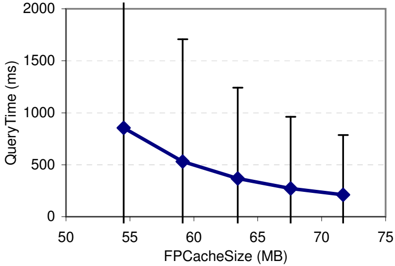
$$\tau_k = \frac{\#concordant - \#discordant}{\sqrt{(\#pairs - \#exactTie)(\#pairs - \#approxTie)}}$$

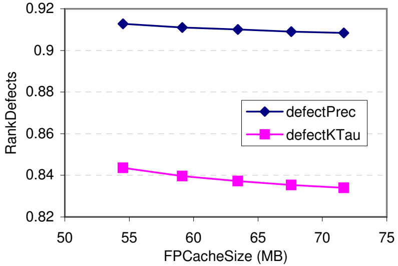
Speed and accuracy of HubRank



- As δ_{abandon} is increased to $3 \times 10^{-6} \ldots 10^{-5}$, dramatic reduction in HubRank query processing time
- While accuracy is very mildly degraded
- With $|V| = 74223$, HubRank is $80\times$, $74\times$, $43\times$ faster for 1-, 2- and 3-word queries
- Gap even more striking for a 320000-node test graph

Effect of FP cache size

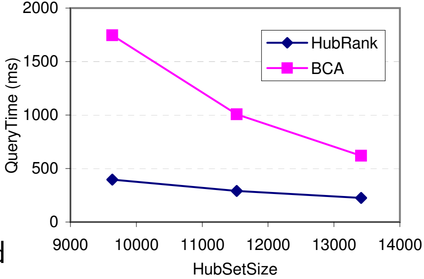




- Earlier we had measured active subgraph size as an indirect indicator of query time
- Here we plot query time and accuracy against physical cache size on disk
- For reference, a Lucene index is 56 MB
- As cache size grows from 50–75 MB,
  - Query time decreases by almost $4\times$
  - Accuracy degrades by less than 1%

Comparison with Berkhin's BCA

- "Bookmark coloring algorithm"
- Elegant approach to identify active subgraph and compute conductance at the same time
- Can exploit hubs like we do
- Does not discuss workload-driven hub-selection
- Does not discuss fast PPV approximations via FPs
- 3–4 times slower in our testbed at around same level of accuracy and same physical cache size



Summary

- Practical interactive graph conductance search system
- Graceful tradeoff between index space and query time
- Index space comparable to basic text index
- Fast query execution with high ranking accuracy
- Preprocessing time tiny compared to full-vocab ObjectRank
- Code+data available, call soumen@cse.iitb.ac.in

Ongoing work

- Guaranteed top-k by enhancing BCA
- New hubset choosing algo for top-k BCA
- Hybrid index of PPVs and FPs
- Improved accuracy with reduced index space