2 Efficient Personalized Authority Ranking

2.1 Memory-efficient Incremental Page Importance

Goals:
- Compute Page-Rank-style authority measure online without having to store the complete link graph
- Recompute authority incrementally as the graph changes

Key idea:
- Each page holds some „cash" that reflects its importance
- When a page is visited, it distributes its cash among its successors
- When a page is not visited, it can still accumulate cash
- This random process has a stationary limit that captures importance of pages

OPIC Algorithm (Online Page Importance Computation)

Maintain for each page i (out of n pages):
 C[i] – cash that page i currently has and distributes
 H[i] – history of how much cash page has ever had in total
plus global counter
 G – total amount of cash that has ever been distributed

```
for each i do { C[i] := 1/n; H[i] := 0 }; G := 0;
do forever {
   choose page i (e.g., randomly);
   H[i] := H[i] + C[i];
   for each successor j of i do C[j] := C[j] + C[i] / outdegree(i);
   G := G + C[i];
   C[i] := 0;
};
```

Note: 1) every page needs to be visited infinitely often (fairness)
 2) the link graph L' is assumed to be strongly connected

OPIC Importance Measure

At each step t an estimate of the importance of page i is:
 $(H_t[i] + C_t[i]) / (G_t + 1)$ (or alternatively: $H_t[i] / G_t$)

Theorem:
Let $X_t = H_t / G_t$ denote the vector of cash fractions accumulated by pages until step t.
The limit $X = \lim_{t \to \infty} X_t$ exists with $|X|_1 = \Sigma_i X[i] = 1$.

with crawl strategies such as:
- random
- greedy: read page i with highest cash C[i]
 (fair because non-visited pages accumulate cash until eventually read)
- cyclic (round-robin)

Adaptive OPIC for Evolving Link Graph

Consider „time" window [now-T, now] where time is the value of G

Estimated importance of page i is: $X_{now}[i] = (H_{now}[i] – H_{now-T}[i]) / T$

For fixed window size T maintain for every page i:
$C_t[i]$ and G_t for each time t that i was visited within the window

For variable window size k maintain for every page i:
$C_t[i]$ and G_t for each time t of the last k visits of i

For two-point estimate (e.g., previous and current crawl):
- maintain two history vectors H[1..n] and G[1..n] for accumulated cash H[i] in current crawl and time G[i] of previous crawl
- set

$$H[i] := \begin{cases} H[i] \cdot \frac{T-(G-G[i])}{T} + C[i] & \text{if } G-G[i] < T \\ C[i] \cdot \frac{T}{G-G[i]} & \text{if } G-G[i] \geq T \end{cases}$$

Implementation and Experiments

essentially a crawler distributed across 4 PCs
with hash-based partitioning of URLs
uses Greedy-style crawl strategy and two-point interpolation

can track „accumulated cash" of href targets without visiting them !

Web crawl visited 400 Mio. pages and
computed importance of 1 Bio. pages
over several months.

2.2 Personalized Page-Rank for Many Users

Efficiently compute solution of personalized PR vector $x_{(n\times 1)}$ with user preference vector u with $|u|_1=1$ and $u_i\neq 0$ only for $i\in H$, with $|H|<<n$, and A_{ij} = 1/outdegree(i) for edge $i\rightarrow j$,0 else: $x = (1-\varepsilon) A x + \varepsilon u$

Key ideas:
1) consider only basis vectors u with $u_p=1$ and $u_i=0$ for $i\neq p$ for hub p and represent full user preference as linear combination
2) represent p-specific Page-Rank vectors in the form of a hub skeleton and a set of partial vectors
3) factor out common parts of different random walks

Notation:
hub set H, preference set $P \subseteq H \subset V = \{1, 2, ..., n\}$
basis vector e_p with single non-zero entry at p
p-specific PR vector r_p
partial vector $r_p - r_p^H$

Basis Vectors for Hub Set H

Linearity Theorem:
For any preference vectors u1 and u2, if v1 and v2 are the corresponding PR vectors, then for any constants α1 and α2 ≥ 0 with α1 + α2 = 1 the following holds:
$\alpha 1\, v1 + \alpha 2\, v2 = (1-\varepsilon)\, A\, (\alpha 1\, v1 + \alpha 2\, v2) + \varepsilon\, (\alpha 1\, u1 + \alpha 2\, u2)$

Corollary:
For an arbitrary user prefence vector u and basis vectors e_p the following holds:

$u = \sum_{p=1}^{m} \alpha_p \cdot e_p$ for some constants α_1 through α_m (m = |H|)

and $v = \sum_{p=1}^{m} \alpha_p \cdot r_p$ for the corresponding PR vector v

Hubs Skeleton and Partial Vectors (1)

Definition:
For pages p, q the *inverse P-distance* $r'_p(q)$ from p to q is:

$$r'_p(q) = \sum_{\substack{tours\ t:\\ p\rightarrow q}} P[t]\,\varepsilon\,(1-\varepsilon)^{length(t)}$$ *(prob. of surfing from p to q before the next random jump)*

where $P[t: w_1 w_2 ... w_k \text{ of length } k-1] = \prod_{i=1}^{k-1} 1/\text{outdegree}(w_i)$

Theorem:
$r'_p(q) = r_p(q)$ for all pages p, q (with the p-specific PR vector $r_p(q)$)

Definition:
For pages p, q and hub set H the *hub-restricted inverse P-distance* $r_p^H(q)$ is:

$$r_p^H(q) = \sum_{\substack{tours\ t: p\rightarrow h\rightarrow q\\ with\ h\in H}} P[t]\,\varepsilon\,(1-\varepsilon)^{length(t)}$$

Hubs Skeleton and Partial Vectors (2)

Partial vectors:
For many q: $r_p(q) - r^H_p(q) = 0$,
and the number of such q increases with |H|.
→ store only sparse vectors $r_p(q) - r^H_p(q)$
Note: partial vectors become smaller when pages in H have high PR

Hubs skeleton: Compute r^H_p vector from partial vectors and a „skeleton"

Hubs theorem: For any page p and hub set H:

$$r_p^H = \frac{1}{\varepsilon} \sum_{h\in H} (r_p(h) - \varepsilon\, e_p(h)) \left(r_h - r_h^H - \varepsilon\, e_h\right)$$ and thus

$$r_p = \left(r_p - r_p^H\right) + \frac{1}{\varepsilon} \sum_{h\in H} (r_p(h) - \varepsilon\, e_p(h)) \left(\left(r_h - r_h^H\right) - \varepsilon\, e_h\right)$$

In addition to the partial vectors, we thus need to compute and store the skeleton $S = \{ r_p(h) \mid h \in H\}$
Note: $r_p(h)$ for all $h \in H$ is much smaller than $r_p(q)$ for all pages q

Hubs Skeleton and Web Skeleton

Intuition behind Hub Theorem:
Distance from p to q through H is distance $r_p(h)$ from p to each $h\in H$ times the distance from $r_h(q)$ from h to q.

The hubs skeleton captures distances from hub to hub; partial vectors capture distances from hub to arbitrary node (without traveling through another hub).

Web skeleton:
In the hubs skeleton $r_p(h)$ is computed only for $p\in H$.
This can be generalized to compute $r_p(h)$ for all $p\in V$ and $h\in H$.
With this kind of Web skeleton, $r_p^H(q)$ would yield an approximation of personalized Page-Rank distances for arbitrary nodes p, q $\in V$.

Factorizing Random-Walk Computations

We need to precompute partial vectors $r_p(q) - r^H_p(q)$ and the hubs skeleton $S = \{ r_p(h) \mid h \in H\}$.
Invoking a power iteration for each p separately would be very slow.

Decomposition theorem:
For any page p: $r_p = \frac{1-\varepsilon}{\text{outdegree}(p)} \sum_{i\in out(p)} r_i + \varepsilon\, e_p$

Algorithmic framework:
- run power iteration k=1, 2, ... until convergence (for computing all full vectors r_p or partial vectors)
- compute lower-approximation $D_k[p]$ for r_p: $D_k[p](q) \le r_p(q)$ for all q and error measure $E_k[p]$.
- maintain invariance: $D_k[p] + \sum_{q\in V} E_k[p](q)\cdot r_q = r_p$ for all k
- start with $D_0[p] = \vec{0}$ and $E_0[p] = e_p$

Basic Dynamic Programming Algorithm

In round k+1:
compute approximation of r_p from the round-k approximations for the successors of p
→ substitute $D_k[p]+E_k[p]$ invariance equation into decomposition theorem:

$$D_{k+1}[p] = \frac{1-\varepsilon}{\text{outdegree}(p)} \sum_{i \in out(p)} D_k[i] + \varepsilon e_p$$

$$E_{k+1}[p] = \frac{1-\varepsilon}{\text{outdegree}(p)} \sum_{i \in out(p)} E_k[i]$$

reduces the error by factor 1-ε in each round

Selective Expansion Algorithm

In round k+1:
choose set $Q_k(p) \subseteq V$ and for each page $q \in Q_k(p)$
„distribute the error" to its successors

$$D_{k+1}[p] = D_k[p] + \sum_{q \in Q_k(p)} \varepsilon E_k[p](q) \cdot e_q$$

$$E_{k+1}[p] = E_k[p] - \sum_{q \in Q_k(p)} E_k[p](q) \cdot e_q + \sum_{q \in Q_k(p)} \frac{1-\varepsilon}{out\deg ree(q)} \sum_{i \in out(q)} E_k[p](q) \cdot e_i$$

The sets $Q_k(p)$ are tunable
(e.g., choose m pages q with highest $E_k[p](q)$)

Repeated Squaring Algorithm

Compute iteration 2k results from iteration k results
(based on Selective Expansion equations):

$$D_{2k}[p] = D_k[p] + \sum_{q \in Q_k(p)} E_k[p](q) \cdot D_k[q]$$

$$E_{2k}[p] = E_k[p] - \sum_{q \in Q_k(p)} E_k[p](q) \cdot e_q + \sum_{i \in out(q)} E_k[p](q) \cdot E_k[q]$$

The sets $Q_k(p)$ are again tunable.

Computing Partial Vectors instead of Full Vectors

Partial vectors:
specialized selected expansion algorithm
by choosing $Q_0(p) = V$ and $Q_k(p) = V–H$ for k≥1

→ $D_k[p] + \varepsilon E_k[p]$ converges to $r_p – r_p^H$

Hubs skeleton:
specialized repeated squaring algorithm
use results $D_k[p]$, $E_k[p]$ from partial-vector computation
apply repeated squaring step using $Q_k(p) = H$

Implementation and Experiments

Keep only two successive values for $D_k[*]$ and $E_k[*]$
Partition disk-resident data structure into blocks P1, ..., Pm (e.g., m=10)
with Pi: Vi – nodes that reside in this block,
Ei – adjacency lists with edges (p,q) for p ∈Vi,
Lk_{ij} – intermediate results for $D_k[p](q)$ and $E_k[p](q)$ with p ∈Vi and q ∈Vj
In each iteration k compute results by partition (read into memory)

On Stanford WebBase with 120 Mio. pages,
using a 1.4 GHz CPU with 3.5 GB memory, with |H|=10000
the time for computing (for all basis vectors p)
the full PR vectors r_p was about 10000 * 3 seconds ≈ 8 hours,
for the partial vectors $r_p–r_p^H$ it was 10000 * 0.3 seconds ≈ 50 minutes,
for the hubs skeleton $r_p(H)$ it was about 10 hours.
A full PR vector (with > 14 Mio. non-zero entries) can be constructed from partial vectors and hubs skeleton in 6 seconds.

Literature

- Serge Abiteboul, Mihai Preda, Gregory Cobena: Adaptive on-line page importance computation, WWW Conference, 2003.
- Glen Jeh, Jennifer Widom: Scaling personalized web search, WWW Conference, 2003. (see also Technical Report 2002-12, CS Dept., Stanford University)