


**<isweb>** ISWeb – Information Systems & Semantic Web  
University of Koblenz • Landau

## Chapter 6

### Efficiency and Scalability

Sergej Sizov  
Web Mining  
Summer term 2007



### 6.1 Top-k Query Processing with Scoring

Vector space model suggests  $m \times n$  term-document matrix, but data is sparse and queries are even very sparse  
→ better use *inverted index lists* with terms as keys for B+ tree

q: professor  
research  
xml

B+ tree on terms

professor	...	research	...	xml
17: 0.3		12: 0.5		11: 0.6
44: 0.4		14: 0.4		17: 0.1
52: 0.1		28: 0.1		28: 0.7
53: 0.8		44: 0.2		⋮
55: 0.6		51: 0.6		52: 0.3
⋮		⋮		⋮

index lists with (DocId, s = tf\*idf) sorted by DocId

Google: > 10 mio. terms  
> 8 bio. docs  
> 4 TB index

terms can be full words, word stems, word pairs, word substrings, etc.  
(whatever „dictionary terms“ we prefer for the application)

queries can be conjunctive or „andish“ (soft conjunction)

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 6 – Efficiency and Scalability 6.2

### DBS-Style Top-k Query Processing

q: professor  
research  
xml

B+ tree on terms

professor	...	research	...	xml
17: 0.3		12: 0.5		11: 0.6
44: 0.4		14: 0.4		17: 0.1
52: 0.1		28: 0.1		28: 0.7
53: 0.8		44: 0.2		⋮
55: 0.6		51: 0.6		52: 0.3
⋮		⋮		⋮

index lists with (DocId, s = tf\*idf) sorted by DocId

Google: > 10 mio. terms  
> 8 bio. docs  
> 4 TB index

**Given:** query  $q = t_1 t_2 \dots t_z$  with  $z$  (conjunctive) keywords  
similarity scoring function  $score(q,d)$  for docs  $d \in D$ , e.g.:  $\vec{q} \cdot \vec{d}$   
with precomputed scores (index weights)  $s_i(d)$  for which  $q_i \neq 0$

**Find:** top k results w.r.t.  $score(q,d) = \text{aggr}\{s_i(d)\}$  (e.g.:  $\sum_{i \in q} s_i(d)$ )

**Naive join&sort QP algorithm:**

top-k (

$\sigma[\text{term}=t_1]$ (index)	×	DocId
$\sigma[\text{term}=t_2]$ (index)	×	DocId
...	×	DocId
$\sigma[\text{term}=t_z]$ (index)		

order by s desc)

### Computational Model for Top-k Queries

Assume *local scores*  $s_i$  for query  $q$ , data item  $d$ , and dimension  $i$ , and *global scores*  $s$  of the form  $s(q,d) = \text{aggr}\{s_i(q,d) \mid i = 1..m\}$  with a *monotonic* aggregation function  $\text{aggr} : [0,1]^m \rightarrow [0,1]$

Examples:  $s(q,d) = \sum_{i=1}^m s_i(q,d)$      $s(q,d) = \max\{s_i(q,d) \mid i = 1..m\}$

**Find top-k data items with regard to global scores:**

- process  $m$  *index lists*  $L_i$  with *sorted access (SA)* to entries  $(d, s_i(q,d))$  in *ascending order of doc ids* or *descending order of  $s_i(q,d)$*
- maintain for each candidate  $d$  a set  $E(d)$  of evaluated dimensions and a *partial score „accumulator“*
- for candidate  $d$  with incomplete  $E(d)$  consider looking up  $d$  in  $L_i$  for all  $i \in R(d)$  by *random access (RA)*
- terminate index list scans when enough candidates have been seen
- if necessary sort final candidate list by global score

### Index List Processing by Merge Join

Keep  $L(i)$  in *ascending order of doc ids*  
Compress  $L(i)$  by actually storing the gaps between successive doc ids (or using some more sophisticated prefix-free code)

QP may start with those  $L(i)$  lists that are *short and have high idf*  
Candidate results need to be looked up in other lists  $L(j)$   
To avoid having to uncompress the entire list  $L(j)$ ,  
 $L(j)$  is encoded into groups of entries with a *skip pointer* at the start of each group  
→  $\sqrt{\text{length}}$  evenly spaced skip pointers for list of length  $n$

$L_i$	2	4	9	16	59	66	128	135	291	311	315	591	672	899	...
$L_j$	1	2	3	5	8	17	21	35	39	46	52	66	75	88	...

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 6 – Efficiency and Scalability 6.5

### Efficient Top-k Search

[Buckley85, Güntzer/Balke/Kießling 00, Fagin01]

*threshold algorithms: efficient & principled top-k query processing with monotonic score aggr.*

**Data items:**  $d_1, \dots, d_n$

$s(t_1, d_1) = 0.7$   
 $s(t_m, d_1) = 0.2$

Query:  $q = (t_1, t_2, t_3)$

**TA with sorted access only (NRA):**  
can index lists; consider  $d$  at pos <sub>$i$</sub>  in  $L_i$ ;  
 $E(d) := E(d) \cup \{i\}$ ;  $\text{high}_i := s(t_i, d)$ ;  
 $\text{worstscore}(d) := \text{aggr}\{s(t_i, d) \mid i \in E(d)\}$ ;  
 $\text{bestscore}(d) := \text{aggr}\{\text{worstscore}(d), \text{aggr}\{\text{high}_i \mid i \notin E(d)\}\}$ ;  
if  $\text{worstscore}(d) > \text{min-k}$  then add  $d$  to top-k  
 $\text{min-k} := \min\{\text{worstscore}(d') \mid d' \in \text{top-k}\}$ ;  
else if  $\text{bestscore}(d) > \text{min-k}$  then  
   $\text{cand} := \text{cand} \cup \{d\}$ ;  
   $\text{threshold} := \max\{\text{bestscore}(d') \mid d' \in \text{cand}\}$ ;  
  if  $\text{threshold} \leq \text{min-k}$  then exit;

Rank	Doc	Worst-score	Best-score
1	d10	2.1	2.1
2	d78	1.4	2.0
3			1.8
4			2.0

keep  $L(i)$  in *descending order of scores*

Course Web Mining 4.1.23 Summer Term 2007 Sergej Sizov Chapter 6 – Efficiency and Scalability 6.6

**Threshold Algorithm (TA, Quick-Combine, MinPro)**  
(Fagin'01; Guntzer/Balke/Kießling; Nepal/Ramakrishna)

scan all lists  $L_i$  ( $i=1..m$ ) in parallel:  
 consider  $d_j$  at position  $pos_j$  in  $L_i$ ;  
 $high_i := s_i(d_j)$ ;  
 if  $d_j \notin top-k$  then {  
   look up  $s_v(d_j)$  in all lists  $L_v$  with  $v \neq i$ ; // random access  
   compute  $s(d_j) := \text{aggr} \{s_v(d_j) \mid v=1..m\}$ ;  
   if  $s(d_j) > \text{min score among top-k then}$   
     add  $d_j$  to top-k and remove min-score  $d$  from top-k; }  
 $threshold := \text{aggr} \{high_i \mid i=1..m\}$ ;  
 if min score among top-k  $\geq$  threshold then exit;

*but random accesses are expensive !*

$m=3$   
aggr: sum  
 $k=2$

f: 0.5
b: 0.4
c: 0.35
a: 0.3
h: 0.1
d: 0.1

a: 0.55
b: 0.2
f: 0.2
g: 0.2
e: 0.1
i: 0.1

h: 0.35
d: 0.35
b: 0.2
a: 0.1
e: 0.05
f: 0.05

**top-k:**  
f: 0.75  
a: 0.95  
b: 0.8

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.7

**No-Random-Access Algorithm**  
(NRA, Stream-Combine, TA-Sorted)

scan index lists in parallel:  
 consider  $d_j$  at position  $pos_j$  in  $L_i$ ;  
 $E(d_j) := E(d_j) \cup \{i\}$ ;  $high_i := si(q, d_j)$ ;  
 $bestscore(d_j) := \text{aggr}\{x_1, \dots, x_m\}$   
   with  $x_i := si(q, d_j)$  for  $i \in E(d_j)$ ,  $high_i$  for  $i \notin E(d_j)$ ;  
 $worstscore(d_j) := \text{aggr}\{x_1, \dots, x_m\}$   
   with  $x_i := si(q, d_j)$  for  $i \in E(d_j)$ , 0 for  $i \notin E(d_j)$ ;  
 $top-k := k$  docs with largest worstscore;  
 $threshold := bestscore\{d \mid d \text{ not in top-k}\}$ ;  
 if min worstscore among top-k  $\geq$  threshold then exit;

$m=3$   
aggr: sum  
 $k=2$

f: 0.5
b: 0.4
c: 0.35
a: 0.3
h: 0.1
d: 0.1

a: 0.55
b: 0.2
f: 0.2
g: 0.2
e: 0.1
i: 0.1

h: 0.35
d: 0.35
b: 0.2
a: 0.1
e: 0.05
f: 0.05

**top-k:**  
a: 0.95  
b: 0.8

**candidates:**  
f: 0.7 + ?  $\leq$  0.7 + 0.1  
h: 0.45 + ?  $\leq$  0.45 + 0.2  
c: 0.35 + ?  $\leq$  0.35 + 0.3  
d: 0.35 + ?  $\leq$  0.35 + 0.3  
g: 0.2 + ?  $\leq$  0.2 + 0.4

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.8

**Optimality of TA**

Definition:  
 For a class  $\mathcal{A}$  of algorithms and a class  $\mathcal{D}$  of datasets,  
 let  $\text{cost}(A, D)$  be the execution cost of  $A \in \mathcal{A}$  on  $D \in \mathcal{D}$ .  
 Algorithm B is *instance optimal* over  $\mathcal{A}$  and  $\mathcal{D}$  if  
 for every  $A \in \mathcal{A}$  on  $D \in \mathcal{D}$ :  $\text{cost}(B, D) = O(\text{cost}(A, D))$ ,  
 that is:  $\text{cost}(B, D) \leq c \cdot \text{cost}(A, D) + c'$   
 with optimality ratio (competitiveness)  $c$ .

Theorem:

- TA is instance optimal over all algorithms that are based on sorted and random access to (index) lists (no „wild guesses“).  
 TA has optimality ratio  $m + m(m-1) C_{RA}/C_{SA}$   
 with random-access cost  $C_{RA}$  and sorted-access cost  $C_{SA}$ .
- NRA is instance-optimal over all algorithms with SA only.

*if „wild guesses“ are allowed,  
 then no deterministic algorithm is instance-optimal*

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.9

**Execution Cost of TA Family**

**Run-time cost** is  $O\left(n^{\frac{m-1}{m}} \cdot k^{\frac{1}{m}}\right)$  with arbitrarily high probability  
 (for independently distributed  $L_i$  lists)

**Memory cost** is  $O(k)$  for TA  
 and  $O(n^{(m-1)/m})$  for NRA (priority queue of candidates)

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.10

**6.2 Approximate Top-k Query Processing**

Approximation TA:  
 A  $\theta$ -approximation  $T'$  for top-k query  $q$  with  $\theta > 1$   
 is a set  $T'$  of docs with:

- $|T'|=k$  and
- for each  $d' \in T'$  and each  $d'' \notin T'$ :  $\theta \cdot \text{score}(q, d') \geq \text{score}(q, d'')$

Modified TA:  
 ...  
 Stop when  $\min_k \geq \text{aggr}(high_1, \dots, high_m) / \theta$

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.11

**Pruning and Access Ordering Heuristics**

General heuristics:

- disregard index lists with idf below threshold
- for index scans give priority to index lists that are short and have high idf

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6.12

**Pruning with Combined Authority/Similarity Scoring**  
(Long/Suel 2003)

Focus on  $\text{score}(q,d_j) = r(d_j) + s(q,d_j)$   
 with normalization  $r(\cdot) \leq a, s(\cdot) \leq b$  (and often  $a+b=1$ )  
 Keep index lists sorted in **descending order of „static“ authority  $r(d_j)$**

**Conservative authority-based pruning:**  
 $\text{high}(0) := \max\{r(\text{pos}(i)) \mid i=1..m\}$ ;  $\text{high} := \text{high}(0) + b$ ;  
 $\text{high}(i) := r(\text{pos}(i)) + b$ ;  
 stop scanning  $i$ -th index list when  $\text{high}(i) < \text{min score of top } k$   
 terminate algorithm when  $\text{high} < \text{min score of top } k$   
 effective when total score of top- $k$  results is dominated by  $r$

**First- $k'$  heuristics:**  
 scan all  $m$  index lists until  $k' \geq k$  docs have been found  
 that appear in all lists;  
 the stopping condition is easy to check because of the sorting by  $r$

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-13

**Separating Documents with Large  $s_i$  Values**

**Idea (Google):**  
 in addition to the full index lists  $L(i)$  sorted by  $r$ ,  
 keep short „fancy lists“  $F(i)$  that contain the docs  $d_j$   
 with the highest values of  $s_i(\text{ti},d_j)$  and sort these by  $r$

**Fancy first- $k'$  heuristics:**  
 Compute total score for all docs in  $\cap F(i)$  ( $i=1..m$ )  
 and keep top- $k$  results;  
 $\text{Cand} := \cup_i F(i) - \cap_i F(i)$ ;  
 for each  $d_j \in \text{Cand}$  do {compute partial score of  $d_j$ };  
 Scan full index lists  $L(i)$  ( $i=1..k$ );  
 if  $\text{pos}(i) \in \text{Cand}$   
 {add  $s_i(\text{ti},\text{pos}(i))$  to partial score of  $\text{pos}(i)$ };  
 else {add  $\text{pos}(i)$  to Cand and set its partial score to  $s_i(\text{ti},\text{pos}(i))$ };  
 Terminate the scan when  $k'$  docs  
 have a completely computed total score;

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-14

**Authority-based Pruning with Fancy Lists**

Guarantee that the top  $k$  results are complete by  
 extending the fancy first- $k'$  heuristics as follows:  
 stop scanning the  $i$ -th index list  $L(i)$  not after  $k'$  results,  
 but only when we know that no incompletely scored doc  
 can qualify itself for the top  $k$  results

Maintain:  
 $r\_high(i) := r(\text{pos}(i))$   
 $s\_high(i) := \max\{s_i(q,d_j) \mid d_j \in L(i) - F(i)\}$   
 Scan index lists  $L(i)$  and accumulate partial scores for all docs  $d_j$   
 Stop scanning  $L(i)$  iff  
 $r\_high(i) + \sum_i s\_high(i) < \min\{\text{score}(d) \mid d \in \text{current top-}k \text{ results}\}$

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-15

**6.3 Personalized Page-Rank for Many Users**

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

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

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-16

**PPV: Our strategy**

```

graph TD
  PPV[PPV] --> LT{Linear Theory}
  LT --> BV[basis vectors]
  BV --> DT{Decomposition Theorem}
  DT --> RBV[reduced basis vectors]
  RBV --> HT{Hubs Theorem}
  HT --> PV[partial vectors]
  HT --> HS[hubs skeleton]
  
```

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-17

**Basis Vectors for Hub Set H**

```

graph LR
  PPV[PPV] --> LT{Linear Theory}
  LT --> BV[basis vectors]
  
```

**Linearity Theorem:**  
 For any preference vectors  $u_1$  and  $u_2$ , if  $v_1$  and  $v_2$  are the  
 corresponding PR vectors, then for any constants  $\alpha_1$  and  $\alpha_2 \geq 0$   
 with  $\alpha_1 + \alpha_2 = 1$  the following holds:  
 $\alpha_1 v_1 + \alpha_2 v_2 = (1 - \epsilon) A (\alpha_1 v_1 + \alpha_2 v_2) + \epsilon (\alpha_1 u_1 + \alpha_2 u_2)$

**Corollary:**  
 For an arbitrary user preference vector  $u$  and basis vectors  $e_p$   
 the following holds:

$$u = \sum_{p=1}^m \alpha_p \cdot e_p \quad \text{for some constants } \alpha_1 \text{ through } \alpha_m \text{ (} m = |H| \text{)}$$

and

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

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-18

### Hubs Skeleton and Partial Vectors (1)

**Definition:**  
For pages p, q the *inverse P-distance*  $r_p^i(q)$  from p to q is:

$$r_p^i(q) = \sum_{\text{tours } t: p \rightarrow q} P[t] \epsilon (1 - \epsilon)^{\text{length}(t)} \quad (\text{prob. of surfing from } p \text{ to } q \text{ before the next random jump})$$

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

**Theorem:**  
 $r_p^i(q) = r_p(q)$  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:

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

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-19

### Hubs Skeleton and Partial Vectors (2)

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

**Hubs skeleton:** Compute  $r_p^H$  vector from partial vectors and a „skeleton“

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

$$r_p^H = \frac{1}{\epsilon} \sum_{h \in H} (r_p(h) - \epsilon e_p(h)) (r_h - r_h^H - \epsilon e_h)$$

and thus

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

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

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-20

### 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  $\in V$ .

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-21

### 6.4 PageRank in a Distributed Search System

Problems with collecting web data for large-scale Web experiments:

- Scalability
- Slow Update
- Hidden Web sources
- Robot exclusion rules
- High maintenance costs

General idea - avoid computing complete global PageRank:

- **Local PageRank** vectors are computed on each web server individually (i.e. in a distributed fashion)
- The relative importance of different web servers is measured by computing the **ServerRank** vector
- The **Local PageRank** vectors are then refined using the **ServerRank** vector. Query results on a web server are rated by its **Local PageRank** vector

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-22

### Local Page Rank Computation

**LPR-1**

Remove all inter-server links (ones in pink)  
Apply PageRank Algorithm

**LPR-2**

Servers exchange counts of inter-server link information

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-23

### Distributed PR: Result Fusion

When a search query is submitted to a web server:

1. It is forwarded to relevant Web servers
2. Each receiving server executes the query using Local PageRank vector and returns a ranked list of matches
3. The coordinator performs result fusion and merges result lists

Preliminary results:  
method outcome is a reasonable approximate of a centralized one-server solution

ServerRank

→

$Y_{s1}$ 
 $Y_{q1}$ 
 $\vdots$ 
 $Y_{sn}$ 
 $Y_{qn}$

←

aggregated PR values

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-24

### 6.5 JXP: PageRank computation in P2P networks

#### Motivation

- Link-based ranking algorithms widely used in Web search engines;
- Centralized approaches are not suitable in peer-to-peer (P2P) networks.

#### JXP Contribution

- Decentralized algorithm for computing authority scores of pages in a P2P Network;
- Local PageRank computations + Meetings between peers;
- Approximates global PageRank scores;

### Related Work

Haveliwala et al. [1]

- BlockRank algorithm that exploits the block structure of the Web;

Wang & DeWitt [2]

- Local PageRank and ServerRank combined to approximate true global PageRank;

Both approaches rely on a predefined partition of the Web, which is not suitable in a P2P network.

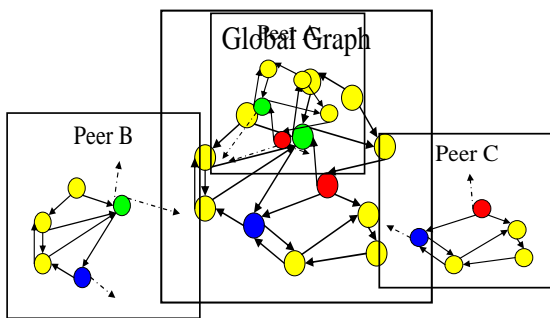
[1] S. Kamvar, T. Haveliwala, C. Manning & G. Golub. *Exploiting the block structure of the web for computing pagerank*. Technical report, Stanford University, 2003.

[2] Y. Wang & D. J. DeWitt. *Computing pagerank in a distributed internet search system*. In VLDB, 2004.

### Computational Model

Every peer crawls Web fragments at its discretion

- Overlaps between peers' graphs may occur;



### Digression: PageRank revisited

Importance of a page depends on the importance of the pages that point to it;

- Computed using power iteration method;

$$PR(q) = \alpha \times \frac{1}{N} + (1 - \alpha) \times \sum_{p|p \rightarrow q} \frac{PR(p)}{\text{out}(p)}$$

- N → Total number of pages;
- PR(p) → PageRank of page p;
- Out(p) → Outdegree of p
- α → Random jump probability

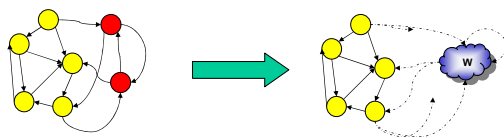
### The JXP Algorithm (I)

“World Node”:

- Special node attached to the local graph at every peer;
- Compact representation of all other pages in the network.

Initialization step:

- Local graph is extended by adding the world node;
- PageRank is computed in the extended graph.



\* JXP: Juxtaposed approxImate PageRank

### The JXP Algorithm (II)

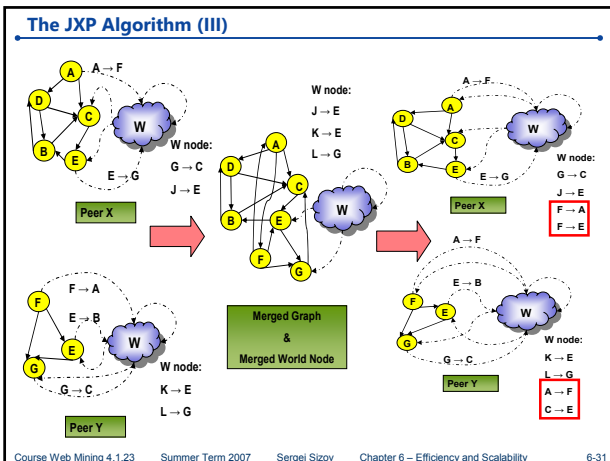
Main algorithm (for every peer<sub>i</sub> in the network)

- Choose peer<sub>i</sub> at random;
- Combine local graphs
  - World nodes are combined as well;
- Compute PageRank in the merged graph → PR vector;
- Update JXP scores:

$$JXP(i) = \begin{cases} PR(i), & \text{if page } i \text{ is in the merged graph} \\ \frac{JXP(i) * PR(W)}{JXP(W)}, & \text{otherwise} \end{cases}$$

Disconnect local Graphs

- World nodes are reconstructed, based on what was learned in the meeting.



### The JXP Algorithm (IV)

Scalable

- PageRank computation performed in relative small graphs;

Efficient

- Computational cost

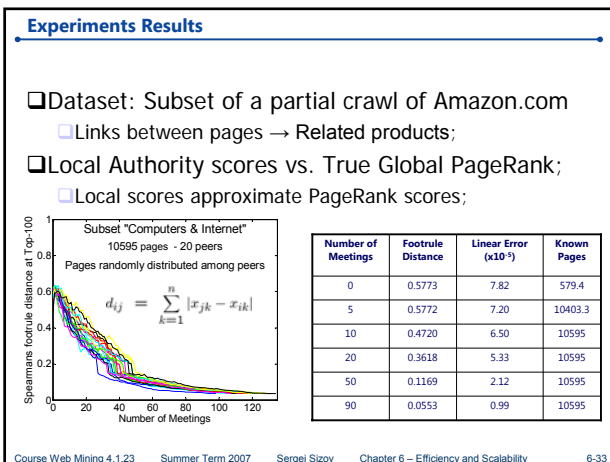
$C(\text{merge graphs}) + C(\text{PageRank}) + C(\text{update JXP scores})$   
 $\approx 2^*(n + n^*in)*out \approx (2^n)^*2 \approx 2^*(n + n^*in)$

- Storage cost

$C(\text{local graph}) + C(\text{pages info})$   
 $\approx 2^*n^*(out + in) \approx 3^*(n + n^*in)$

□  $n$  → Avg. size of a local graph;  
 □  $out$  → Avg. outdegree;  
 □  $in$  → Avg. indegree.

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-32



### JXP: Summary

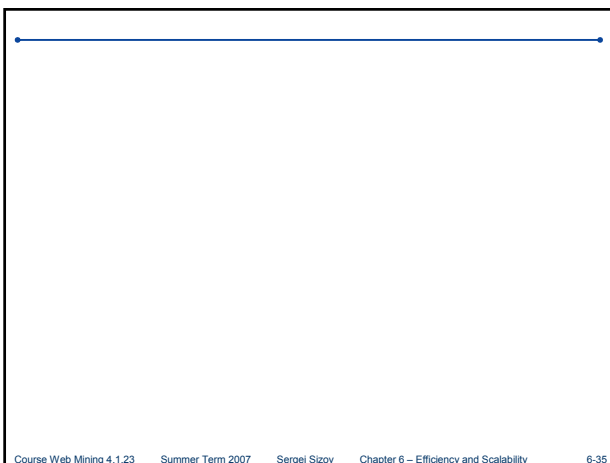
Algorithm for computing authority scores of pages in a P2P network

- Local computations only;
- Local scores approximate global scores;

Open questions

- Mathematical proof of convergence of the algorithm;
- Methods for reducing the number of meetings (e.g. by using Semantic Overlay Networks);
- Protection against cheating peers.

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-34



### Factorizing Random-Walk Computations

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

basis vectors → Decomposition Theorem → reduced basis vectors

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

Algorithmic framework:

- run power iteration  $k=1, 2, \dots$  until convergence (for computing all full vectors  $r_p$  or partial vectors)
- compute lower-approximation  $D_k[p]$  for  $r_p$ :  $D_k[p](q) \leq r_p(q)$  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  $E_0[p] = e_p$

Course Web Mining 4.1.23 Summer Term 2007 Sergei Sizov Chapter 6 – Efficiency and Scalability 6-36

### 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:

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

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

reduces the error by factor  $1-\varepsilon$  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

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

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

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):

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

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

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  $k \geq 1$

→  $D_k[p] + \varepsilon E_k[p]$  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  $P_1, \dots, P_m$  (e.g.,  $m=10$ ) with  $P_i$ :  
 $V_i$  – nodes that reside in this block,  
 $E_i$  – adjacency lists with edges  $(p,q)$  for  $p \in V_i$ ,  
 $Lk_{ij}$  – intermediate results for  $D_k[p](q)$  and  $E_k[p](q)$  with  $p \in V_i$  and  $q \in V_j$   
 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  $\approx 8$  hours,  
 for the partial vectors  $r_p - r_p^H$  it was  $10000 * 0.3$  seconds  $\approx 50$  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.