3.3 Index Access Scheduling

Given:

- index scans over m lists L_i (i=1..m), with current positions pos_i
- score predictors for score(pos) and pos(score) for each L_i
- selectivity predictors for document $d \in L_i$
- current top-k queue T with k documents
- candidate queue Q with c documents (usually c >> k)
- min-k threshold = min{worstscore(d) | $d \in T$}

Questions/Decisions:

- ***Sorted-access (SA) scheduling:***
  for the next batch of b scan steps, how many steps in which list?
  (b_i steps in L_i with $\sum_i b_i = b$)
- ***Random-access (RA) scheduling:***
  when to initiate probes and for which documents?
- Possible constraints and extra considerations:
  some dimensions i may support only sorted access or only random access, or have tremendous cost ratio C_{RA}/C_{SA}

Combined Algorithm (CA)

assume cost ratio $C_{RA}/C_{SA} = r$

perform NRA (TA-sorted)
with [worstscore, bestscore] bookkeeping in priority queue Q
and round-robin SA to m index lists

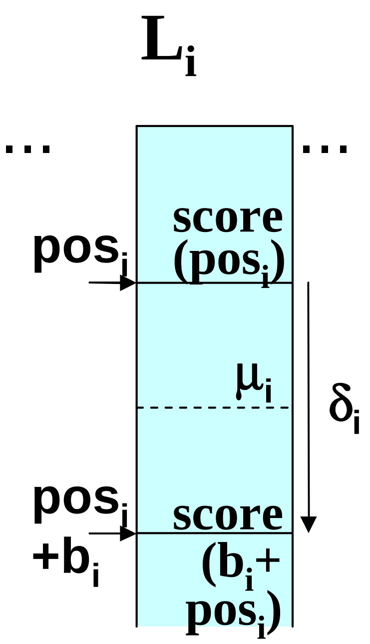
...

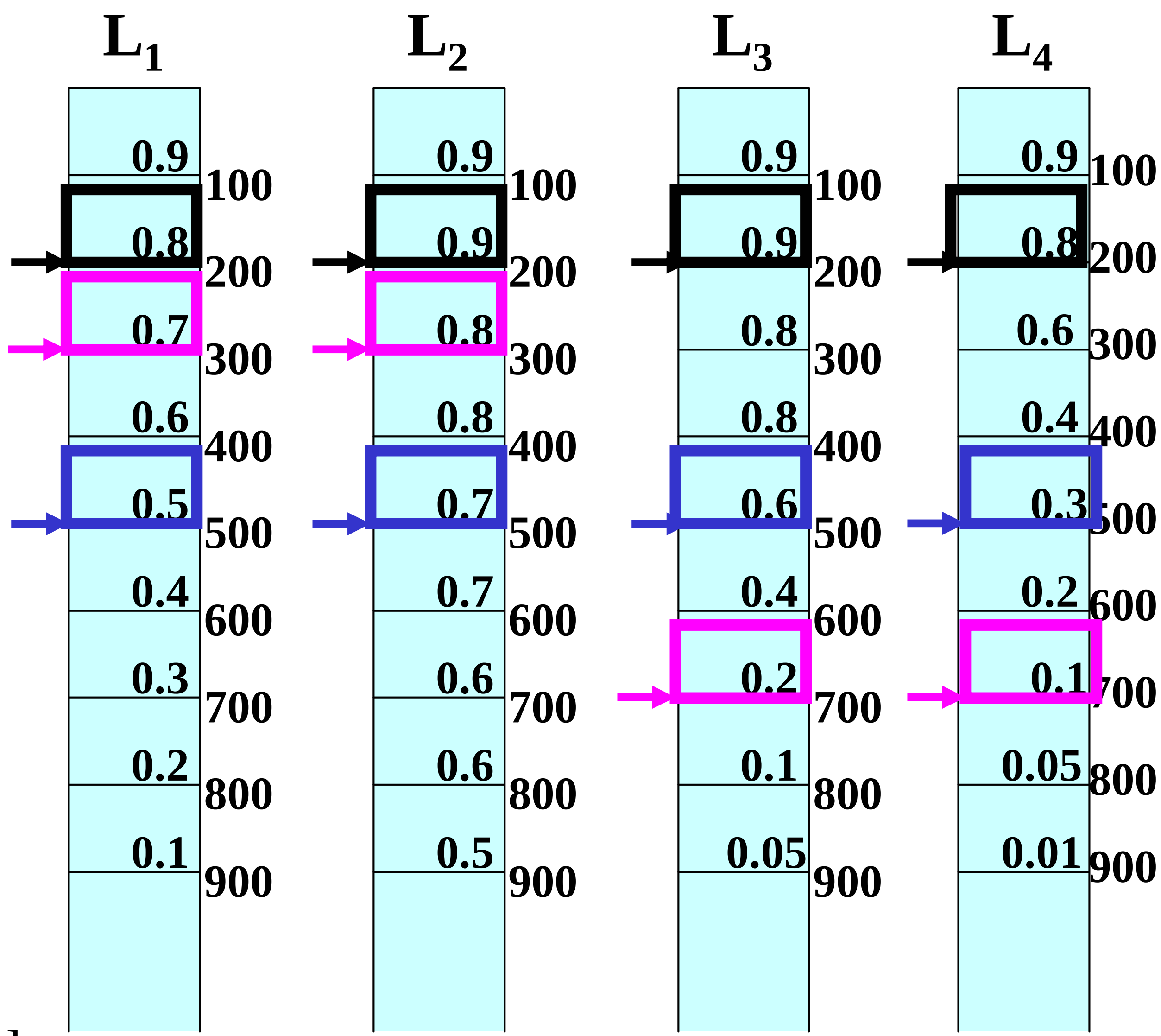
after every r rounds of SA (i.e. m*r scan steps)
perform RA to look up all missing scores of „best candidate" in Q
(where „best" is in terms of
bestscore, worstscore, or E[score], or P[score > min-k])

cost competitiveness w.r.t. „optimal schedule"
(scan until Σ_i $high_i \leq \min\{bestscore(d) \mid d \in$ final top-k$\}$,
then perform RAs for all d' with bestscore(d') > min-k): 4m + k

Sorted-Access Scheduling

available info:





goal:
eliminate candidates quickly
aim for quick drop in $high_i$ bounds

SA Scheduling: Objective and Heuristics

plan next $b_1, ..., b_m$ index scan steps
for batch of b steps overall s.t. $\Sigma_{i=1..m}\ b_i = b$
and benefit($b_1, ..., b_m$) is max!

possible benefit definitions:

$$benefit(b_1..b_m) = \sum_{i=1..m} \Delta_i$$ **with** $$\Delta_i = (high_i - score_i(pos_i + b_i)) / b_i$$

score gradient

$$benefit(b_1..b_m) = \sum_{i=1..m} \delta_i$$ **with** $$\delta_i = score_i(pos_i) - score_i(pos_i + b_i)$$

score reduction

Solve knapsack-style NP-hard optimization problem
(e.g. for batched scans) or use greedy heuristics:
$b_i := b * benefit(b_i=b) / \sum_{\nu=1..m} benefit(b_\nu=b)$

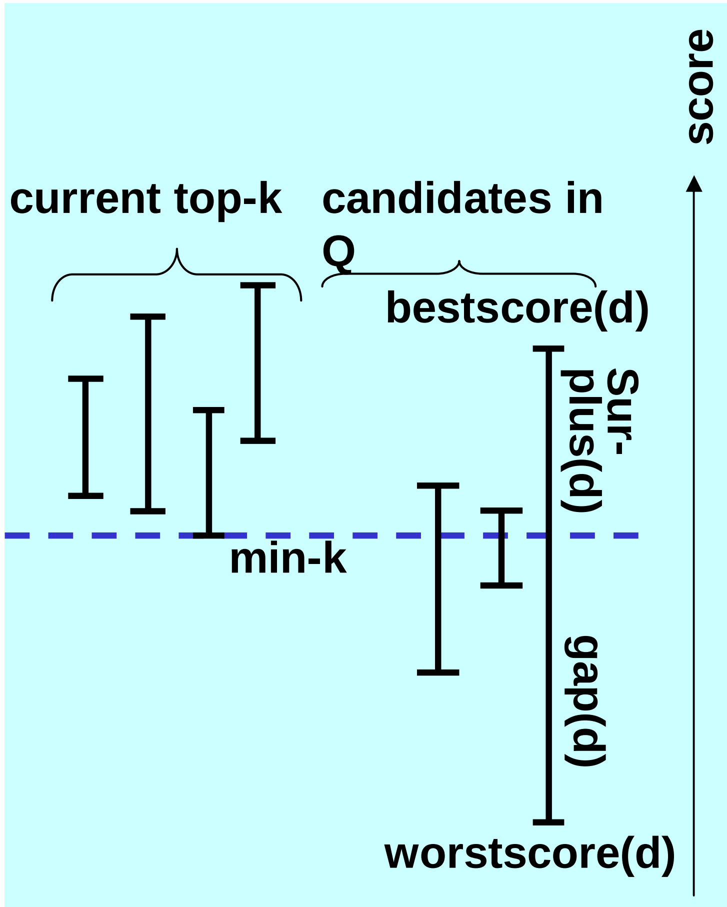
SA Scheduling: Benefit Aggregation Heuristics

Consider current top-k T and andidate queue Q;
for each d ∈ T∪Q we know E(d) ⊆ 1..m, R(d) = 1..m – E(d),
bestscore(d), worstscore(d), p(d) = P[score(d) > min-k]

$$\boldsymbol{benefit(d, b_1..b_m) =}$$
$$\boldsymbol{surplus(d)^{-1} \cdot \sum_{i \notin E(d)} (high_i - score(pos_i + b_i))}$$
$$\boldsymbol{+\, gap(d)^{-1} \cdot \sum_{i \notin E(d)} \mu_i}$$

with $surplus(d) = bestscore(d) - min\text{-}k$
$gap(d) = min\text{-}k - worstscore(d)$
$\mu_i = E[score(j) \mid j \in [pos_i, pos_i + b_i]]$

$$benefit(b_1..b_m) = \sum_{d \in T \cup Q} benefit(d, b_1..b_m)$$



weighs documents and dimensions in benefit function

Random-Access Scheduling: Heuristics

Perform additional RAs when helpful
1) to increase min-k (increase worstscore of d $\in$ top-k) or
2) to prune candidates (decrease bestscore of d $\in$ Q)

For 1) Top Probing:
- **perform RAs for current top-k (whenever min-k changes),**
- **and possibly for best d from Q**
  (in desc. order of bestscore, worstscore, or P[score(d)>min-k])

For 2) 2-Phase Probing:
perform RAs for all candidates at point t
total cost of remaining RAs = total cost of SAs up to t
(motivated by linear increase of SA-cost(t) and
sharply decreasing remaining-RA-cost(t))

Top-k Queries over Web Sources

Typical example:

Address = „2590 Broadway" and Price = $ 25 and Rating = 30
issued against mapquest.com, nytoday.com, zagat.com

Major complication:

some sources do not allow sorted access
highly varying SA and RA costs

Major opportunity:

sources can be accessed in parallel

→ extension/generalization of TA

distinguish S-sources, R-sources, SR-sources

Source-Type-Aware TA

For each R-source $S_i \in S_{m+1} .. S_{m+r}$ set $high_i$:= 1

Scan SR- or S-sources $S_1 .. S_m$

Choose SR- or S-source S_i for next sorted access

for object d retrieved from SR- or S-source L_i do {

E(d) := E(d) ∪ {i}; $high_i$:= si(q,d);

bestscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), $high_i$ for i ∉ E(d);

worstscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), 0 for i ∉ E(d); };

Choose SR- or R-source Si for next random access

for object d retrieved from SR- or R-source L_i do {

E(d) := E(d) ∪ {i};

bestscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), $high_i$ for i ∉ E(d);

worstscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), 0 for i ∉ E(d); };

current top-k := k docs with largest worstscore;

min-k := minimum worstscore among current top-k;

Stop when bestscore(d | d not in current top-k results) ≤ min-k ;

Return current top-k;

essentially NRA with choice of sources

Strategies for Choosing the Source for Next Access

<u>for next sorted acccess:</u>

Escore(Li) := expected si value for next sorted access to Li
(e.g.: $high_i$)

rank(Li) := w_i * Escore(Li) / c_{SA}(Li)
// w_i is weight of Li in aggr
// c_{SA}(Li) is source-specific SA cost

choose SR- or S-source with highest rank(Li)

<u>for next random acccess (probe):</u>

Escore(Li) := expected si value for next random access to Li
(e.g.: ($high_i$ – low_i) / 2)

rank(Li) := w_i * Escore(Li) / c_{RA}(Li)

choose SR- or R-source with highest rank(Li)

or use more advanced statistical score estimators

The Upper Strategy for Choosing Next Object and Source (Marian et al.: TODS 2004)

idea: eagerly prove that candidate objects cannot qualify for top-k

<u>for next random acccess:</u>

among all objects with E(d)≠∅ and R(d) ≠∅
  choose d' with the highest bestscore(d');
if bestscore(d') < bestscore(v) for object v with E(v)=∅ then
  perform sorted access next (i.e., don't probe d')
else {
  Δ := bestscore(d') – min-k;
  if Δ > 0 then {
    consider Li as „redundant" for d' if for all Y ⊆ R(d') – {Li}
    $\sum_{j\in Y} w_j * high_j + w_i * high_i \ge \Delta \;\; \Rightarrow \sum_{j\in Y} w_j * high_j \ge \Delta$;
    choose „non-redundant" source with highest rank(Li) }
  else choose source with lowest c_{RA}(Li);
  };

The Parallel Strategy pUpper (Marian et al.: TODS 2004)

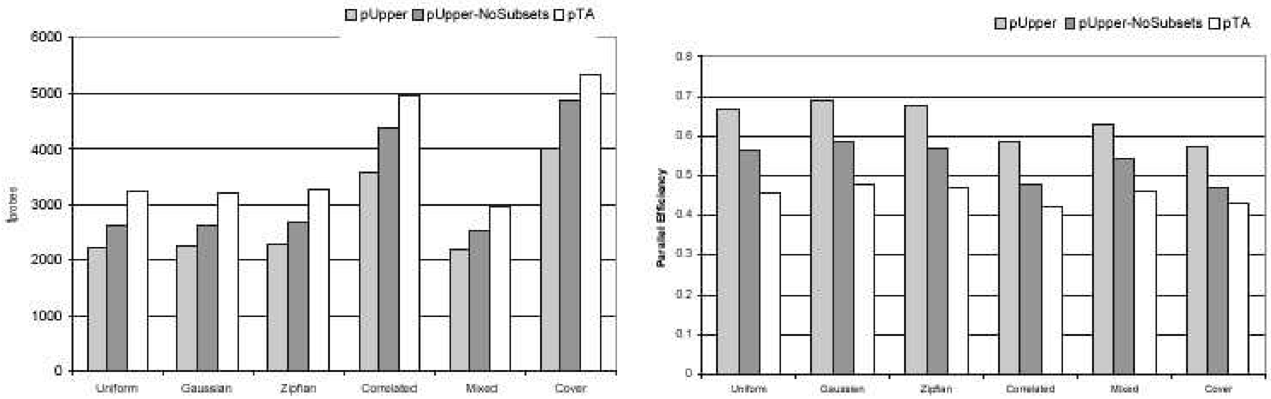
idea: consider up to MPL(Li) parallel probes to the same R-source Li
choose objects to be probed based on
bestscore reduction and expected response time

<u>for next random acccess:</u>

probe-candidates := m objects d with $E(d) \neq \emptyset$ and $R(d) \neq \emptyset$
such that d is among the m highest values of bestscore(d);
for each object d in probe-candidates do {
Δ := bestscore(d) – min-k;
if $\Delta > 0$ then {
choose subset $Y(d) \subseteq R(d)$ such that $\sum_{j \in Y} w_j * high_j \geq \Delta$
and expected response time
$\sum_{Lj \in Y(d)}$ (|{d' | bestscore(d')>bestscore(d) and $Y(d) \cap Y(d') \neq \emptyset$}|
* c_{RA}(Lj) / MPL(Lj))
is minimum };
};
enqueue probe(d) to queue(Li) for all $Li \in Y(d)$
with expected response time as priority;

Experimental Evaluation

pTA:
parallelized TA
(with asynchronous probes,
but same probe order as TA)



(a) Parallel probing time.

(b) Parallel efficiency.

Figure 19: Effect of the attribute score distribution on performance.

synthetic data

real Web sources

SR: superpages (Verizon yellow pages)

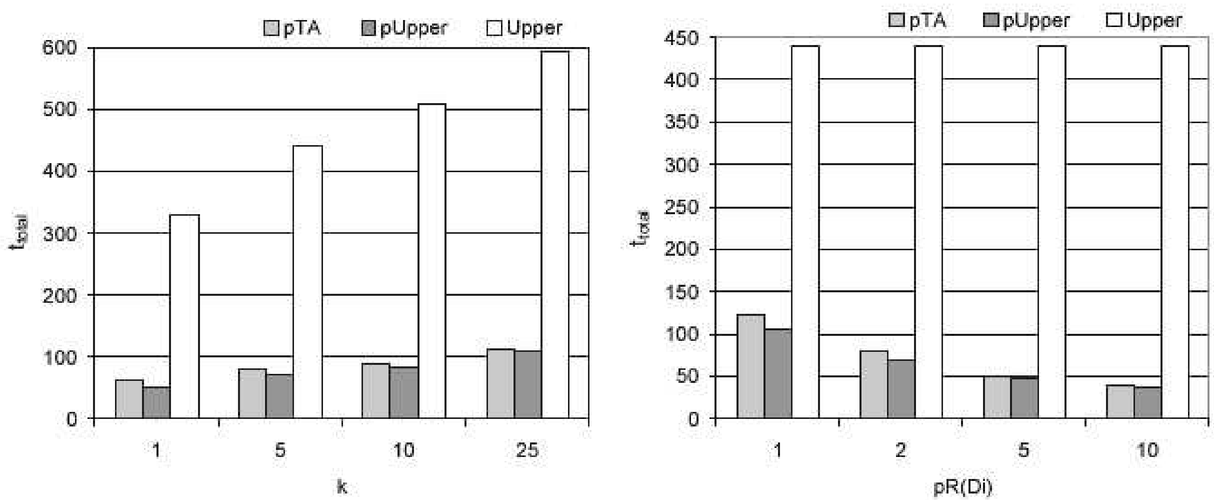
R: subwaynavigator

R: mapquest

R: altavista

R: zagat

R: nytoday



(a) Parallel time t_{total} as a function of k ($pR(D_i) = 2$).

(b) Parallel time t_{total} as a function of $pR(D_i)$ ($k = 5$).

Figure 24: Effect of the number of objects requested k (a) and the number of accesses per source $pR(D_i)$ (b) on the performance of pTA, $pUpper$, and $Upper$ over real web sources.

from: A. Marian et al., TODS 2004

3.4 Index Organization and Advanced Query Types

Richer Functionality:

- Boolean combinations of search conditions
- Search by word stems
- Phrase queries and proximity queries
- Wild-card queries
- Fuzzy search with edit distance

Enhanced Performance:

- Stopword elimination
- Static index pruning
- Duplicate elimination

Boolean Combinations of Search Conditions

combination of AND and ANDish: $(t_1$ AND … AND $t_j)$ t_{j+1} t_{j+2}… t_m

- **TA family applicable with mandatory probing in AND lists**
  → RA scheduling
- **(worstscore, bestscore) bookkeeping and pruning**
  more effective with "boosting weights" for AND lists

combination of AND, ANDish and NOT:

NOT terms considered k.o. criteria for results

TA family applicable with mandatory probing for AND and NOT

→ RA scheduling

combination of AND, OR, NOT in Boolean sense:

- **best processed by index lists in DocId order**
- **construct operator tree and push selective operators down;**
  needs good query optimizer (selectivity estimation)

Search with Morphological Reduction (Lemmatization)

Reduction onto grammatical **ground form**:
nouns onto nominative, verbs onto infinitive,
plural onto singular, passive onto active, etc.
Examples (in German):

- „Winden“ onto „Wind“, „Winde“ or „winden“ depending on phrase structure and context
- „finden“ and „gefundenes“ onto „finden“,
- „Gefundenes“ onto „Fund“

Reduction of morphological variations onto **word stem**:
flexions (e.g. declination), composition, verb-to-noun, etc.
Examples (in German):

- „Flüssen“, „einflößen“ onto „Fluss“,
- „finden“ and „Gefundenes“ onto „finden“
- „Du brachtest ... mit“ onto „mitbringen“,
- „Schweinkram“, „Schweinshaxe“ and „Schweinebraten“ onto „Schwein“ etc.
- „Feinschmecker“ and „geschmacklos“ onto „schmecken“

Stemming

Approaches:

• Lookup in comprehensive lexicon/dictionary (e.g. for German)

• Heuristic affix removal (e.g. Porter stemmer for English):

remove prefixes and/or suffixes
based on (heuristic) rules
Example:
stresses $\rightarrow$ stress, stressing $\rightarrow$ stress, symbols $\rightarrow$ symbol
based on rules: sses $\rightarrow$ ss, ing $\rightarrow$ ε, s $\rightarrow$ ε, etc.

The benefit of stemming for IR is debated.
Example:

Bill is operating a company.
On his computer he runs the Linux operating system.

Phrase Queries and Proximity Queries

phrase queries such as:

„George W. Bush“, „President Bush“, „The Who“, „Evil Empire“, „PhD admission“, „FC Schalke 04“, „native American music“, „to be or not to be“, „The Lord of the Rings“, etc. etc.

difficult to anticipate and index all (meaningful) phrases
sources could be thesauri (e.g. WordNet) or query logs

→ standard approach:
combine single-term index with separate position index

B+ tree on term

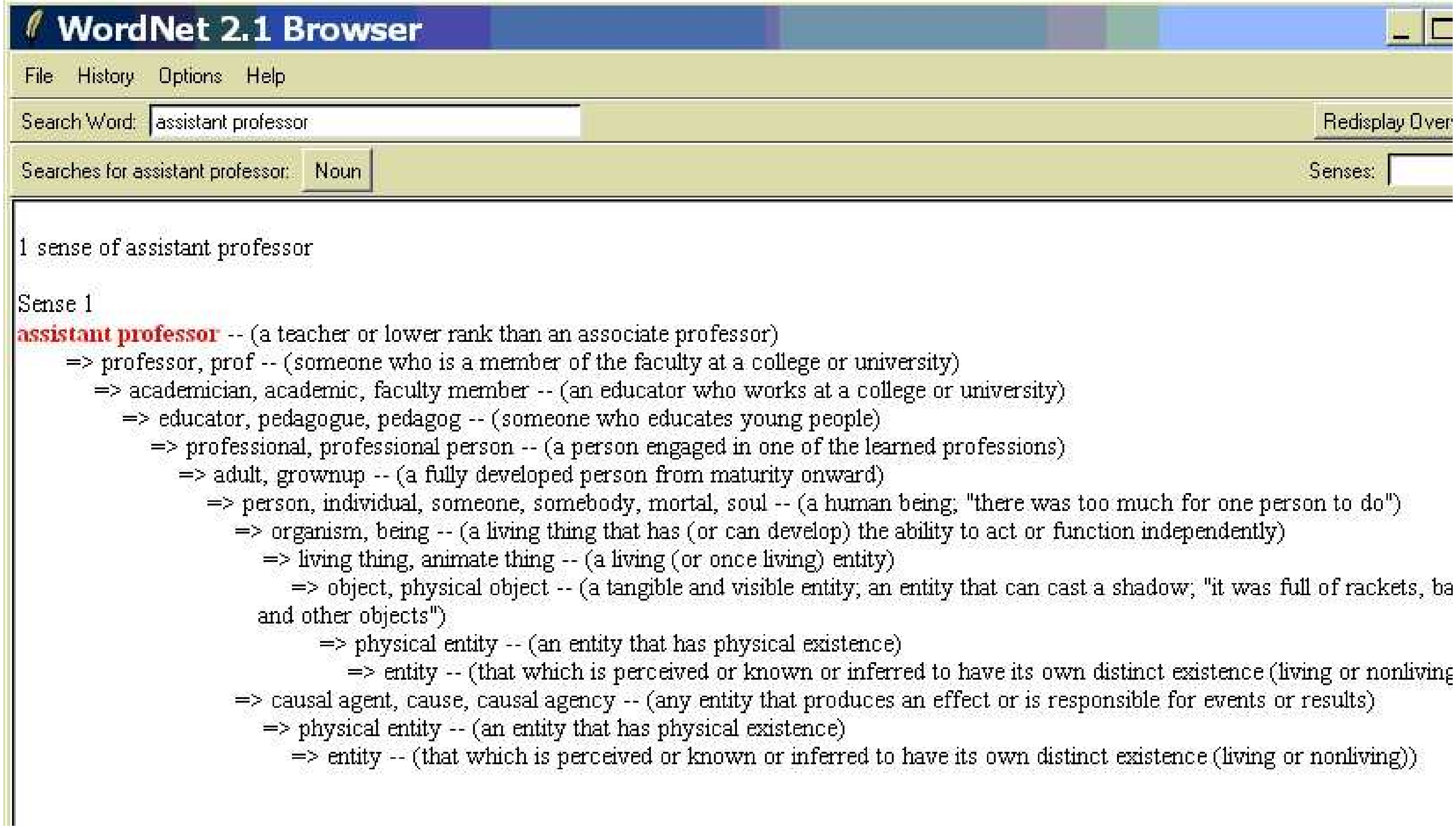
term	doc	score
...		
empire	77	0.85
empire	39	0.82
...		
evil	49	0.81
evil	39	0.78
evil	12	0.75
...		
evil	77	0.12
...		

term	doc	offset
...		
empire	39	191
empire	77	375
...		
evil	12	45
evil	39	190
evil	39	194
evil	49	190
...		
evil	77	190
...		

B+ tree on term, doc

Thesaurus as Phrase Dictionary

Example: WordNet (Miller/Fellbaum), http://wordnet.princeton.edu



Biword and Phrase Indexing

build index over all word pairs:
index lists (term1, term2, doc, score) or
for each term1 nested list (term2, doc, score)

variations:

- **treat nearest nouns as pairs,**
  or discount articles, prepositions, conjunctions
- **index phrases from query logs, compute correlation statistics**

query processing:

- **decompose even-numbered phrases into biwords**
- **decompose odd-numbered phrases into biwords**
  with low selectivity (as estimated by df(term1))
- **may additionally use standard single-term index if necessary**

Examples:
to be or not to be $\rightarrow$ (to be) (or not) (to be)
The Lord of the Rings $\rightarrow$ (The Lord) (Lord of) (the Rings)

N-Gram Indexing and Wildcard Queries

Queries with wildcards (simple regular expressions), to capture mis-spellings, name variations, etc.

<u>Examples:</u>

Brit*ney, Sm*th*, Go*zilla, Marko*, reali*ation, *raklion

<u>Approach:</u>

- **decompose words into N-grams of N successive letters and index all N-grams as terms**
- **query processing computes AND of N-gram matches**

Example (N=3):

Brit*ney $\rightarrow$ Bri AND rit AND ney

Generalization: decompose words into frequent fragments (e.g., syllables, or fragments derived from mis-spelling statistics)

Refstring Indexing (Schek 1978)

In addition to indexing all N-grams for some small N (e.g. 2 or 3), determine frequent fragments – refstrings $r \in R$ – with properties:

- **df(r) is above some threshold θ**
- **if $r \in R$ then for all substrings s of r: $s \notin R$ unless $df(s|\neg r) = |\{docs\ d \mid d \text{ contains } s \text{ but not } r\}| \geq \theta$**

Refstring index build:

1) **Candidate generation $\rightarrow$ preliminary set R:**
   generate strings r with $|r|>N$ in increasing length, compute df(r); remove r from candidates if r=xy with $df(x)< \theta$ or $df(y)< \theta$
2) **Candidate selection: consider candidate $r \in R$ with $|r|=k$ and sets**
   $left(r)=\{xr \mid xr \in R \wedge |xr|=k+1\}$, $right(r)=\{ry \mid ry \in R \wedge |ry|=k+1\}$,
   $left^{-}(r)=\{xr \mid xr \notin R \wedge |xr|=k+1\}$, $right^{-}(r)=\{ry \mid \notin R \wedge |ry|=k+1\}$
   select r if $weight(r) = df(r) - \max\{leftf(r), rightf(r)\} \geq \theta$ with
   $leftf(r) = \sum_{q \in left(r)} df(q) + \sum_{q \in left-(r)} \max\{leftf(q), rightf(q)\}$ and
   $rightf(r) = \sum_{q \in right(r)} df(q) + \sum_{q \in right-(r)} \max\{leftf(q), rightf(q)\}$

QP decomposes term into small number of refstrings contained in t

Fuzzy Search with Edit Distance

Idea:
tolerate mis-spellings and other variations of search terms and score matches based on editing distance

Examples:

1) query: Microsoft
   fuzzy match: Migrosaft
   score ~ edit distance 3
2) query: Microsoft
   fuzzy match: Microsiphon
   score ~ edit distance 5
3) query: Microsoft Corporation, Redmond, WA
   fuzzy match at token level: MS Corp., Redmond, USA

Similarity Measures on Strings (1)

Hamming distance of strings s1, s2 $\in \Sigma^*$ with |s1|=|s2|:
number of different characters (cardinality of $\{i: s1_i \neq s2_i\}$)

Levenshtein distance (edit distance) of strings s1, s2 $\in \Sigma^*$:
minimal number of editing operations on s1
(replacement, deletion, insertion of a character)
to change s1 into s2

For edit (i, j): Levenshtein distance of s1[1..i] and s2[1..j] it holds:

$$\text{edit}(0,0) = 0,\ \text{edit}(i,0) = i,\ \text{edit}(0,j) = j$$

$$\text{edit}(i,j) = \min \{ \text{edit}(i-1,j)+1,\ \text{edit}(i,j-1)+1,\ \text{edit}(i-1,j-1)+\text{diff}(i,j) \}$$

with diff (i, j) = 1 if $s1_i \neq s2_j$, 0 otherwise

$\rightarrow$ efficient computation by dynamic programming

Similarity Measures on Strings (2)

Damerau-Levenshtein distance of strings s1, s2 $\in \Sigma^*$:
minimal number of replacement, insertion, deletion, or transposition operations (exchanging two adjacent characters) for changing s1 into s2

For edit (i, j): Damerau-Levenshtein distance of s1[1..i] and s2[1..j] :

$$edit(0,0) = 0,\ edit(i,0) = i,\ edit(0,j) = j$$

$$edit(i,j) = \min \{ edit(i-1,j) + 1,\ edit(i,j-1) + 1,\ edit(i-1,j-1) + diff(i,j),\ edit(i-2,j-2) + diff(i-1,j) + diff(i,j-1) + 1 \}$$

with $diff(i,j) = 1$ if $s1_i \neq s2_j$, 0 otherwise

Similarity based on N-Grams

Determine for string s the set of its N-Grams:

G(s) = {substrings of s with length N}

(often trigrams are used, i.e. N=3)

Distance of strings s1 and s2:

$|G(s1)| + |G(s2)| - 2|G(s1) \cap G(s2)|$

<u>Example:</u>

G(rodney) = {rod, odn, dne, ney}

G(rhodnee) = {rho, hod, odn, dne, nee}

distance (rodney, rhodnee) = 4 + 5 – 2*2 = 5

Alternative similarity measures:

Jaccard coefficient: $|G(s1) \cap G(s2)| / |G(s1) \cup G(s2)|$

Dice coefficient: $2\,|G(s1) \cap G(s2)| / (|G(s1)| + |G(s2)|)$

N-Gram Indexing for Fuzzy Search

Theorem (Jokinen and Ukkonen 1991):
for query string s and a target string t,
the Levenshtein edit distance is bounded by the N-Gram overlap:

$$edit(s,t) \le d \Rightarrow \left| Ngrams(s) \cap Ngrams(t) \right| \ge |s| - (N-1) - dN$$

$\rightarrow$ **for fuzzy-match queries with edit-distance tolerance d,**
perform top-k query over Ngrams,
using count for score aggregation

Phonetic Similarity (1)

Soundex code:

Mapping of words (especially last names) onto 4-letter codes such that words that are similarly pronounced have the same code

- first position of code = first letter of word
- code positions 2, 3, 4 (a, e, i, o, u, y, h, w are generally ignored):

b, p, f, v	$\rightarrow$ 1	c, s, g, j, k, q, x, z	$\rightarrow$ 2
d, t	$\rightarrow$ 3	l	$\rightarrow$ 4
m, n	$\rightarrow$ 5	r	$\rightarrow$ 6

- Successive identical code letters are combined into one letter (unless separated by the letter h)

Examples:

Powers $\rightarrow$ P620 , Perez $\rightarrow$ P620

Penny $\rightarrow$ P500, Penee $\rightarrow$ P500

Tymczak $\rightarrow$ T522, Tanshik $\rightarrow$ T522

Phonetic Similarity (2)

Editex similarity:
edit distance with consideration of phonetic codes

For editex (i, j): Editex distance of s1[1..i] and s2[1..j] it holds:

$$
\begin{aligned}
&\text{editex}(0, 0) = 0,\\
&\text{editex}(i, 0) = \text{editex}(i-1, 0) + d(s1[i-1], s1[i]),\\
&\text{editex}(0, j) = \text{editex}(0, j-1) + d(s2[j-1], s2[j]),\\
&\text{editex}(i, j) = \min \{\ \text{editex}(i-1, j) + d(s1[i-1], s1[i]),\\
&\qquad\qquad\qquad\qquad \text{editex}(i, j-1) + d(s2[j-1], s2[j]),\\
&\qquad\qquad\qquad\qquad \text{edit}(i-1, j-1) + \text{diffcode}(i, j)\ \}
\end{aligned}
$$

with $\text{diffcode}(i, j) = 0$ if $s1_i = s2_j$,
1 if $\text{group}(s1_i) = \text{group}(s2_j)$, 2 otherwise

und $d(X, Y) = 1$ if $X \neq Y$ and X is h or w,
diffcode (X, Y) otherwise

with group:
{a e i o u y}, {b p}, {c k q}, {d t}, {l r},
{m n}, {g j}, {f p v}, {s x z}, {c s z}

3.4 Index Organization and Advanced Query Types

Richer Functionality:

- Boolean combinations of search conditions
- Search by word stems
- Phrase queries and proximity queries
- Wild-card queries
- Fuzzy search with edit distance

Enhanced Performance:

- Stopword elimination
- Static index pruning
- Duplicate elimination

Stopword Elimination

Lookup in stopword list
(possibly considering domain-specific vocabulary,
e.g. „definition“ or „theorem“ in math corpus

Typical English stopwords
(articles, prepositions, conjunctions, pronouns,
„overloaded“ verbs, etc.):

a, also, an, and, as, at, be, but, by,
can, could, do, for, from, go,
have, he, her, here, his, how,
I, if, in, into, it, its,
my, of, on, or, our, say, she,
that, the, their, there, therefore, they,
this, these, those, through, to, until,
we, what, when, where, which, while, who, with, would,
you, your

Static Index Pruning (Carmel et al. 2001)

Scoring function S' is an ε-variation of scoring function S if
$(1-\varepsilon)S(d) \le S'(d) \le (1+\varepsilon)S(d)$ for all d

Scoring function S_q' for query q is (k, ε)-good for S_q if
there is an ε-variation S' of S_q such that
the top-k results for S_q' are the same as those for S'.
S_q' for query q is (δ, ε)-good for S_q if
there is an ε-variation S' of S_q such that
the top- δ results for S_q' are the same as those for S',
where top- δ results are all docs with score above δ*score(top-1)

Given k and ε, prune index lists so as to guarantee (k, εr)-good results for all queries q with r terms where r < 1/ ε.

→ for each index list Li, let $s_{i(k)}$ be the rank-k score;
prune all Li entries with score < ε* $s_{i(k)}$

Efficiency and Effectiveness of Static Index Pruning

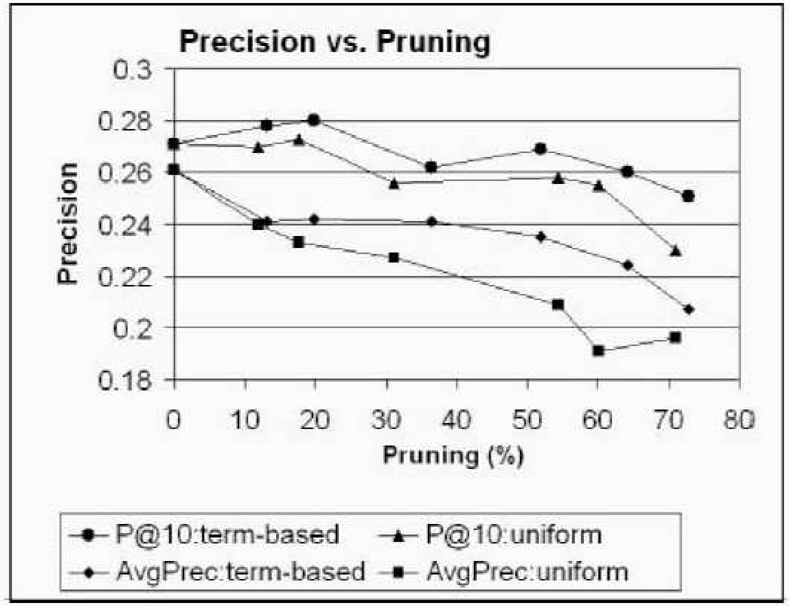


Figure 2: Precision of search results at varying levels of pruning.

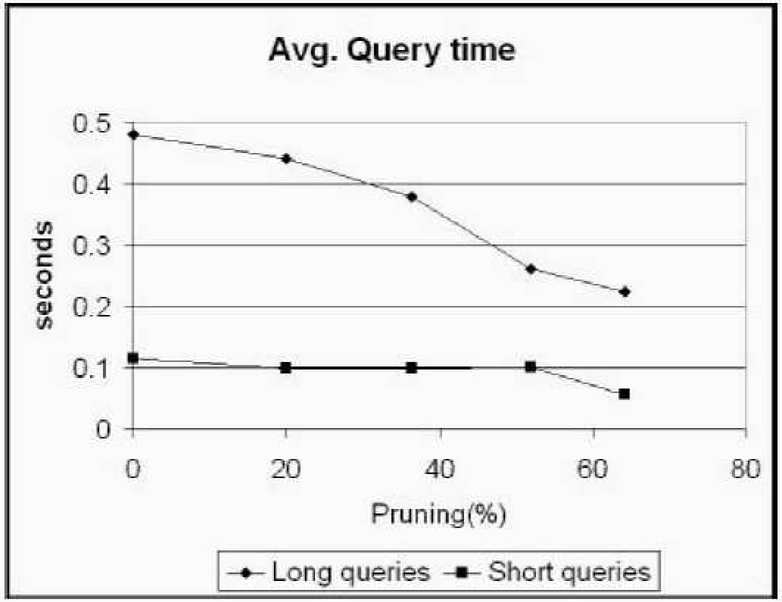


Figure 4: Average query processing time at varying levels of pruning.

from: D. Carmel et al., Static Index Pruning for Information Retrieval Systems, SIGIR 2001

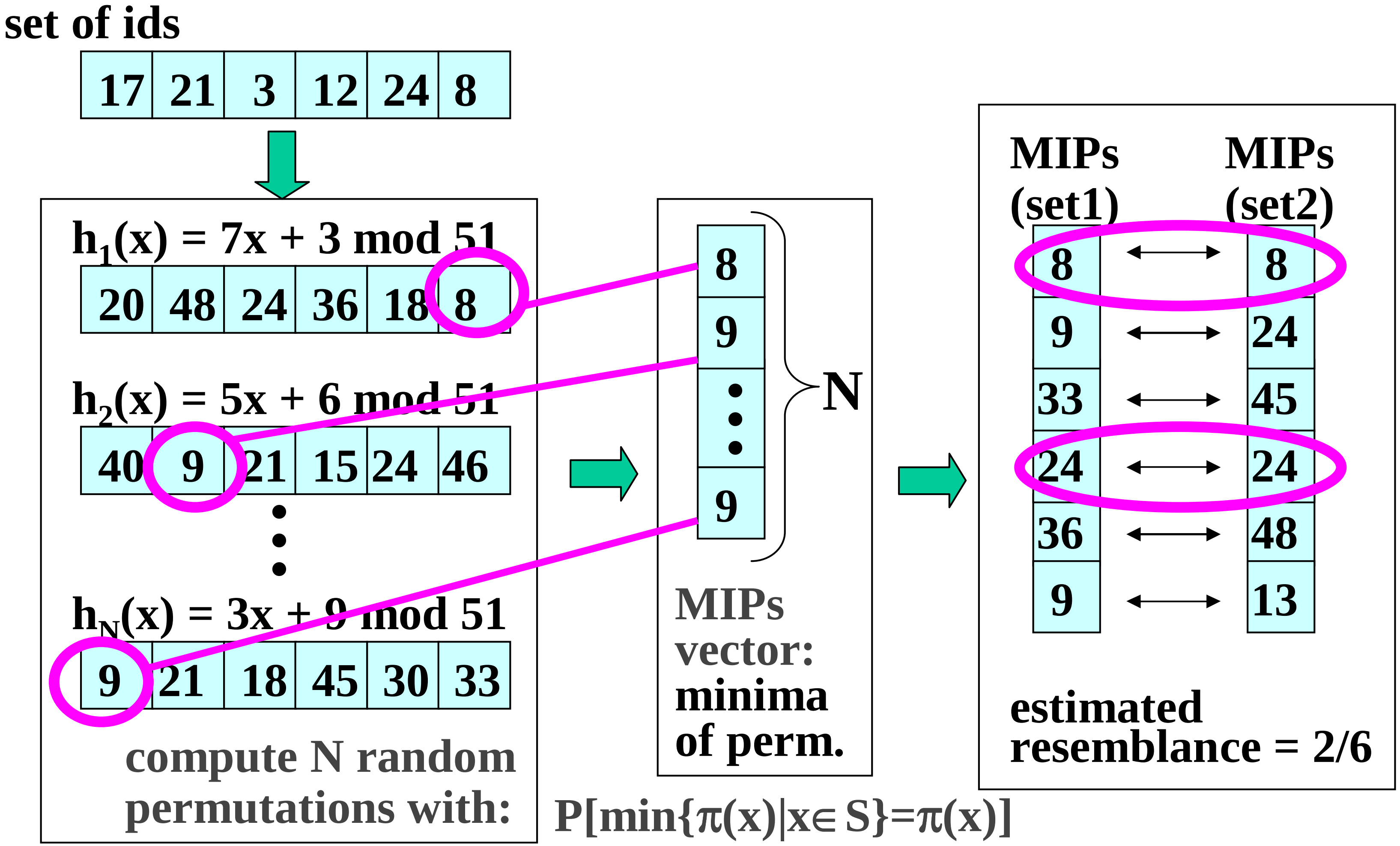
Duplicate Elimination (Broder 1997)

duplicates on the Web may be slightly perturbed
crawler & indexing interested in identifying near-duplicates

<u>Approach:</u>

- **represent each document d as set (or sequence) of shingles (N-grams over tokens)**
- **encode shingles by hash fingerprints (e.g., using SHA-1), yielding set of numbers $S(d) \subseteq [1..n]$ with, e.g., $n=2^{64}$**
- **compare two docs d, d‘ that are suspected to be duplicates by**
  - **resemblance:** $\dfrac{|S(d) \cap S(d')|}{|S(d) \cup S(d')|}$
  - **containment:** $\dfrac{|S(d) \cap S(d')|}{|S(d)|}$
- **drop d‘ if resemblance or containment is above threshold**

Min-Wise Independent Permutations (MIPs)



$P[\min\{\pi(x) | x \in S\} = \pi(x)] = 1/|S|$

MIPs are unbiased estimator of resemblance:

$P\ [\min\ \{h(x)\ |\ x \in A\} = \min\ \{h(y)\ |\ y \in B\}] = |A \cap B|\ /\ |A \cup B|$

MIPs can be viewed as repeated sampling of x, y from A, B

Efficient Duplicate Detection in Large Corpora

avoid comparing all pairs of docs

Solution:

1) **for each doc compute shingle-set and MIPs**
2) **produce (shingleID, docID) sorted list**
3) **produce (docID1, docID2, shingleCount) table with counters for common shingles**
4) **Identify (docID1, docID2) pairs with shingleCount above threshold and add (docID1, docID2) edge to graph**
5) **Compute connected components of graph (union-find) → these are the near-duplicate clusters**

Trick for additional speedup of steps 2 and 3:
- compute super-shingles (meta sketches) for shingles of each doc
- docs with many common shingles have common super-shingle w.h.p.

Additional Literature for Chapter 3

Top-k Query Processing:

- Grossman/Frieder Chapter 5
- Witten/Moffat/Bell, Chapters 3-4
- A. Moffat, J. Zobel: Self-Indexing Inverted Files for Fast Text Retrieval, TOIS 14(4), 1996
- R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware, J. of Computer and System Sciences 66, 2003
- S. Nepal, M.V. Ramakrishna: Query Processing Issues in Image (Multimedia) Databases, ICDE 1999
- U. Guentzer, W.-T. Balke, W. Kiessling: Optimizing Multi-FeatureQueries in Image Databases, VLDB 2000
- C. Buckley, A.F. Lewit: Optimization of Inverted Vector Searches, SIGIR 1985
- M. Theobald, G. Weikum, R. Schenkel: Top-k Query Processing with Probabilistic Guarantees, VLDB 2004
- M. Theobald, R. Schenkel, G. Weikum: Efficient and Self-Tuning Incremental Query Expansion for Top-k Query Processing, SIGIR 2005
- X. Long, T. Suel: Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003
- A. Marian, N. Bruno, L. Gravano: Evaluating Top-k Queries over Web-Accessible Databases, TODS 29(2), 2004

Additional Literature for Chapter 3

Index Organization and Advanced Query Types:

- Manning/Raghavan/Schütze, Chapters 2-6, http://informationretrieval.org/
- H.E. Williams, J. Zobel, D. Bahle: Fast Phrase Querying with Combined Indexes, ACM TOIS 22(4), 2004
- WordNet: Lexical Database for the English Language, http://wordnet.princeton.edu/
- H.-J. Schek: The Reference String Indexing Method, ECI 1978
- D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y.S. Maarek, A. Soffer: Static Index Pruning for Information Retrieval Systems, SIGIR 2001
- G. Navarro: A guided tour to approximate string matching, ACM Computing Surveys 33(1), 2001
- G. Navarro, R. Baeza-Yates, E. Sutinen, J. Tarhio: Indexing Methods for Approximate String Matching. IEEE Data Engineering Bulletin 24(4), 2001
- A.Z. Broder: On the Resemblance and Containment of Documents, Compression and Complexity of Sequences Conference 1997
- A.Z. Broder, M. Charikar, A.M. Frieze, M. Mitzenmacher: Min-Wise Independent Permutations, Journal of Computer and System Sciences 60, 2000