4. Simple IR Models and their Implementation

Outline

IR model

Implicit as well as explicit aspects:

- Assumptions, for example:
  - no matter if word occurs once or ten times in the document
  - independence of individual index terms
- Representation of documents and queries
- Retrieval function
  - calculation of relevance score for a query and a document
  - approximation of the relevance probability

4.1 Pattern Matching

- Very simple IR model
- Submodel of some IR models
- Task: search for a string (search pattern) in longer text
- Realization:
  - compare sequentially search pattern of length m from left to right with the text of length n to be searched
    $\Rightarrow O(m * n)$
  - better algorithms: Boyer-Moore-Horspool-Sunday algorithm, Knuth-Morris-Pratt algorithm
    $\Rightarrow O(n)$

4.2 Word Search

- Very simple IR model
- Task:
  - given: word W
  - result: all documents containing W
- possible extension: use stem or base form reduction

Word set

Extension to word set $\{W_1, ..., W_k\}$

- Search for documents containing all given words
- Implicit AND combination over the searched words
- First step to Boolean retrieval

Difference to pattern matching

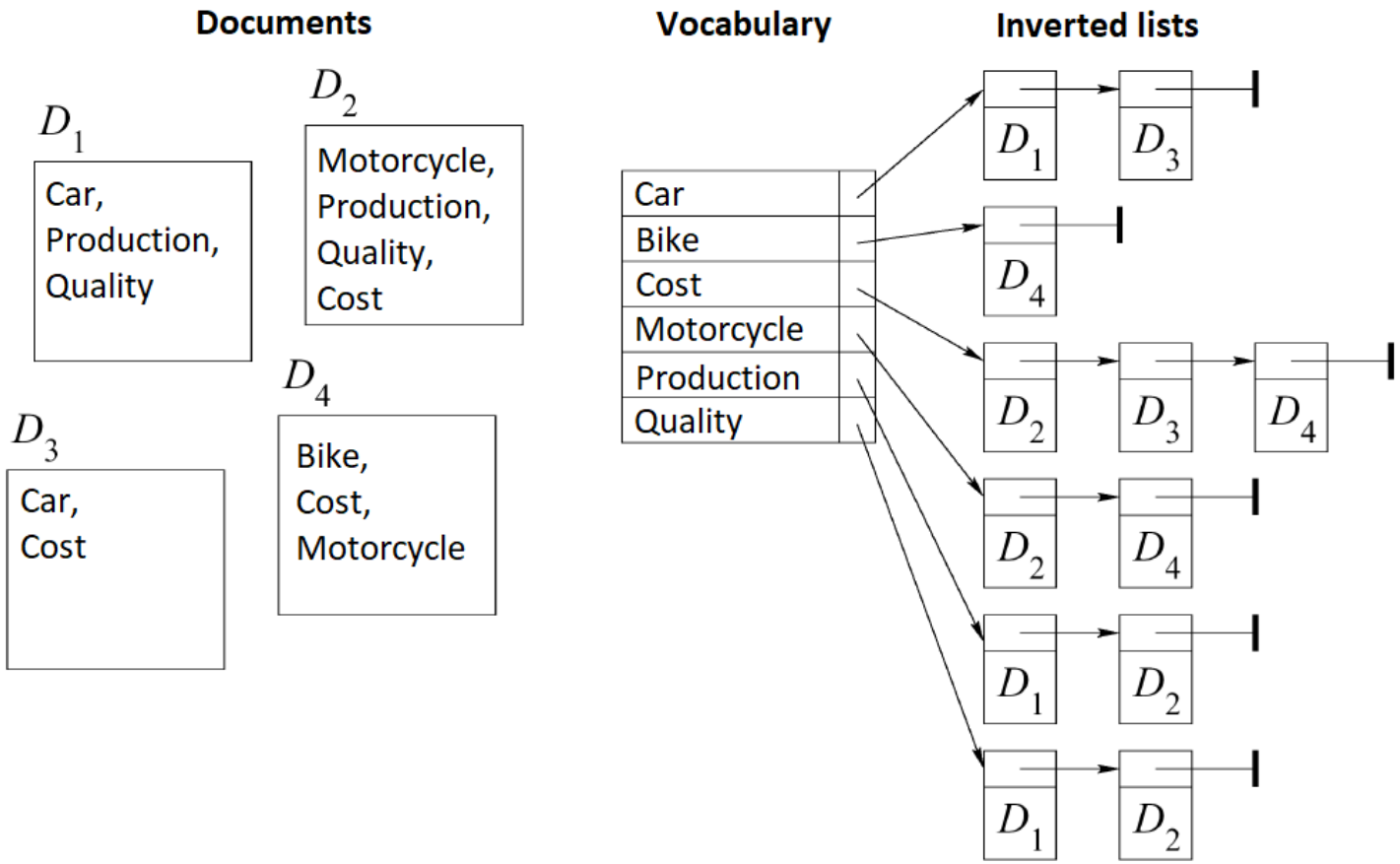
- No test whether given pattern or search word is contained in one text, but determination of set of documents containing search word
  $\Rightarrow$ extension to word sets
- Search word must appear in the document as a word

Implementation approaches for word search

- Pattern matching algorithm on each document and subsequent test for word
  - problem: search effort grows linearly with the size of the document collection
- Create index structure in advance
  - inverted lists
  - signatures

Inverted lists

For each word a sorted list of documents containing this word is managed



Word search: search word

Search for all documents that contain exactly the search word

- Scanning through the inverted list for the given search word

Word search: same stem

- Search for occurrences with the same stem:
  1. form stem for search word
  2. create temporary list set: all lists for words whose stem is identical to the stem of the search word
  3. form union of the document IDs of the lists
  4. documents of the union set form the result of the query

Word search: same stem

- Efficient determination of the union set over multiple lists using *sorting of the lists by document IDs*
  - lists are scanned in parallel
  - exploiting the sorting:
    1. initialize all lists to the first element
    2. smallest current element gets into result list
    3. lists with smallest current element are set to next element
    4. if there are still elements in total
       $\Rightarrow$ step 2

Word search: by word set

- Intersection of affected lists
- Lists are scanned in parallel
- Exploiting the sorting of the lists
  1. initialize all lists to the first element
  2. all current elements the same
     $\Rightarrow$ transfer to results list
  3. all current elements not equal
     $\Rightarrow$ lists with smallest element are set to next element
  4. if there are still elements in total
     $\Rightarrow$ step 2

Implementation of inverted lists

Required storage space:

- Example TREC collection
  - contains article from *Financial Times*
  - size of the document collection is 564 MB
  - collection contains 210,158 articles, each of which has an average of about 150 different words
  - assumption: 4 bytes per document ID
    $\Rightarrow$ 210,158 * 150 * 4 bytes = 126,094,800 bytes = 120 MB
  - disregarded: terms themselves, administrative data, frequency of occurrence, location
  - conclusion: index can quickly reach 20% of the document collection
    $\Rightarrow$ background storage

Zipf's law

Observations while counting words in texts:

- A few words occur very frequently
- English texts:
  - the two most frequently occurring words represent 10% of all occurrences
  - the six most frequent words represent 20% of all occurrences
  - the fifty most frequent words represent 50% of all occurrences
- Very many words occur very rarely

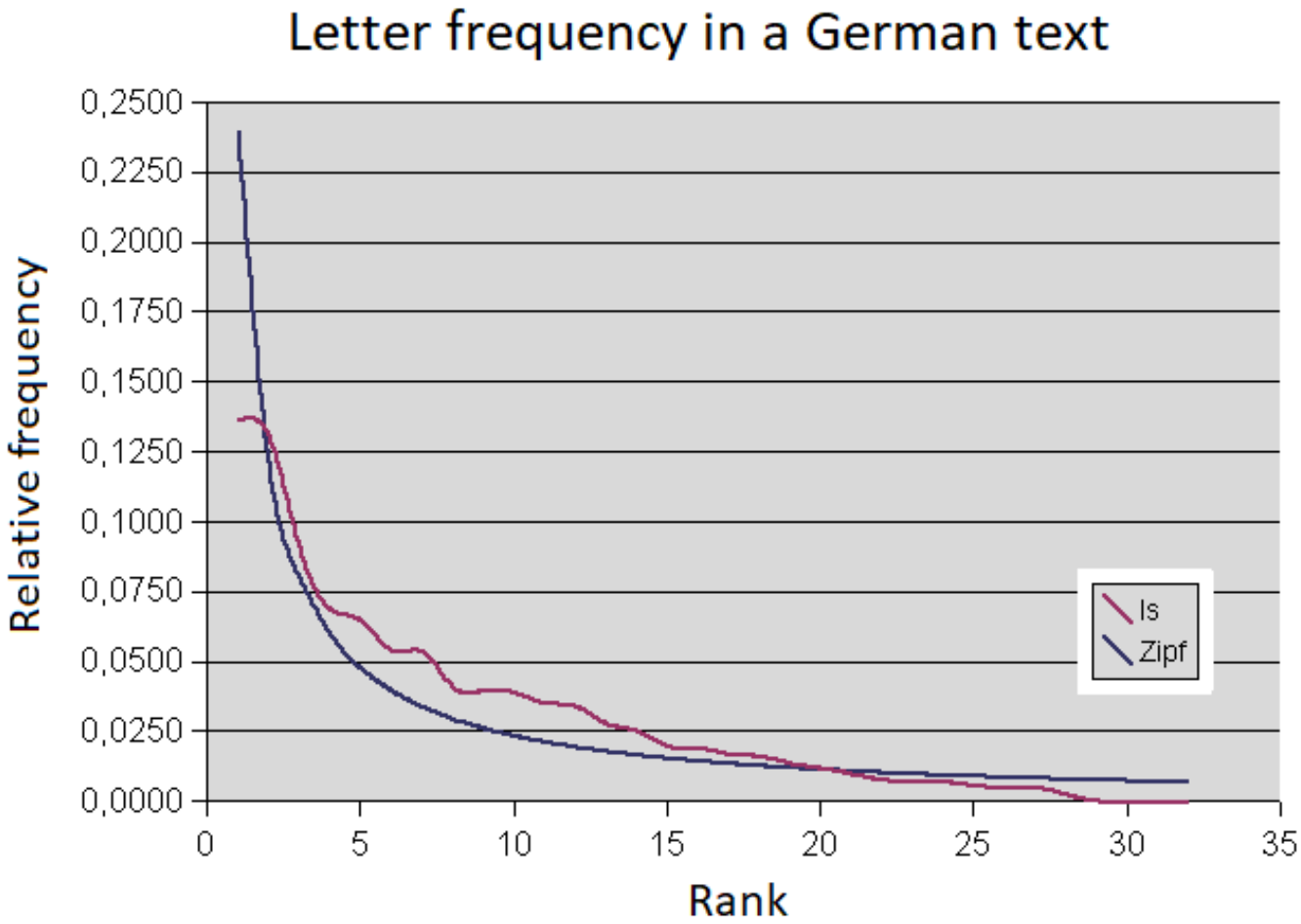
Zipf's Law: Brown and Lob

Rank r(w)	Frequency h(w)	$\frac{r(w) \cdot h(w)}{100000}$	Word or Term
1	138323	1.3832	the
2	72159	1.4432	of
3	56750	1.7025	and
4	52941	2.1176	to
5	46523	2.3262	a
6	42603	2.5562	in
7	22177	1.5524	that
⋮	⋮	⋮	⋮
430	443	1.9049	following
431	443	1.9093	short
⋮	⋮	⋮	⋮
2804	73	2.0469	destroy
2805	73	2.0476	determination
⋮	⋮	⋮	⋮
12032	11	1.3235	would-be
12033	11	1.3236	yachting
12034	11	1.3237	yell

Zipf's law

- Sort words by frequency of occurrence
  $\Rightarrow$ product of rank number and frequency of occurrence is constant (Zipf's Law)
  - most common word about 7 million times
  - second most common word about 3.5 million times
  - third most common word about 2.3 million times
- Formal:
  - representative text corpus C
  - $W(C)$ is the set of words that occur in C
  - $h(w)$ is frequency of which a word $w \in W(C)$ occurs in the corpus
  - let $r(w)$ denote the rank of $w \in W(C)$ when the words are sorted by decreasing frequency
  - then: $r(w) * h(w) \approx c(constant), \forall w \in W(C)$

Zipf's Law



Example from Wikipedia

Zipf's Law and inverted lists

Conclusion:

- Few long lists
  despite removal of stop words
- Many short lists
  despite removal of very rare words (Crouch)
- Search structure needed to map words to corresponding list
  $\Rightarrow$ e.g. B-tree or digital trees

B^+-tree

- Known from lecture database technology
  - application in databases and file systems
- Balanced tree
- Storage of data sorted by keys
- Leaf nodes contain data

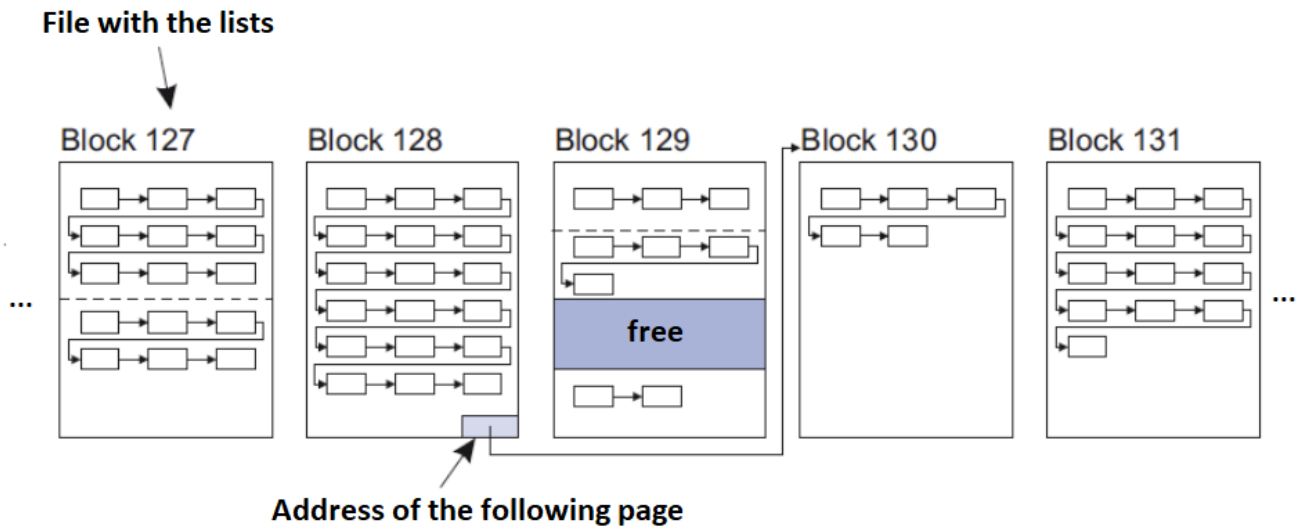
Inverted list management

- Persistent management on background storage
- Management in blocks/pages
  - 1:1 mapping between lists and pages not useful due to variability of list lengths
  - several small lists on one page
  - large lists across multiple pages
- Multilevel addressing of a list
  1. file identifier
  2. block number in file
  3. start address in block (additionally length)

Inverted list management

Management table →

Term	Block number	Length	Start position
...	...	...	...
Goldfisch	129	256	0
Hund	128	1024	0
Katze	127	512	512
Kuh	131	1024	0
...	...	...	...
Schlange	129	256	768
Tiger	127	512	0
...	...	...	...
Zebra	129	256	256



Signatures

- Alternative to inverted lists
- Hash mapping l-character word $w_1 w_2 ... w_l$ to signature (bit string of fixed length F)
- Efficient comparison operations on signatures
- Example hash mapping:

```
hash := 0;
for i := 1 to l do
    hash := (hash + w_i) * 157;
end
hash := hash mod 2^F;
```

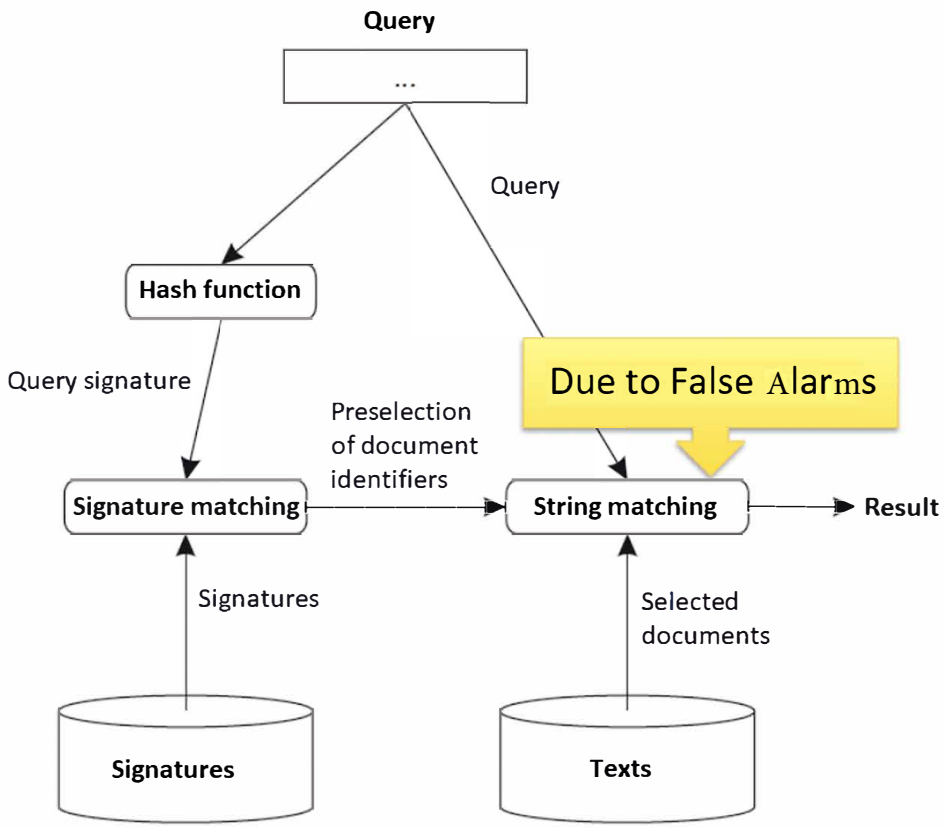
Signature properties

- Hash function h usually maps large domain to small range (filter)
- $h(word_1) = h(word_2)$ with $word_1 \neq word_2$ (collision) is possible, i.e. different words get same hash value
- Operations on signatures are therefore not exact

Signature file idea

- Signature file as inexact filter
- Check necessary condition for document to contain word
- Inexact $\Rightarrow$ although fulfilling the condition, document does not always contain word (false alarm)
- False alarms must be filtered out afterwards
- But: post-filter on relatively few documents

Signature file: architecture



Motivation for signatures

- Pattern matching
  - all documents must always be scanned
- Signatures
  - signature comparison faster than pattern matching
  - candidate selection
  - pattern matching on smaller set of documents

Overlaying signatures

Superimposed Coding:

- Mapping of 1 word to 1 signature consisting of F bits with m (signature weight) bits set
  $\Rightarrow \binom{F}{m}$ different signatures possible
- Signatures of several words are bitwise by OR combined $\Rightarrow$ block signature
  $\Rightarrow$ Fewer signatures per document

Block signature: example

F=12, m=4

Word	Signature
free	001 000 110 010
text	000 010 111 000
Block signature	001 010 111 010

Hash function

Problem: hash value should generate m set bits per word in bitcode of length F

- Use more than m hash functions with mapping to $[1...F]$
- Results indicate position of the set bits
- Equal positions possible
  $\Rightarrow$ use of the m first different positions

Hash function

- i-th hash function with mapping to $[0...F-1]$
- Argument: l-character word $w_1 w_2 ... w_l$
- i-th prime number p_i

```
hash_i
for k := 1 to l do
    hash_i := (hash_i + w_k) * p_i;
end
hash_i := hash_i mod F;
```

Parameters for superimposed coding

- Signature length F
- Signature weight m
- Overlay factor D:
  number of word signatures per block signature

Logical blocks

- Documents are divided into logical blocks (and a rest block)
- A logical block contains exactly D different words to be indexed
- Each logical block has a block signature
- Result: multiple block signatures for one document

Search for documents

Search for documents containing word w

- Calculation of word signature S_w for word w
- Bitwise AND combination of S_w with each block signature S_{b_i} of the signature file
- If $S_w = S_w \wedge S_{b_i}$, then corresponding document is in candidate set
- Removal of false alarms from candidate set using pattern matching

Conjunctive query

- Several search words combined with AND
- $\Rightarrow$ OR combination the signatures of the search words of the query
- $\Rightarrow$ Only documents found where all query words are within one block

Block	Search word	Signature
1	Information	001
2	Retrieval	100

- Query "Information Retrieval" (101) would not find the document

Conjunctive query

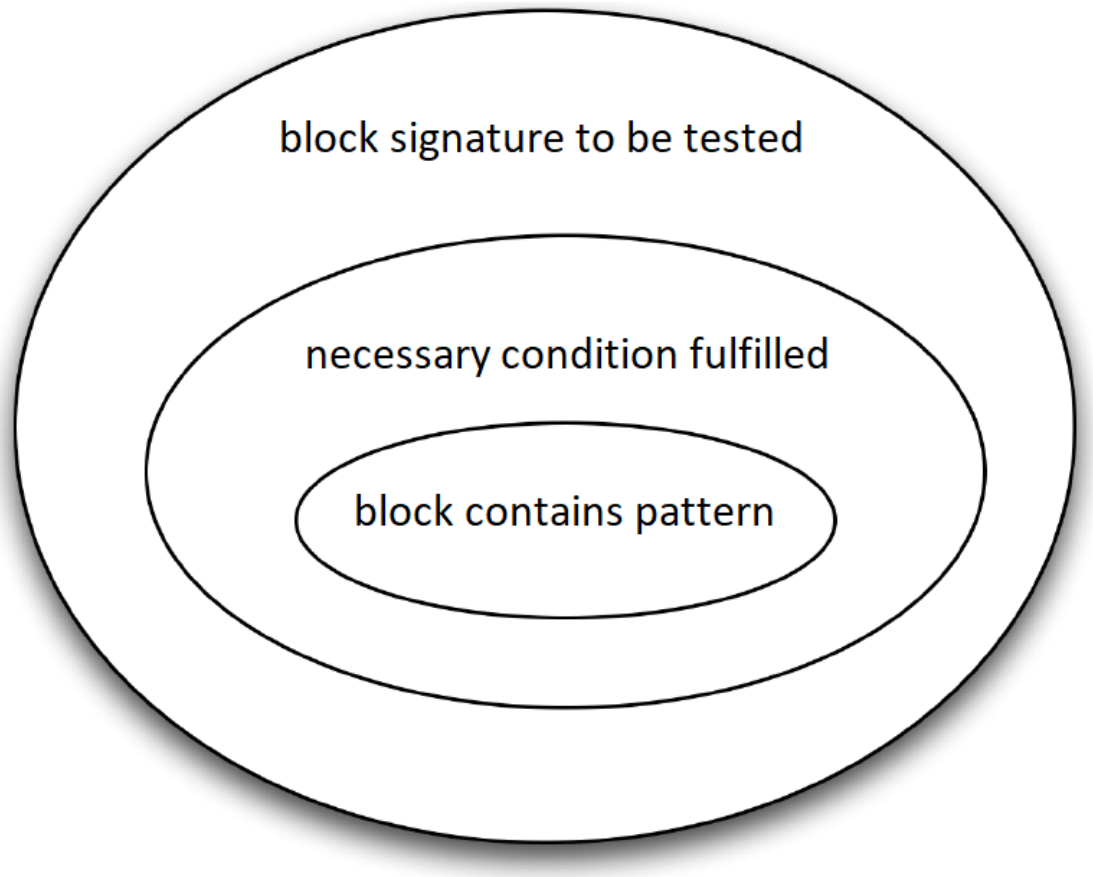
- Alternative:
  - determine candidate set for each query word
  - before pattern matching $\Rightarrow$ set intersection
- Advantage: procedure works always
  - How to proceed with OR combination?

False Alarms

- Can occur with signature files
- This means:
  - block or its block signature fulfills necessary condition
  - the pattern to be searched for is not contained in the block or the document represented
- Reasons:
  - hash function can map different words to identical signature
  - by overlaying the signatures

False alarms

False alarms = necessary condition fulfilled \ block contains pattern



False alarms: example

- Example:
  - let: $F = 12$, $m = 3$ and $D = 3$
  - query *'text'* AND *'search'*

Word	Signature
text	010 010 001 000
search	010 000 100 100
method	100 100 000 001
full	010 110 000 000
knowledge	000 111 000 000
retrieval	000 000 111 000
lexicon	100 010 010 000

False alarms: example

Word	Signature
text	010 010 001 000
search	010 000 100 100
method	100 100 000 001
full	010 110 000 000
knowledge	000 111 000 000
retrieval	000 000 111 000
lexicon	100 010 010 000

Text blocks	Block signatures
text search method	110 110 101 101
search knowledge retrieval	010 111 111 100
full text search	010 110 101 100
lexicon knowledge retrieval	100 111 111 000

Candidates are the first 3 text blocks

2nd block is false alarm

Minimize the number of false alarms

- Right choice of design parameters F, m and D
- Goals:
  - minimal storage space for signatures
  - minimal probability of false alarms

  $\Rightarrow$ conflicting goals

Error probability F_d

- Probability that signature or block signature condition is met, but represented document does not contain the pattern to be searched for
- Formal definition using probability:
  - $F_d = P$ (search pattern not in block $\wedge$ signature fulfills condition)
- Assumptions:
  - query consists of one word
  - equal distribution of signatures

Calculation of the error probability

- $\frac{m}{F}$: probability that a bit of a word signature is set
- $(1 - \frac{m}{F})^D$: probability that bit in block signature is not set (overlay due to D)
- $1 - (1 - \frac{m}{F})^D$: probability that bit in block signature is set
- $(1 - (1 - \frac{m}{F})^D)^m$: probability that block signature matches query (m bits set must match; candidate set)
- If all candidates are false alarms (very pessimistic), then $F_d = (1 - (1 - \frac{m}{F})^D)^m$
- F increases $\Rightarrow$ F_d decreases monotonically
- D increases $\Rightarrow$ F_d increases monotonically
- m increases $\Rightarrow$ F_d can decrease or increase
  $\Rightarrow$ minimization finds best value m for given values F and D

Optimization for m

- For large D, formula for F_d can be approximated by exponential distribution:

$$F_d = (1 - e^{-\frac{mD}{F}})^m$$

- From this, the optimal value for m is obtained by derivation and transformation as:

$$m_{opt} = \frac{F * ln2}{D}$$

- Insert into the formula to calculate the probability of error:

$$F_d = (\frac{1}{2})^{\frac{F * ln2}{D}}$$

- Use of $m_{opt} \Rightarrow$ number of ones is 50%

Practical application of the probability of error

- Storage space calculation:
  - word consists of c_w characters on average
  - character requires 8 bit
  - let Y be the total number of words in all documents
  - storage space requirement for documents: $S_D = Y * c_w * 8bit$
  - storage space requirement for signatures: $S_S = Y * \frac{F}{D} bit$
  - we put both values in a ratio α: $S_S = \alpha * S_D$
  - thus: $\frac{F}{D} = \alpha * c_w * 8$

Practical application of the probability of error

- Example $\alpha = 0.1$ and $c_w = 10$
  - $\frac{F}{D} = 8$, for instance $F = 80$ and $D = 10$
  - $F_d = 2.14\%$
- Example $\alpha = 0.2$ and $c_w = 10$
  - $\frac{F}{D} = 16$, for instance $F = 80$ and $D = 5$
  - $F_d = 0.0459\%$

Storage structures for signature files

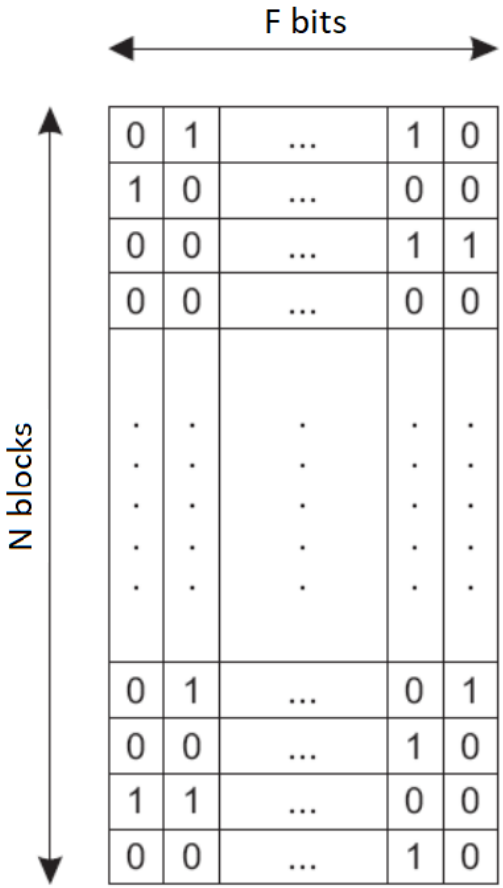
- Three alternatives:
  - sequential signature files
  - bit slice organization
  - S-tree

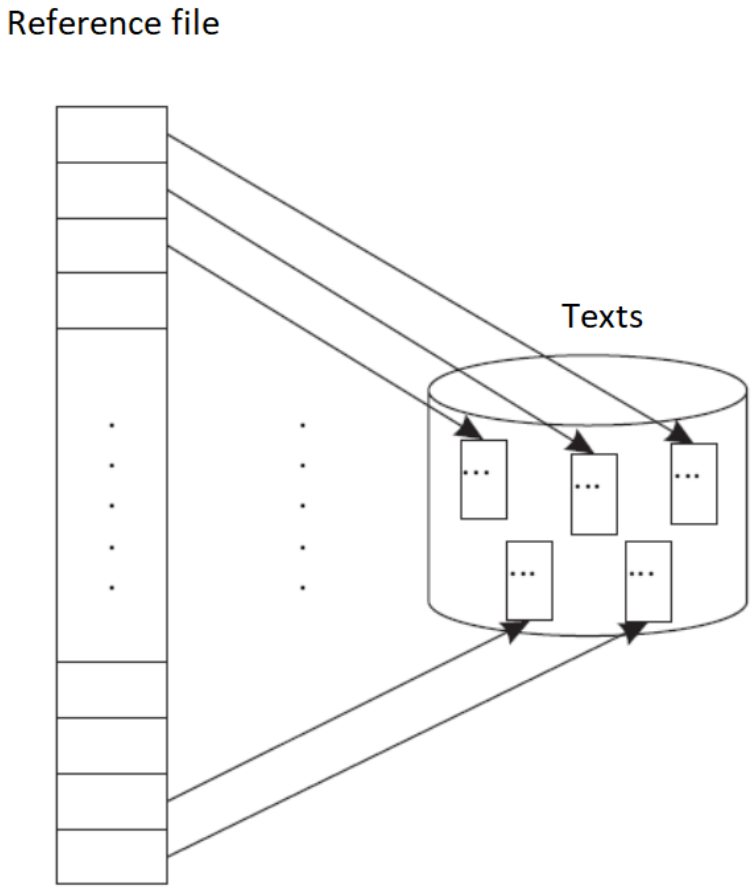
Sequential signature files

- Signature file for N blocks is a bit matrix with F columns and N rows
- Most simple storage: sequential storage of the individual rows of the matrix
  $\Rightarrow$ SSF (Sequential Signature File)
- After each signature reference to document
- Alternative: all references in separate file, because only some references have to be read

- Search effort grows linearly with document collection size
- Files may fit into main memory

Sequential signature files

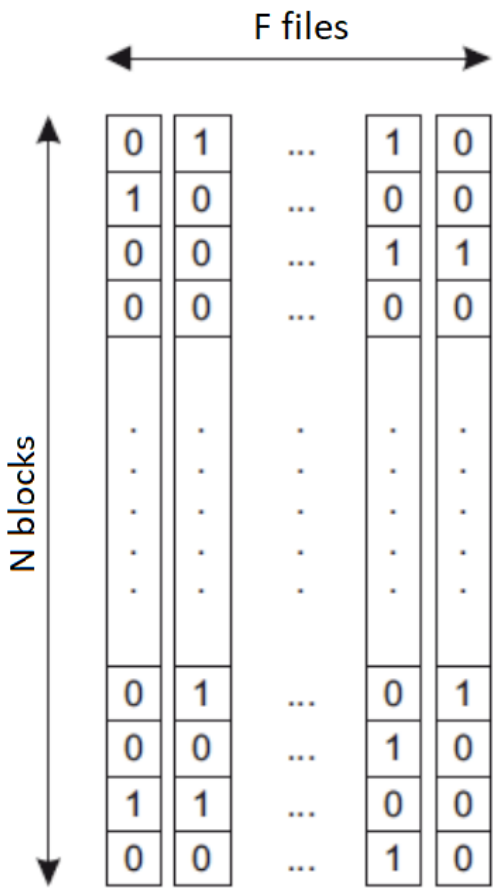


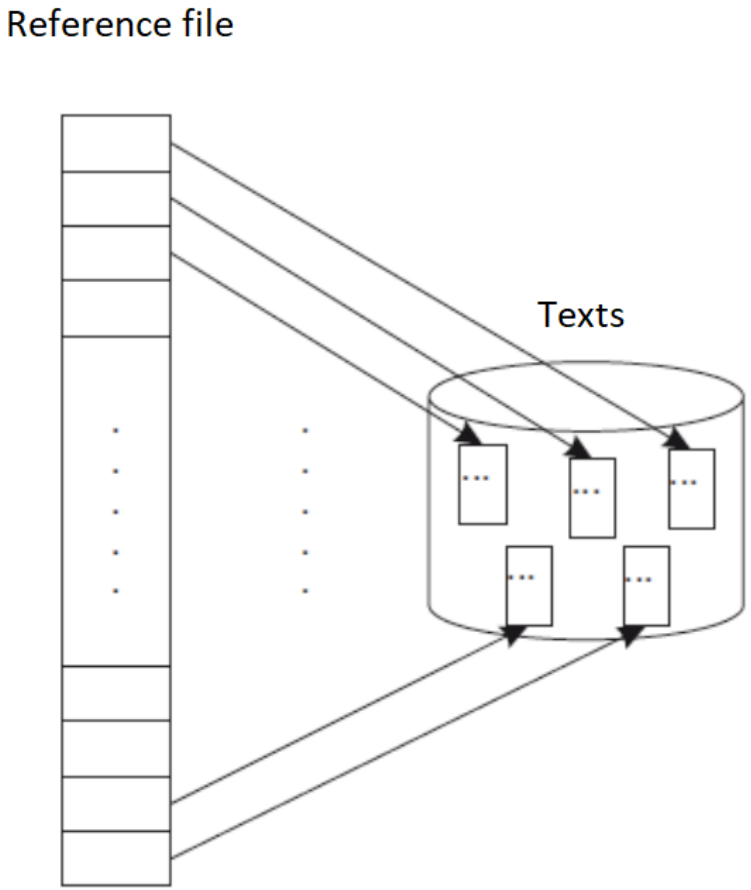


Bit slice organization

- SSF: block signatures are read completely
- Actually only columns needed where query signature bits are set
- Idea BSSF (Bit-Sliced Signature Files):
  - each column in separate file $\Rightarrow$ F files (bit slice)
  - per query (one search word) m bit slices must be read

Bit slice organization: structure





Bit slice organization: example

Query	Query signature
text search	010 010 101 100

Word	Signature
text	010 010 001 000
search	010 000 100 100
method	100 100 000 001
full	010 110 000 000
knowledge	000 111 000 000
retrieval	000 000 111 000
lexicon	100 010 010 000

Bit slice number	2	5	7	9	10
text search method	1	1	1	1	1
search knowledge retrieval	1	1	1	1	1
full text search	1	1	1	1	1
lexicon knowledge retrieval	0	1	1	1	0

Block is false alarm

Block no candidate

Bit slice organization: evaluation

- Advantage over SSF: not all bits have to be checked
- Disadvantage: update operations require F disk accesses

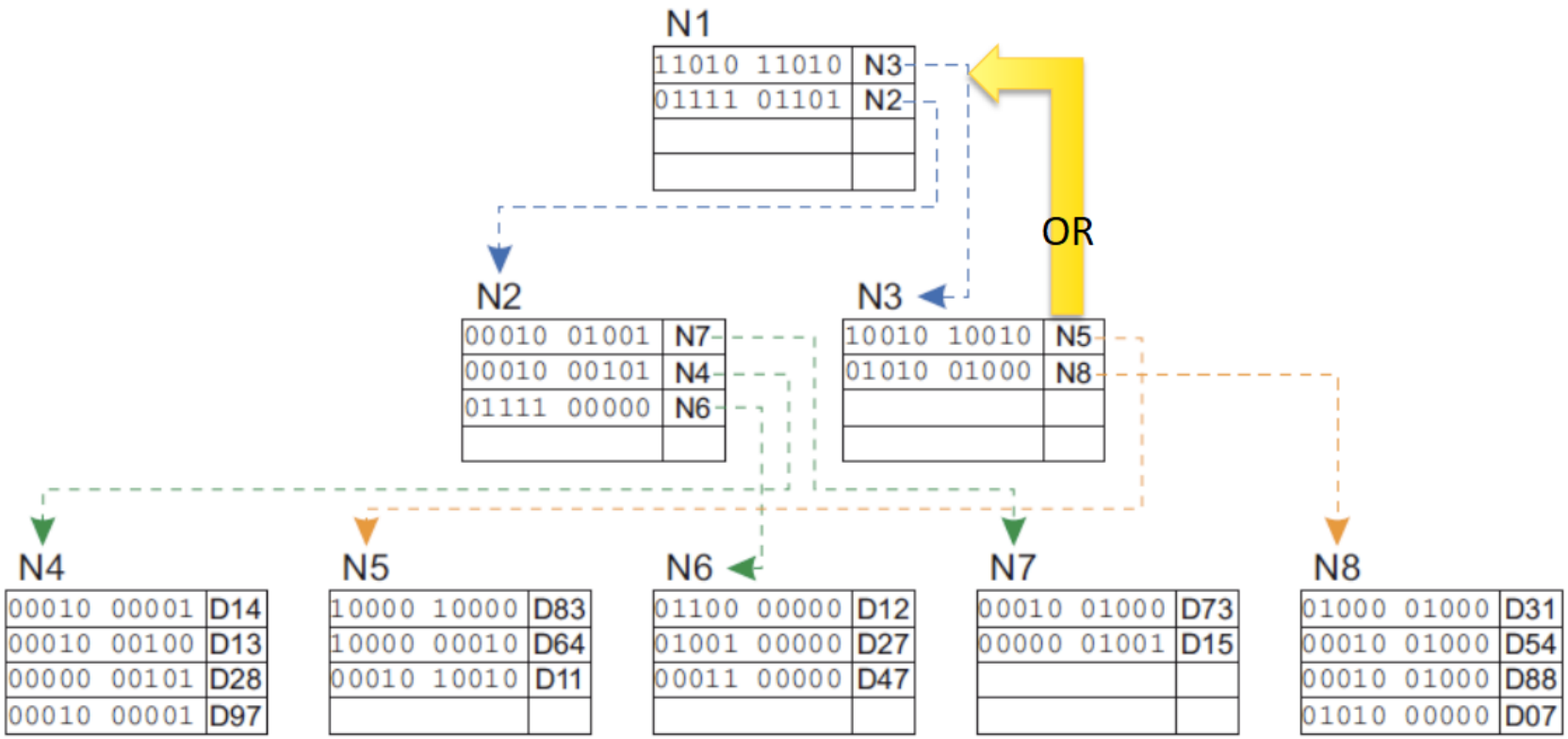
S-tree

- Horizontal division of the $N \times F$ bit matrix
- Similar to B-tree
- Structure:
  - each node has a maximum of k entries
  - entry: signature and address
  - inner nodes: address refers to child in tree
  - leaf: address refers to document/block

S-Tree: overlay

- Signature of an inner node is OR combination of child signatures
- Problem: nodes near the root can contain many 'ones'

S-tree: structure ($k = 4$)



Search in S-tree

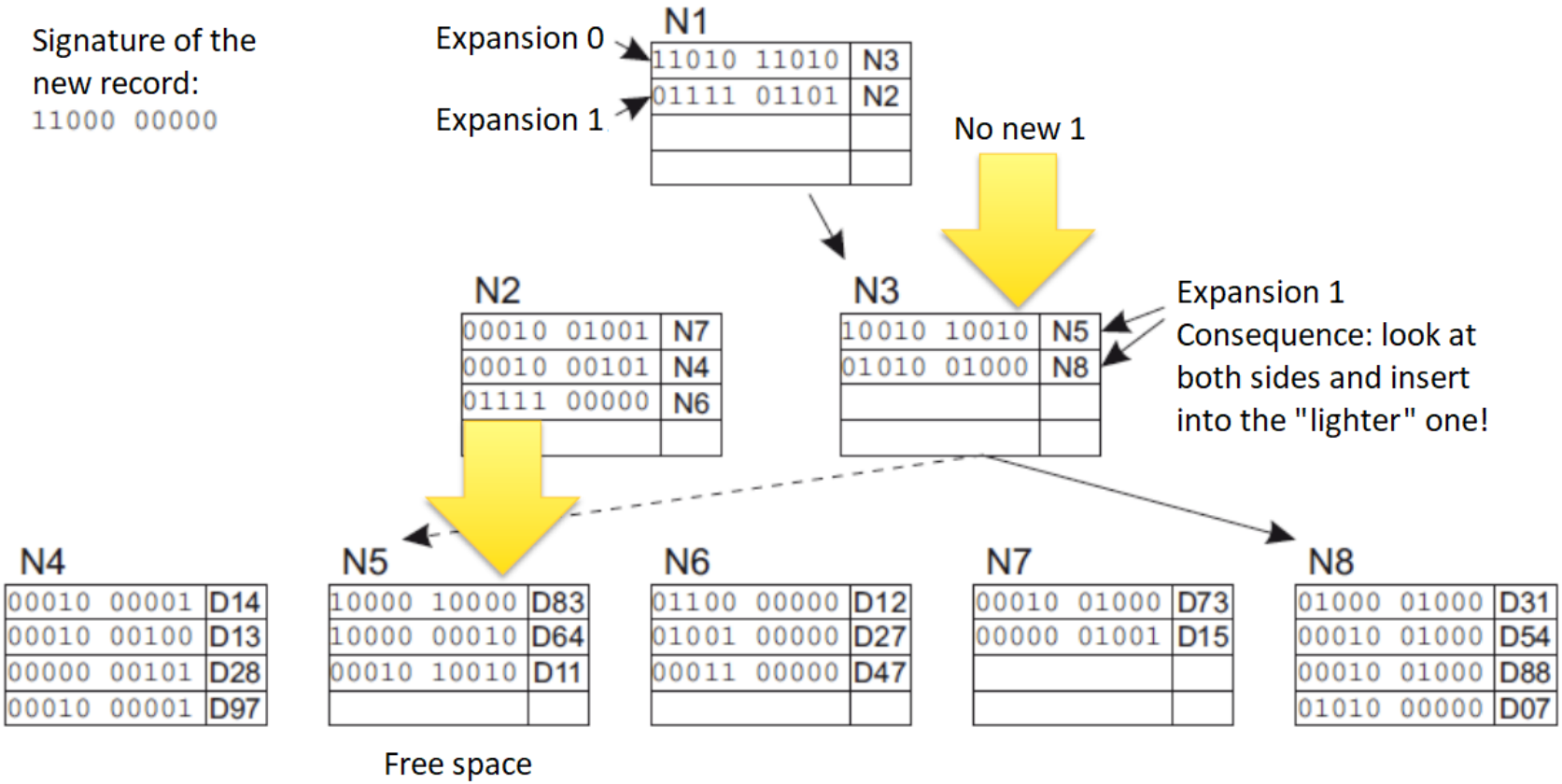
- Start at the root
- Traverse all children for which AND match with query signature successful
- Multiple search paths possible

- Example: query signature 01000 00000 generates traverse of N1, N2, N3, N6, N8, D12, D27, D31, D07

Insertion into S-Tree

- Navigate to the leaf via nodes that would need minimal expansion
- Overflow:
  - splitting the leaf
  - generate two "seed signatures", which should be maximally different (effort $O(k^2)$)
  - fast heuristics for finding seed signatures:
    search for the signatures whose bits are furthest to the left or right
  - then alternately assign most similar signature to each seed signature

Insertion into S-Tree



S-Tree: conclusion

- Advantage: search effort grows only sublinear to the size of the document collection
- Disadvantage: nodes near the root often have a lot of set bits
  $\Rightarrow$ only few nodes can be omitted from search

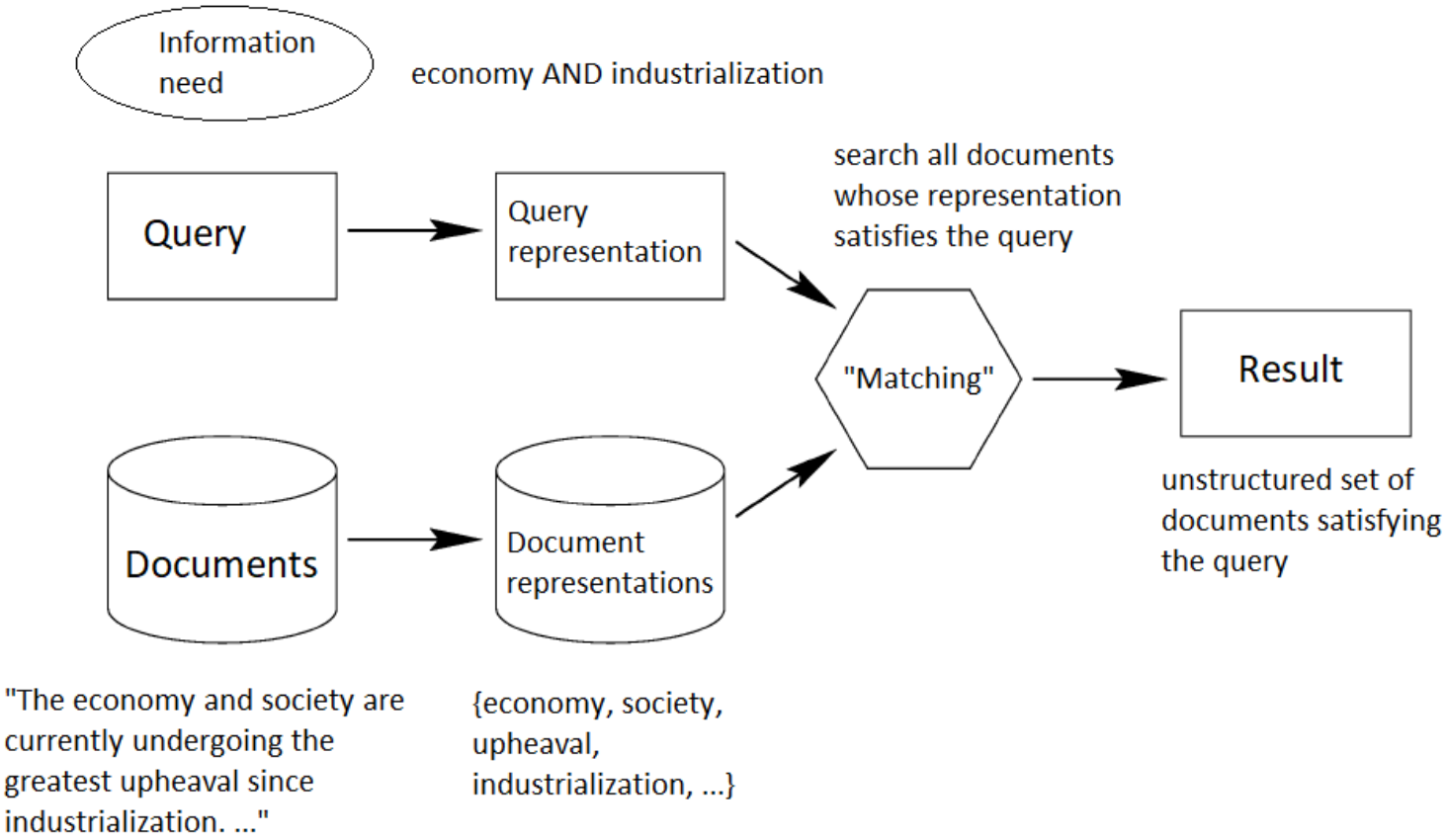
Comparison of signature storage structures

Profile indicates importance of operations

Structure	Profile			
	Search	Insert	Delete	Storage
Sequential	rare	dominant	rare	important
Bit slices	dominant	rare	rare	less important
S-tree	(little)	much	much	less important

4.3 Boolean Retrieval

- Boolean expressions over search words



Document representation

- Document as word set (possibly after stop word elimination and/or stem form reduction)
- Variants:
  - multiset instead of set
    $\Rightarrow$ storage of the frequency of occurrence
  - set of (word, occurrence location) pairs
  - location of occurrence:
    - for example in the title, heading or text
    - alternatively word position in the text (sentence number, word number)
      $\Rightarrow$ useful for NEAR queries

Query representation

- Information need is formulated as a Boolean expression
- Words in a word set are implicitly combined with OR or AND
- Examples:
  - ‘eagle bear’ $\Rightarrow$ ‘eagle’ AND ‘bear’
  - ‘eagle’ AND (‘bear’ OR ‘lion’)
  - ‘eagle’ AND NOT ‘lion’
  - BUT operator instead of AND NOT

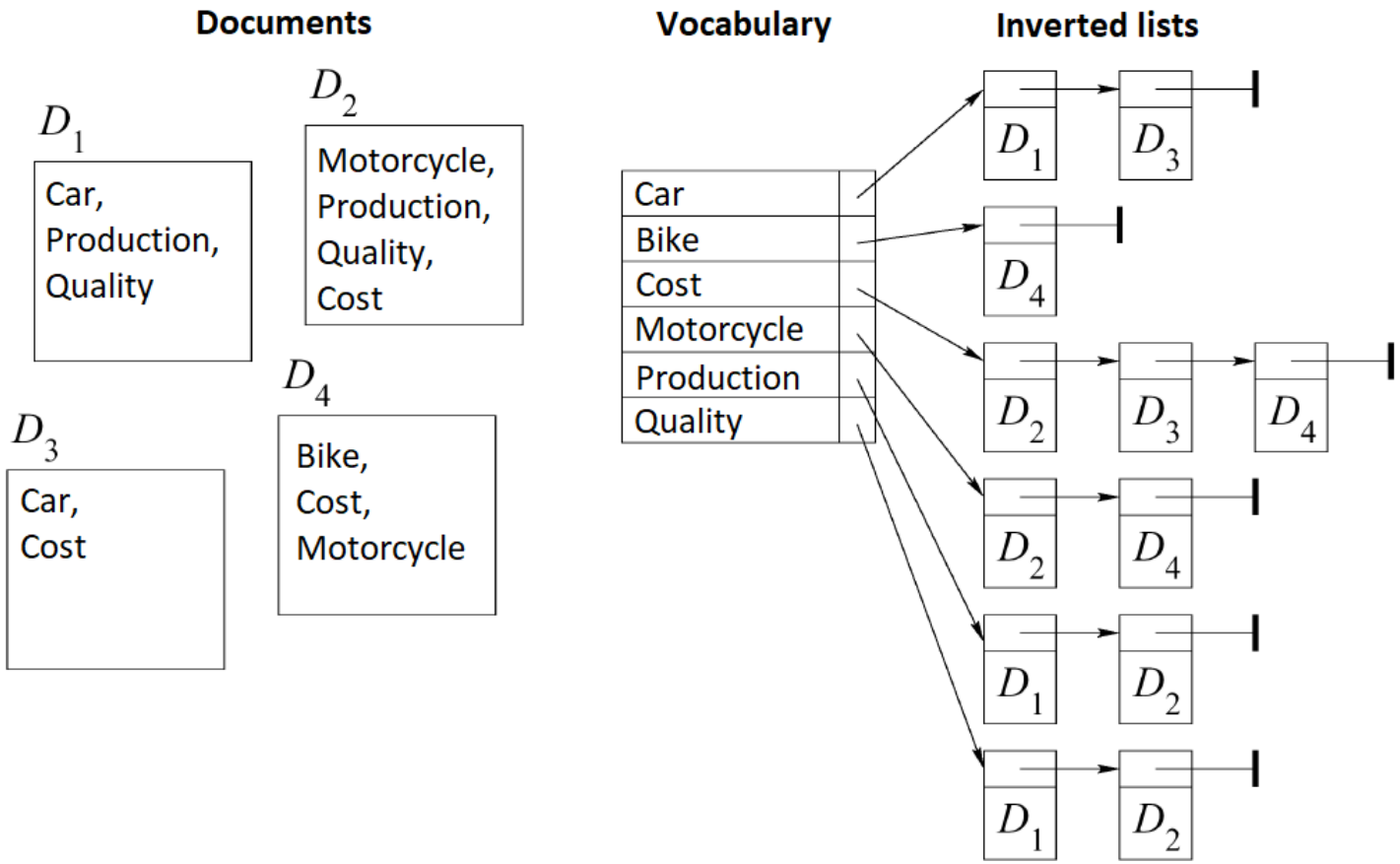
Implementation

Variants:

- Pattern matching
  very inefficient for large document collections
- Signatures
  well suited for AND combinations (but only blockwise)
- Inverted list
  most versatile option

Implementation with inverted lists

Simple (one search word) query



Combined queries

- AND combination
  - intersection of the lists (using sorting for efficiency)
- OR combination
  - union
- Unary negation $q = NOT\ q_2$
  - usually very large result, binary variant is better
- Binary negation $q = q_1\ AND\ NOT\ q_2$
  - complementary set

Other operations

- $t_i NEAR[k] t_j$
  - term t_i must not be further than k words away from term t_j
  - requires occurrence position in inverted list
  - scan through lists in parallel and test possible pairs
- IN_TITLE
  - word appears in the title

Boolean Retrieval: summary

Problems:

- In simplest form no reduction to basic and stem form
- No weighting of words by location or frequency
- No decomposition of compound terms
- Hardly predictable size of the result set
- No ranking of the results
- User must be proficient in Boolean logic

4.4 Coordination Level Match

Motivation:

- Example query of Boolean retrieval:
  'House AND Garden AND Italy'
- Problem: documents with 2 word matches do not appear, just like documents with 0 or 1 match
- Goal: if no document with 3 word matches
  $\Rightarrow$ documents with 2 matches as result

Coordination level match

Query consists of

- Terms that should occur and
- Terms that should not occur

Example: 'House, Garden, NOT France'

Retrieval function sums up values (ranking):

- Desired term in the document: +1
- Undesired term in the document: -1

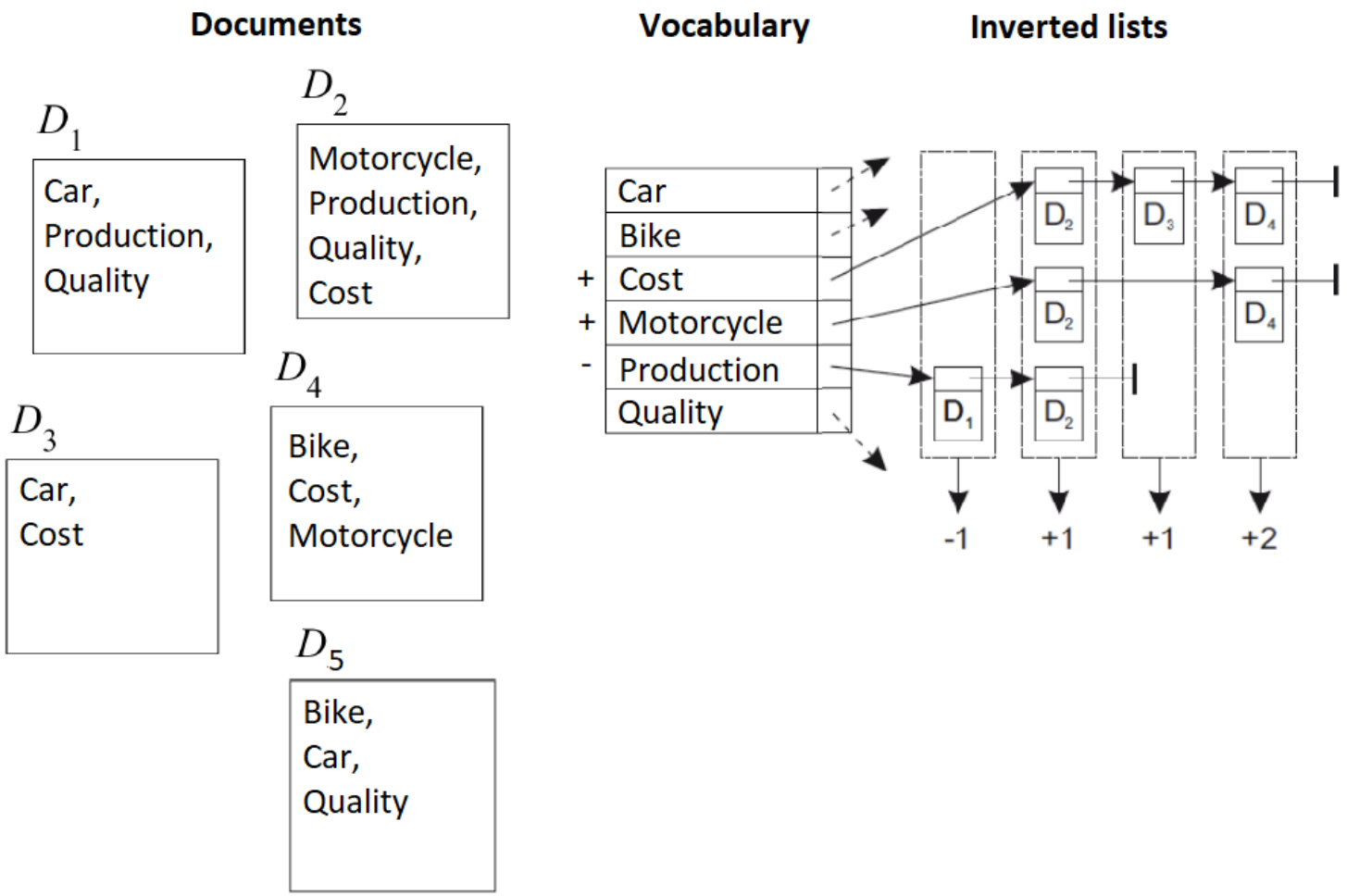
Example document: 'Garden in France' $\Rightarrow$ 0

Implementation

- Using inverted lists
- Utilization of sorted lists
- Parallel scanning through the lists and addition of the values
- Ignore the documents with values less than or equal to 0
- Display of result documents sorted by value
- Effort: $O(n * m)$ with n for number of documents and m for number of lists to be scanned

Inverted lists

Query: "Cost, Motorcycle, NOT Production"



4.5 Extended Boolean

- Boolean retrieval leads to low user satisfaction
- 'Softening' of the strict set semantics by partial matching
- Weighting of the occurrence of individual terms and the query terms

Extended Boolean retrieval

- E.g. Waller/Kraft (1979)
  - Parameterized junctor

$$t_1 \wedge / \vee \ldots \wedge / \vee t_n$$
$$(1-\gamma) * min(w_{t_1}, \ldots, w_{t_n}) + \gamma * max(w_{t_1}, \ldots, w_{t_n})$$

  - Conjunction characteristic: $0 \leq \gamma \leq 0.5$
  - Disjunction characteristic: $0.5 \leq \gamma \leq 1$

Extended Boolean with weights

- $x_i \in [0,1]$ is i-th similarity value with respect to search term t_i and document d_j
- E.g. p-norm model (Salton et al.; 1983)

$$q_{or} = t_1 \vee^p \ldots \vee^p t_n \qquad q_{and} = t_1 \wedge^p \ldots \wedge^p t_n$$

$$sim(q_{or}, d_j) = \left(\frac{\sum_{i=1}^{n} (x_i)^p}{n}\right)^{\frac{1}{p}}$$

$$sim(q_{and}, d_j) = 1 - \left(\frac{\sum_{i=1}^{n} (1-x_i)^p}{n}\right)^{\frac{1}{p}}$$

$$1 \leq p \leq \infty$$

Properties p-norm

- $p = 1$

$$sim(q_{or}, d_j) = sim(q_{and}, d_j) = (\frac{\sum_{i=1}^{n} x_i}{n})$$

- $p = \infty$

$$sim(q_{or}, d_j) = max(x_i)$$
$$sim(q_{and}, d_j) = min(x_i)$$

4.6 Fuzzy Set Model

- Idea: application of fuzzy set theory (Zadeh) as retrieval model
- Boolean retrieval: term contained or not in document
- Problem: Boolean values are too sharp
- Example: document with term 'SQL' is semantically related to term 'database'
- Approach: term-to-document association as membership value from [0, 1], where 1 or 0 means maximum or minimum membership, respectively

Fuzzy set

A fuzzy set S over a population G is described by function $\mu_S : G \rightarrow [0, 1]$, which gives the degree of membership to S for each element from G

Combination (operations proposed by Zadeh):

- Negation: $\mu_{\neg S}(u) = 1 - \mu_S(u)$
- Disjunction: $\mu_{A \cup B}(u) = max(\mu_A(u), \mu_B(u))$
- Conjunction: $\mu_{A \cap B}(u) = min(\mu_A(u), \mu_B(u))$

Representation of the query/documents

- Query analogous to Boolean retrieval
- Document D_j : $\mu_i(D_j)$ for all terms t_i of the vocabulary
- Retrieval function: Zadeh operations

Representation of the query/documents: example

- Vocabulary: t_1='House', t_2='Garden', t_3='Italy', t_4='France'
- Query: 'House AND (Italy OR France) AND NOT Garden'
- Retrieval function:
  $min(min(\mu_1(D_j), max(\mu_3(D_j), \mu_4(D_j))), 1 - \mu_2(D_j))$

Term	**Documents** D_j				
	D_1	D_2	D_3	D_4	D_5
House	0,5	0,0	0,7	0,3	0,9
Garden	0,9	1,0	0,1	0,0	0,0
Italy	1,0	0,0	0,0	1,0	0,0
France	1,0	1,0	0,4	1,0	0,5
Query	0,1	0,0	0,4	0,3	0,5

Determination of the membership values

Approach of Ogawa, Morita, Kobayashi [OMK91]

- Calculation via term-to-term matrix $[c_{i,l}]$
- $c_{i,l}$ is correlation between terms t_i and t_l

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

- n_l is number of documents that contain t_l
- $n_{i,l}$ is number of documents containing t_i and t_l
- $c_{i,l} = 1$ holds for $i = l$

Determination of the membership values

Example

- Term ‘house’ in 7 documents
- Term ‘roof’ in 6 documents
- Both together in 5 documents
- $n_i = 7, n_l = 6, n_{i,l} = 5 \Rightarrow c_{i,l} = \frac{5}{8}$

Determination of the membership values

Calculation of term-to-document membership from term-to-term matrix

$$\mu_i(D_j) = 1 - \prod_{t_l \in D_j} (1 - c_{i,l})$$

- Replace $1 - x$ by $\neg x$ and $x * y$ by $x \wedge y$ and apply De Morgan's laws and double negation rule

  $$\Rightarrow \mu_i(D_j) = \bigvee_{t_l \in D_j} c_{i,l}$$

- Term-to-term values of the document are factually disjunctively combined

Implementation

Usage of inverted lists

- List for term t_i contains documents D_j with $\mu_i(D_j) > 0$
- Another problem: long lists
  shorter lists by using threshold value

Evaluation fuzzy set model

- Advantages
  - support of the vagueness of the language by term correlations
  - ranking
- Disadvantages
  - dominance of Zadeh operations *min/max*
  - complex Boolean query formulation
  - performance problems due to long lists
  - term-to-term matrix on dynamic document collection (better on representative data set)
  - alternative: term correlation from thesaurus

Dominance problem of the Zadeh operations

- Query: 'House AND Italy'

Term	D_1	D_2
House	0.3	0.9
Italy	0.3	0.2
Query	0.3	0.2

- System considers D_1 more relevant than D_2