Information Retrieval and Data Mining
Part 1 – Information Retrieval

Today's Question

1. Information Retrieval
2. Text Retrieval Models
3. Relevance Feedback
4. Inverted Files
5. Web Information Retrieval

Problem 2. Semantics (Interpretation)

- Query processing: use explicit structure of data for interpretation
  - Locating data (typically using first-order logical statements)

PGA Tour

ID	Name	Earning	Driving Dist
1	T. Woods	6496	296
2	S. Garcia	2319	285
3	E. Els	3180	279
4	D. Toms	2534	271
5	J. Daly	630	305
6	D. Duval	489	290

Pro = { p | Earning(p) > 6000 and Driving Dist(p) > 280}

- Data Mining: turn implicit structure of data into explicit
  - detecting (unexpected) patterns in structured data

PGA Tour

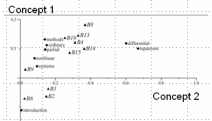
ID	Name	Earning	Driving Dist
1	T. Woods	6496	296
2	S. Garcia	2319	285
3	E. Els	3180	279
4	D. Toms	2534	271
5	J. Daly	630	305
6	D. Duval	489	290

Earning(p) > 1000 => Driving Dist(p) > 270
(confidence 75%, support 66%)

- Information retrieval: use implicit structure of data for interpretation
  - Locating relevant information

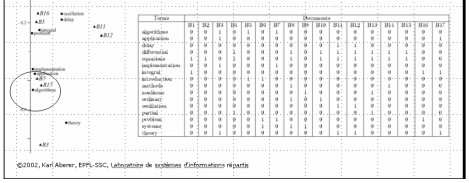
Example: Information Retrieval

Concept 1



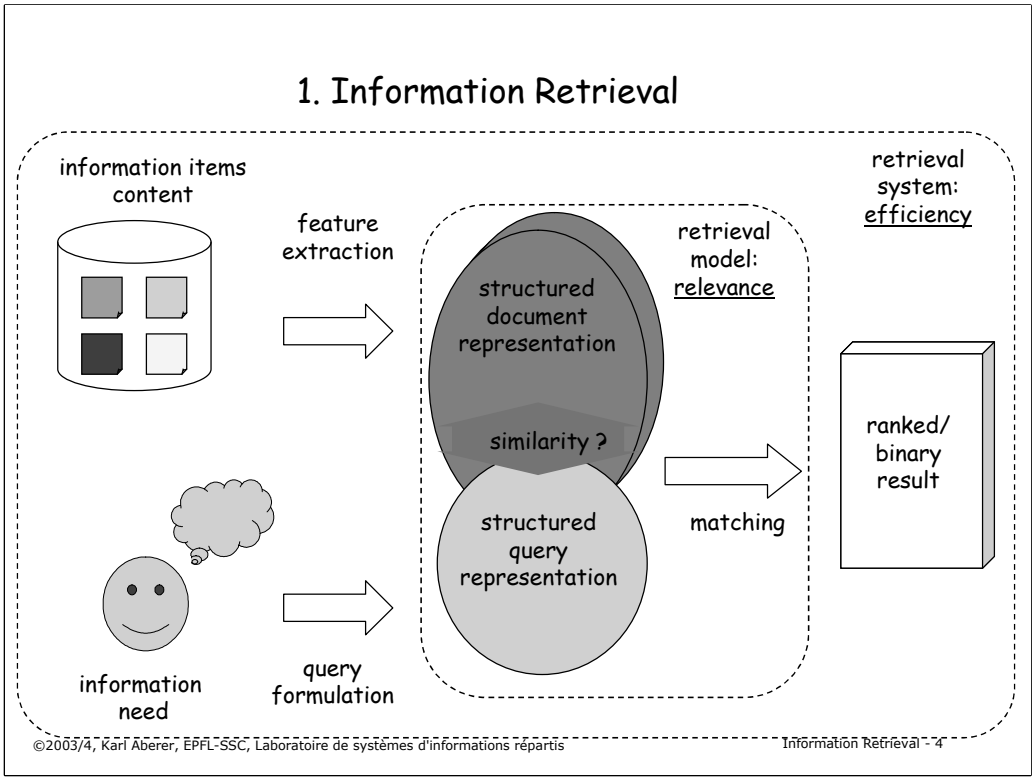
Concept 2

- Analysis of term-document matrix reveals large Eigen-values
- Interpreted as concepts = clusters = implicit structure
- Concepts are used to retrieve similar documents



What do you think ?

- How is a Web search engine working ?
  -
  -
  -
  -
  -
  -



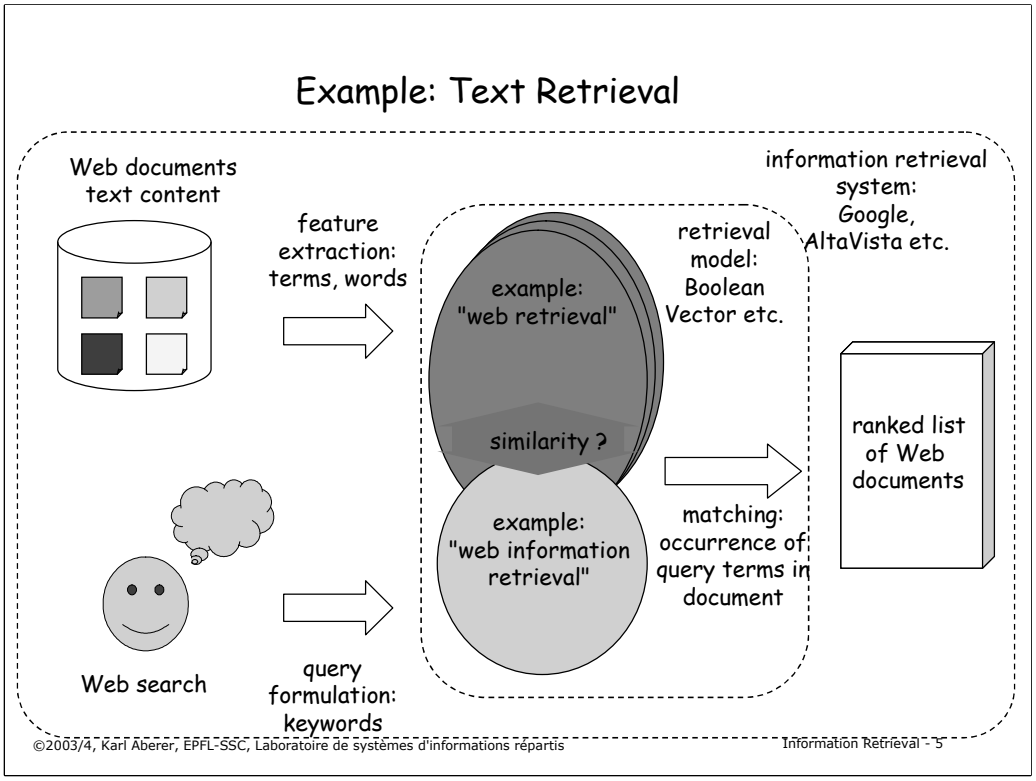
Therefore the information retrieval system has to deal with the following tasks:

•Generating structured representations of information items: this process is called **feature extraction** and can include simple tasks, such as extracting words from a text as well as complex methods, e.g. for image or video analysis.

•Generating structured representations of information needs: often this task is solved by providing users with a query language and leave the formulation of structured queries to them. This is the case for example for simple keyword based query languages, as used in Web search engines. Some information retrieval systems also support the user in the **query formulation,** e.g. through visual interfaces.

•Matching of information needs with information items: this is the algorithmic task of computing similarity of information items and information need and constitutes the heart of the **information retrieval model**. Similarity of the structured representations is used to model **relevance** of information for users. As a result a selection of relevant information items or a ranked result can be presented to the user.

Since information retrieval systems deal usually with large information collections and/or large user communities, the **efficiency** of an information retrieval system is crucial. This imposes fundamental constraints on the retrieval model. Retrieval models that would capture relevance very well, but



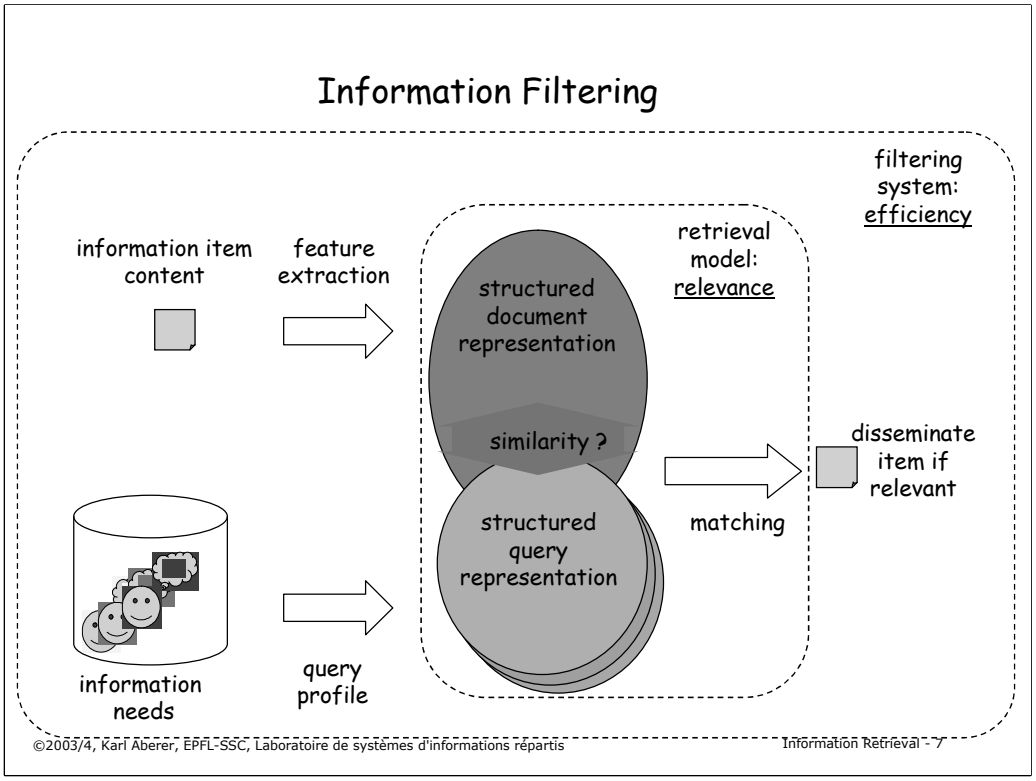
The currently most popular information retrieval systems are Web search engines. To a large degree, they are text retrieval system, since they exploit only the textual content of Web documents for retrieval. However, more recently Web search engines also start to exploit link information and even image information. The three tasks of a Web search engine for retrieval are:

1. extracting the textual features, which are the words or terms that occur in the documents. We assume that the web search engine has already collected the documents from the Web using a Web crawler.
2. support the formulation of textual queries. This is usually done by allowing the entry of keywords through Web forms.
3. computing the similarity of documents with the query and producing from that a ranked result. Here Web search engines use standard text retrieval methods, such as Boolean retrieval and vector space retrieval. We will introduce these methods in detail subsequently.

Retrieval Model

- Determines
  - the structure of the document representation
  - the structure of the query representation
  - the similarity matching function
- Relevance
  - determined by the similarity matching function
  - should reflect right topic, user needs, authority, recency
  - no objective measure
- Quality of a retrieval model depends on how well it matches user needs !
- Comparison to database querying
  - correct evaluation of a class of query language expressions
  - can be used to implement a retrieval model

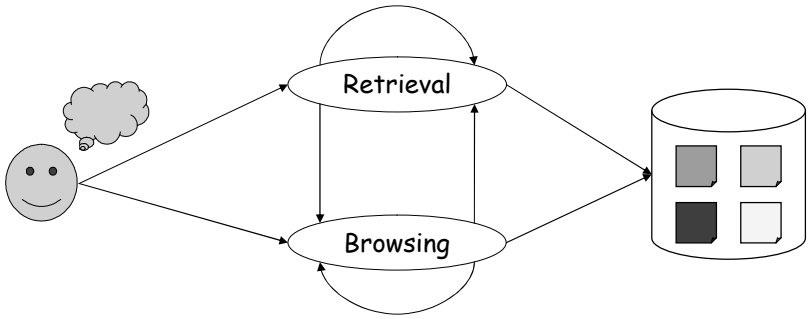
The heart of an information retrieval system is its retrieval model. The model is used to capture the meaning of documents and queries, and determine from that the relevance of documents with respect to queries. Although there exist a number of intuitive notions of what determines relevance one must keep clearly in mind that it is not an objective measure. The quality of a retrieval system can principally only be determined through the degree of satisfaction of its users. This is fundamentally different to database querying, where there exists a formally verifiable criterion for the task to be performed: whether a result set retrieved from a database matches the conditions specified in a query.



Similarly, as in a XML-based message filtering system the roles of documents and queries are inverted, also in an information retrieval system the roles of information items and information needs can be inverted, such that one obtains an information filtering system. Information filtering systems can be based on the same retrieval models as classical information retrieval systems for ad-hoc query access.

Information Retrieval and Browsing

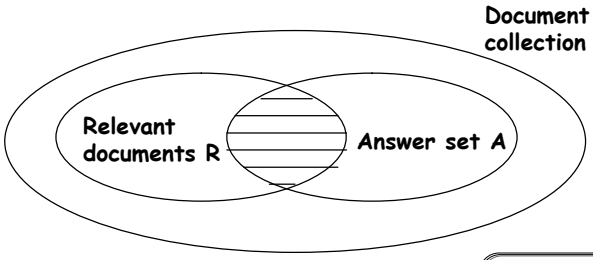
- Retrieval
  - Produce a ranked result from a user request
  - Interpretation of the information by the system
- Browsing
  - Let the user navigate in the information set
  - Interpretation of the information by the human



Information retrieval is usually closely connected to the task of browsing. Browsing is the explorative access of users to large document collections. By browsing a user implicitly specifies his/her information needs by the selection of documents. This feedback can be used by an information retrieval system in order to improve its retrieval model and thus the retrieval result. One example of such an approach we will see with relevance feedback. On the other hand, results returned by information retrieval systems are usually large, and therefore browsing is used by users in order to explore the results. Both activities, retrieval and browsing thus can be combined into an iterative process.

Evaluating Information Retrieval

- Recall is the fraction of relevant documents retrieved from the set of total relevant documents collection-wide
- Precision is the fraction of relevant documents retrieved from the total number retrieved (answer set)
- Test collections, where the relevant documents are identified manually are used to determine the quality of an IR system (e.g. TREC)



Recall = $|A| \cap |R| / |R|$

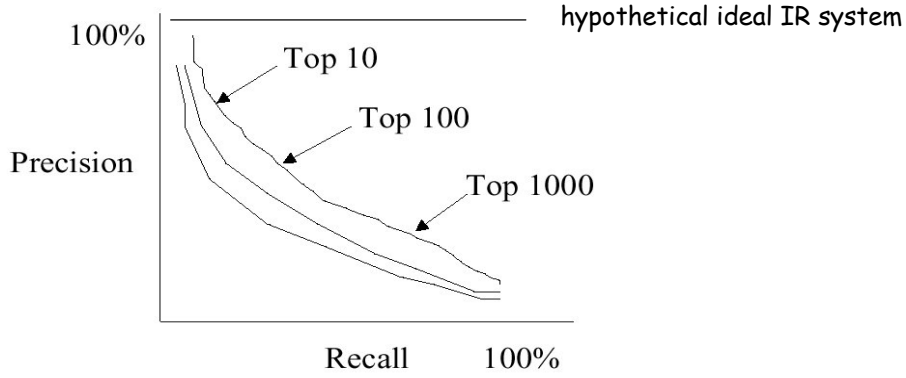
Precision = $|A| \cap |R| / |A|$

Since there exists no objective criterion whether an information retrieval query is correctly answered, other means for evaluating the quality of an information retrieval system are required. The approach is to compare the performance of a specific system to human performance in retrieval. For that purpose test collections of documents, such as TREC, are created and for selected queries human experts select the relevant documents. Note that this approach assumes that humans have an agreed-upon, objective notion of relevance, an assumption that can be easily challenged of course. Then the results of IR systems are compared to the expected result in two ways:

1. Recall measures how large a fraction of the expected results is actually found.
2. Precision measures how many of the results returned are actually relevant.

Precision/Recall Tradeoff

- An IR system ranks documents by a similarity coefficient, allowing the user to trade off between precision and recall by choosing the cutoff level



One of the two measures of recall and precision can always be optimized. Recall can be optimized by simply returning the whole document collection, whereas precision can be optimized by returning only very few results. Important is the trade-off: the higher the precision for a specific recall, the better the information retrieval system. A hypothetical, optimal information retrieval system would return results with 100% percent precision always. If a system ranks the results according to relevance the user can control the relation between recall and precision by selecting a threshold of how many results he/she inspects.

Summary

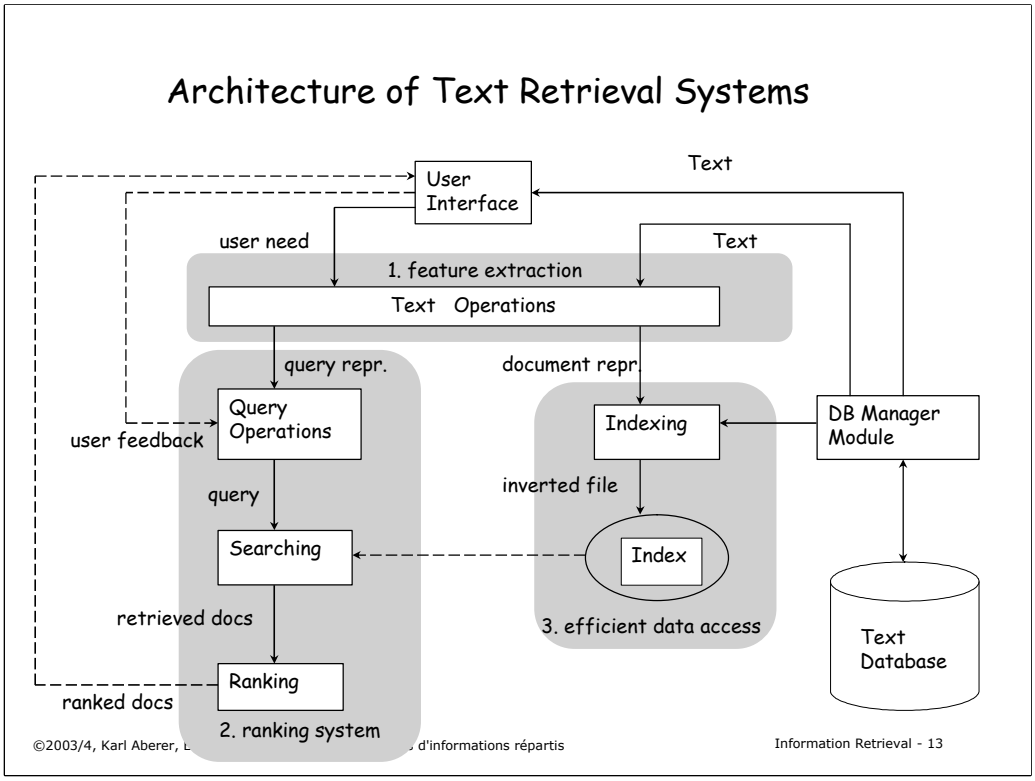
- What is the difference between data search and information retrieval ?
- What are the main processing steps in information retrieval ?
- How do browsing and filtering relate to information retrieval ?
- How is an information retrieval system evaluated ?

2. Text-based Information Retrieval

- Most of the information needs and content are expressed in natural language
  - Library and document management systems
  - Web (Search Engines)
- Basic approach: use the words that occur in a text as features for the interpretation of the content
  - This is called the "full text" retrieval approach
  - Ignore grammar, meaning etc.
  - Simplification that has proven successful
  - Document structure may be taken into account additionally (e.g. PageRank/Google)

Classical information retrieval was concerned over the last 20 years with the problem of retrieving information from large bodies of documents with mostly textual content, as they where typically found in library and document management systems. The problems addressed were classification and categorization of documents, systems and languages for retrieval, user interfaces and visualization. The area was perceived as being one of narrow interest for highly specialized applications and users. The advent of the WWW changed this perception completely, as the web is a universal repository of documents with universal access.

Since nowadays most of the information content is still available in textual form, text is an important basis for information retrieval. Natural language text carries a lot of meaning, which still cannot fully be captured computationally. Therefore information retrieval systems are based on strongly simplified models of text, ignoring most of the grammatical structure of text and reducing texts essentially to the terms they contain. This approach is called full text retrieval and is a simplification that has proven to be very successful. Nowadays this approach is gradually extended by taking into account other features of documents, such as the document or link structure.

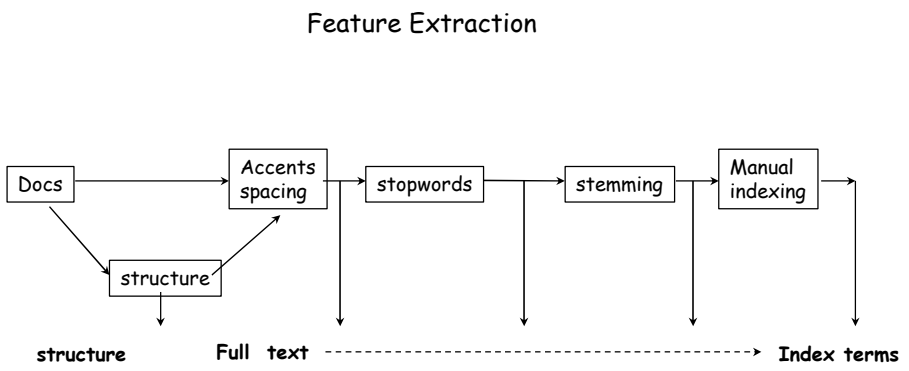


This figure illustrates the basic architecture with the different functional components of a text retrieval system. We can distinguish three main groups of components:

1. the feature extraction component: it performs text processing to turn queries and text documents into a keyword-based representation
2. the ranking system: it implements the retrieval model. In a first step user queries are potentially modified (in particular if user relevance feedback is used), then the documents required for producing the result are retrieved from the database and finally the similarity values are computed according to the retrieval model in order to compute the ranked result.
3. the data access system: it supports the ranking system by efficiently retrieving documents containing specific keywords from large document collections. The standard technique to implement this component is called "inverted files".

In addition we recognize two components to interface the system to the user on the one hand, and to the data collection on the other hand.

Pre-Processing Text for Text Retrieval



In full text retrieval each document is represented by a set of representative keywords or index terms. An index term is a document word useful for capturing the document's main topics. Often, index terms are only nouns because nouns carry meaning by themselves, whereas verbs express relationships between words. These relationships are more difficult to extract.

When using words as text features normally a stepwise processing approach is taken: in a first step the document structure, e.g. from XML, is extracted and if required stored for further processing. The remaining text is stripped of special characters, producing the full text of the document. Then very frequent words which are not useful for retrieval, so-called "stopwords", are eliminated (e.g. "a", "and" etc.). As the same word can occur in natural language in different forms, usually stemming is used: Stemming eliminates grammatical variations of the same word by redusing it to a word root, e.g. all the words connecting, connection, connections would be reduced to the same "stem" connect. This step can be followed by a manual intervention where humans can select or add index terms based on their understanding of the semantics of the document. The result of the process is a set of index terms which represents the

Text Retrieval - Basic Concepts and Notations

Document d:	expresses ideas about some topic in a natural language
Query q:	expresses an information need for documents pertaining to some topic
Index term:	a semantic unit, a word, short phrase, or potentially root of a word
Database DB:	collection of n documents d_j ä DB, j=1,…,n
Vocabulary T:	collection of m index terms k_i ä T, i=1,…,m

A document is represented by a set of index terms k_i

The importance of an index term k_i for the meaning of a document is represented by a weight w_{ij} ä [0,1]; we write $d_j = (w_{1j}, …, w_{mj})$

The IR system assigns a similarity coefficient $sim(q, d_j)$ as an estimate for the relevance of a document d_j ä DB for a query q.

We introduce the precise terminology we will use in the following for text retrieval systems. Note that the way of how specific weights are assigned to an index term with respect to a document and of how similarity coefficients are computed are part of the definition of the text retrieval model.

Example: Documents

B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay
B12 Oscillation Theory of Delay Differential Equations
B13 Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14 Sinc Methods for Quadrature and Differential Equations
B15 Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16 The Boundary Integral Approach to Static and Dynamic Contact Problems
B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

This is an example of a (simple) document collection.

Term-Document Matrix

Vocabulary (contains only terms that occur multiple times, no stop words)

Terms	Documents																
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17
algorithms	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
application	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
delay	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
differential	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
equations	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
implementation	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
integral	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
introduction	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
methods	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
nonlinear	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
ordinary	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
oscillation	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
partial	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
problem	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
systems	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
theory	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

all weights are set to 1 (equal importance)

In text retrieval we represent the relationship between the index terms and the documents in a term-document matrix. In this example only a selected vocabulary is used for retrieval, consisting of all index terms that occur in more than one document and only weights 1 are assigned, if the term occurs in the document.

Boolean Retrieval

- Users specify which terms should be present in the documents
  - Simple, based on set-theory, precise meaning
  - Frequently used in old library systems
- Example query
  - "application" AND "theory"
  - answer: B3, B17

Retrieval Language

expr ::= term | (expr) | not expr | expr AND expr | expr OR expr

Weights for index terms appearing in documents

$w_{ij} = 1$ if $k_i \hat{I} d_j$ and 0 otherwise

Early information retrieval systems (as well as many systems today on the Web, such as amazon) use the Boolean retrieval model. This model is actually more similar to database querying, as requests are specified as first order (Boolean) expressions. Term weights are set to 1 when a term occurs in a document, just as in the term-document matrix on the previous slide.

"Similarity" Computation in Boolean Retrieval

- Step 1: Determine the disjunctive normal form of the query
  - A disjunction of conjunctions
  - Using distributivity and Morgans laws
  - e.g. NOT (s AND t) º NOT s OR NOT t
- Step 2: For each conjunctive term check whether its weight vector matches the document weight vector
  - Weight vector of conjunctive term ct = t_1 AND … AND t_k

    vec(ct)=(w_1,…,w_m) : w_j = 1 if k_j Î {t_1,…,t_k} else 0
- Step 3: If one weight vector of a conjunctive term matches the document weight vector then the document is relevant
  - ct matches d_j : w_i=1 ® w_{ij}=1

Computing the similarity of a document with a query reduces in Boolean retrieval to the problem of checking whether the term occurrences in the document satisfy the Boolean condition specified by the query. In order to this in a systematic manner a Boolean query is first normalized into disjunctive normal form. In this equivalent representation checking whether a document matches the query reduces to the problem of checking whether the document vector, i.e. the column of the term-document matrix corresponding to the document, matches one of the conjunctive terms of the query. A match is established if the document vector contains all the terms of the query vector.

Example

- Index terms {application, algorithm, theory}
- Query "application" AND ("algorithm" OR NOT "theory")
- Disjunctive normal form of query

  ("application" AND "algorithm" AND "theory") OR
  ("application" AND "algorithm" AND NOT "theory") OR
  ("application" AND NOT "algorithm" AND NOT "theory")

- Query weight vectors $q=\{(1,1,1), (1,1,0), (1,0,0)\}$
- Documents

d_1={algorithm, theory, application}	(1,1,1)
d_2={algorithm, theory}	(1,0,1)
d_3={application, algorithm}	(1,1,0)

- Result $sim(d_1, q) = sim(d_3, q) = 1, sim(d_2, q) = 0$

Vector Space Retrieval

- Limitations of Boolean Retrieval
  - No ranking: problems with handling large result sets
  - Queries are difficult to formulate
  - No tolerance for errors
- Key Idea of Vector Space Retrieval
  - represent both the document and the query by a weight vector in the m-dimensional keyword space assigning non-binary weights
  - determine their distance in the m-dimensional keyword space
- Properties
  - Ranking of documents according to similarity value
  - Documents can be retrieved even if they don't contain a keyword in the query
- Todays standard text retrieval technique
  - Web Search Engines
  - The vector model is usually as good as the known ranking alternatives
  - It is simple and fast to compute

The main limitation of the Boolean retrieval model is its incapability to rank the result and to match documents that do not contain all the keywords of the query. In addition, more complex requests become very difficult to formulate. The vector space retrieval model addresses these issues, by supporting non-binary weights, i.e. real numbers in [0,1], both for documents and queries, and producing continuous similarity measures in [0,1]. The similarity measure is derived from the geometrical relationship of vectors in the t-dimensional space of document/query vectors. The vector space retrieval model is the standard retrieval technique used both on the Web and for classical text retrieval.

Similarity Computation in Vector Space Retrieval

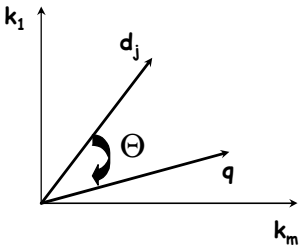
$$\vec{d}_j = \left(w_{1j}, w_{2j}, ..., w_{mj}\right),\ w_{ij} > 0 \quad if\ k_i \in d_j$$

$$\vec{q} = \left(w_{1q}, w_{2q}, ..., w_{mq}\right),\ w_{iq} \geq 0$$

$$sim(\vec{q}, \vec{d}_j) = \cos(q) = \frac{\vec{d}_j \bullet \vec{q}}{\left|\vec{d}_j\right|\left|\vec{q}\right|} = \frac{\sum_{i=1}^{m} w_{ij} w_{iq}}{\left|\vec{d}_j\right|\left|\vec{q}\right|}$$

$$|v| = \sqrt{\sum_{i=1}^{m} v_i^2}$$

Since $w_{ij} > 0$ and $w_{iq} \geq 0$, $0 \leq sim(q, d_j) \leq 1$



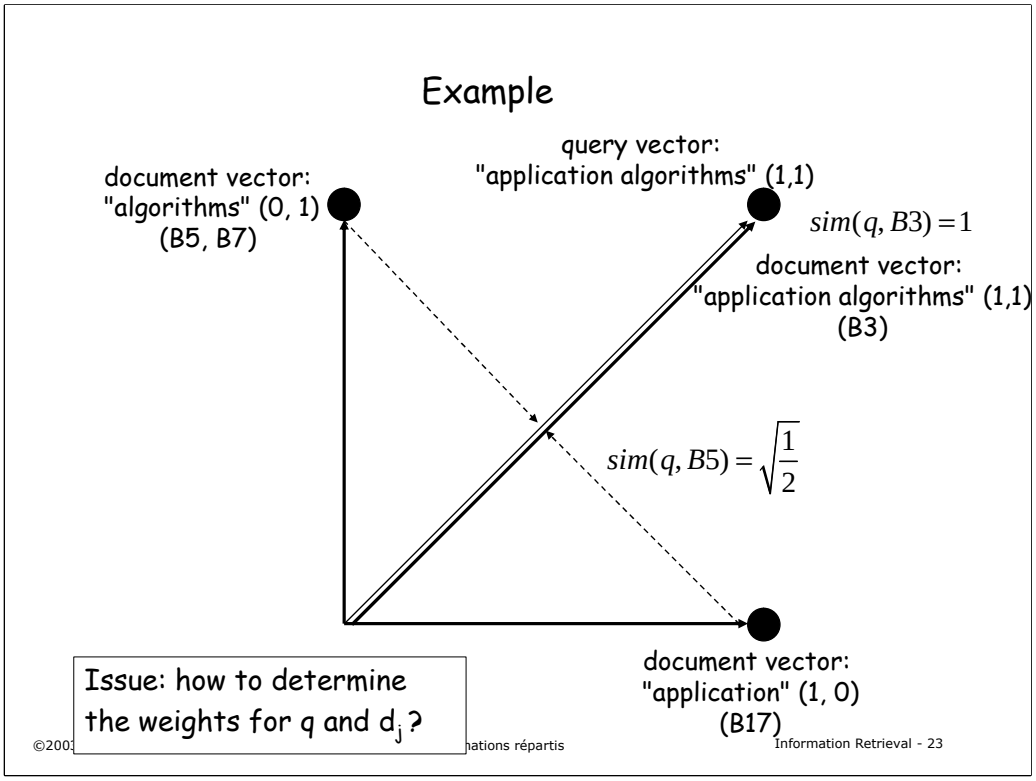
The distance measure for vectors has to satisfy the following properties:

•If two vectors coincide completely their similarity should be maximal, i.e. equal to 1.

•If two vectors have no keywords in common, i.e. if wherever the query vector has positive weights the document vector has weight 0, and vice versa – or in other words if the vectors are orthogonal – the similarity should be minimal, i.e. equal to 0.

•in all other cases the similarity should be between 0 and 1.

The scalar product (which is equivalent to the cosine of the angle of two vectors) has exactly these properties and is therefore (normally) used as similarity measure for vector space retrieval.



If we use a weighting scheme for document and query vectors as in Boolean retrieval, we would obtain for the example given earlier for Boolean retrieval, obtain with vector space retrieval the following result for a query: also documents containing only one of the two keywords occurring in the query, would show up in the result, although with lower similarity value.

Since in vector space retrieval no longer exclusively binary weights are used, a central question is of how to determine weights that more precisely determine the importance of a term for the document. Obviously not all terms carry the same amount of information on the meaning of a document (this was for example one of the reasons to eliminate stop words, as they normally carry no meaning at all)

Weights of Document Vectors: Term Frequency

- Documents are similar if they contain the same keywords (frequently)
  - Therefore use the frequency freq(i,j) of the keyword k_i in the document d_j to determine the weight

(Normalized) term frequency of term k_i in Document d_j

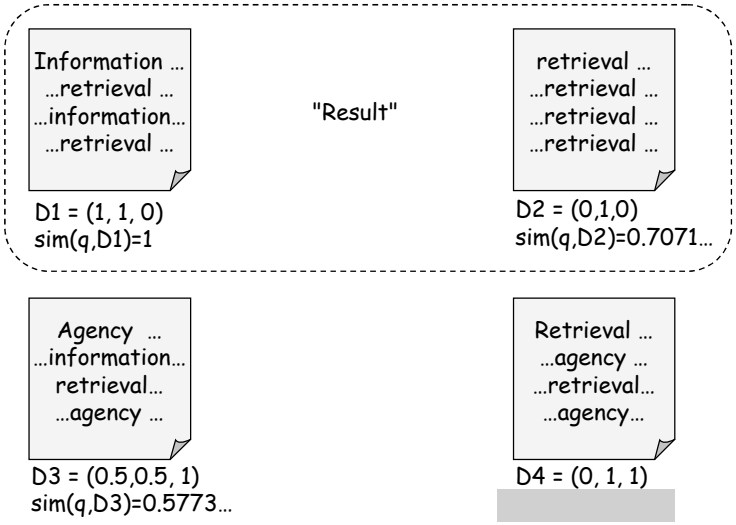
$$tf(i, j) = \frac{freq(i, j)}{\max_{k \in T} freq(k, j)}$$

An obvious difference that can be made among terms is with respect to their frequency of occurrence. Thus a weighting scheme for document weights can be defined by considering the (relative) frequency of terms within a document. The term frequency is normalized with respect to the maximal frequency of all terms occurring in a document.

Example

$$sim(\vec{q},\vec{d}_j)=\frac{\sum_{i=1}^{m} w_{ij} w_{iq}}{\left|\vec{d}_j\right|\left|\vec{q}\right|}$$

Vocabulary T = {information, retrieval, agency}
Query q = (information, retrieval) = (1,1,0)



This example illustrates the use of term frequency. Assume we form the query vector by simply setting a weight 1 if the keyword appears in the query. Then we would obtain D1 and D2 as result. Actually, this result appears to be non-intuitive, since we would expect that D3 is much more similar to D3 than D2. What has gone wrong?

The problem is that the term "retrieval", since it occurs very frequently in D2, leads to a high similarity value for D3. On the other hand the term retrieval has very little power to disambiguate meaning in this document collection, since every document contains this term. From an information-theoretic perspective one can state, that the term "retrieval" does not reduce the uncertainty about the result at all.

Inverse Document Frequency

- We have not only to consider how frequent a term occurs within a document (measure for similarity), but also how frequent a term is in the document collection of size n (measure for distinctiveness)

Inverse document frequency of term k_i

$$idf(i) = \log(\frac{n}{n_i}) \in [0, n]$$

n_i number of documents in which term k_i occurs

- Inverse document frequency can be interpreted as the amount of information associated with the term k_i

Term weight $w_{ij} = tf(i,j)\ idf(i)$

Thus we have to take into account not only the frequency of a term within a document, when determining the importance of the term for characterizing the document, but also the disambiguating power of the term with respect to the document collection as a whole. For that purpose the inverse document frequency is computed and included into the term weight.

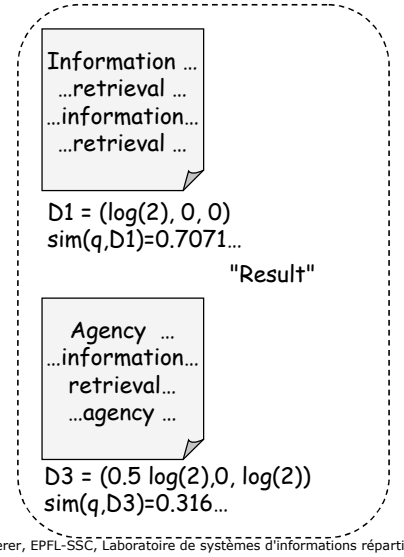
We can see now from this weighting scheme that eliminating stop words is actually an optimization of computing similarity measures in vector space retrieval. Since stop words normally occur in every document of a collection, their term weights will normally be 0 and thus the terms do not play a role in retrieval. Thus it is of advantage to exclude them already from the retrieval process at the very beginning.

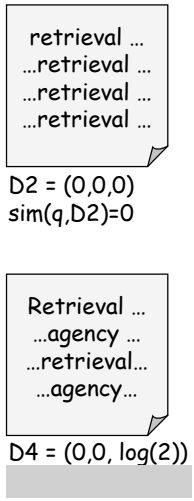
$$idf(i) = \log(\frac{n}{n_i}) \in [0, n]$$

Example

Vocabulary T = {information, retrieval, agency}
Query q = (information, retrieval) = (1,1,0)

idf(information)=idf(agency)=log(2)
idf(retrieval)=





We have now: N=4, $n_{information}$=2, $n_{retrieval}$=4, n_{agency}=2

The result corresponds much better to the "expectation" when using the inverse document frequencies.

Query Weights

- The same considerations as for document term weights apply also to query term weights

Query weight for query q

$$w_{iq} = \frac{freq(i,q)}{\max_{k \in T} freq(k,q)}) \log(\frac{n}{n_i})$$

- Example: Query q = (information, retrieval)
  - Query vector: (log(2), 0, 0)
  - Scores: sim(q, D1)= 0.569…
    sim(q, D2)=0
    sim(q, D3)=0.254
    sim(q, D4)=0

Finally, we have to look at the question of how to determine the weights for the query vector. One can apply the same principles as for determining the document vector, as is shown. In practice there exist a number of variations of this approach.

Example

- Query q = "application theory"
- Boolean retrieval result
  - application AND theory: B3, B17
  - application OR theory: B3, B11, B12, B17
- Vector retrieval result
  - Query vector (0, 2.14…, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.447…)
  - Ranked Result:

B17	0.770078…
B3	0.684042…
B12	0.232951…
B11	0.232951…

This examples gives a small illustration of the differences of Boolean and vector space retrieval.

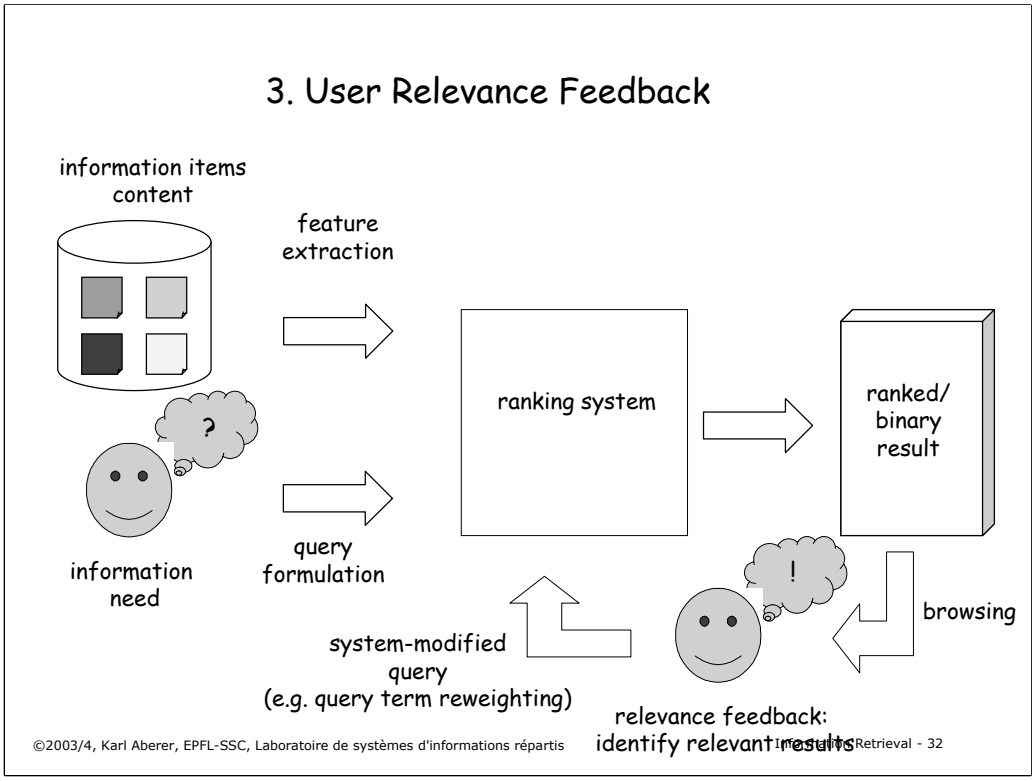
Discussion of Vector Retrieval Model

- The vector model with tf-idf weights is a good ranking strategy for general collections
  - many alternative weighting schemes exist, but are not fundamentally different
- Advantages
  - term-weighting improves quality of the answer set
  - partial matching allows retrieval of docs that approximate the query conditions
  - cosine ranking formula sorts documents according to degree of similarity to the query
- Disadvantages
  - assumes independence of index terms
  - not clear that this is a disadvantage

We summarize here the main advantages of the vector space retrieval model. It has proven to be a very successful model for general text collections, i.e. if there exists no additional (context) information on the documents that could be exploited, e.g. from a specific application domain. Providing a ranked results improves the usability of the approach. The model inherently assumes that there exist no dependencies in the occurrence of the terms, i.e. that certain terms appear together more frequently than others. Studies have however shown that taking such co-occurrence probabilities additionally into account can actually HURT the performance of the retrieval system. The reason is that co-occurrence probabilities are often related to specific application domains and thus do not easily transfer to general-purpose retrieval.

Summary

- How are the weights of document vectors and query vectors computed in Booelan retrieval ?
- How is the similarity coefficient computed in Boolean retrieval ?
- What is the basic abstraction the vector model uses to determine similarity of documents ?
- What are document frequency and inverse document frequency ?
- Why is inverse document frequency used in vector retrieval ?
- How are the weights of document vectors and query vectors computed in vector retrieval ?
- How is the similarity coefficient computed in vector retrieval ?
- Which documents receive similarity value of zero in vector retrieval ?



The user does not necessarily know

- what his information need is
- how to appropriately formulate the query
- BUT he can identify relevant documents

Therefore the idea of user relevance feedback is to reformulate the query taking into account feedback of the user on the relvance of retrieved documents.

The advantages of such an approach are the following:

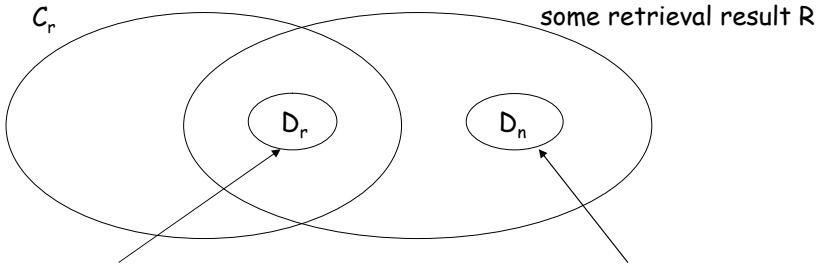
- The user is not involved in query formulation, just points to interesting data items.
- The search task can be split up in smaller steps.
- The search task becomes a process converging to the desired result.

Identifying Relevant Documents

- If C_r is the set of relevant documents, then the optimal query vector is

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\vec{d}_j \notin C_r} \vec{d}_j$$

- Idea: approximate this vector by letting the user identify relevant and non-relevant documents



If we would know the exact set of relevant documents, it can be proven, that the optimal query vector could be expressed in terms of the document vectors as shown above. Intuitively this expression can be well understood: it gives, proportional to the frequency of the documents, positive weights to document vectors in the set of relevant documents and negative weights for the others. The problem for practical retrieval is of course that we do not know C_r. However, with user relevance feedback we have the possibility to **approximate** this optimal query vector.

Calculation of Expanded Query

- If users identify some relevant documents D_r from the result set R or a retrieval query q
  - Assume all elements in $R \setminus D_r$ are not relevant, i.e. $D_n = R \setminus D_r$
  - Modify the query to approximate the theoretically optimal query (Rochio)

$$\vec{q}_{approx} = a\vec{q} + \frac{b}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{g}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r}\vec{d}_j$$

α, β, γ are tuning parameters

The best approximation for C_r that we can obtain is by considering the set of documents D_r that the user indicated to be relevant, as the set of relevant documents. This set is used to modify the original query vector in a way that tries to approximate the optimal query vector. The scheme for doing this shown here is called Rochio scheme. There exist several variations of this scheme that however follow all the same principle. The tuning parameters are used to set the relative importance of

- Keeping the original vector
- Increasing the weight of vectors from D_r
- Decreasing the weights from vectors of the complement of D_r

Example

$$\vec{q}_{approx} = a\vec{q} + \frac{b}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{g}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r}\vec{d}_j$$

- Query q= "application theory"
- Result

0.77: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.68: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.23: B11 Oscillation Theory for Neutral Differential Equations with Delay
0.23: B12 Oscillation Theory of Delay Differential Equations

- Query reformulation

$$\vec{q}_{approx} = \frac{1}{4}\vec{q} + \frac{1}{4}\vec{d}_3 - (\vec{d}_{17} + \vec{d}_{12} + \vec{d}_{11}), a = b = g = \frac{1}{4}$$

- Result for reformulated query

0.87: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.61: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.29: B7 Knapsack Problems: Algorithms and Computer Implementations
0.23: B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra

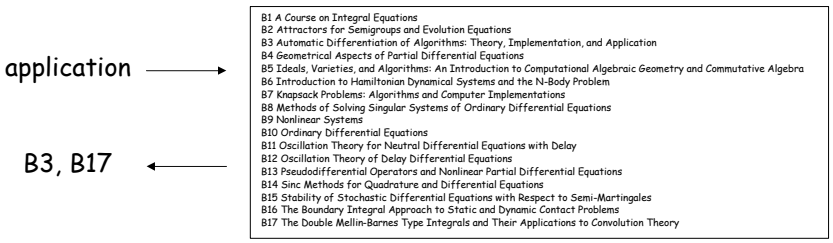
This example shows how query reformulation works. By identifying document B3 as being relevant and modifying the query vector it turns out that new documents (B5 and B7) become relevant. The reason is that those new documents share terms with document B3, and these terms are newly considered in the reformulated query.

Summary

- Which capability of users is taken advantage of with relevance feedback ?
- Which query vector is approximated when adapting a user query with relevance feedback ?
- Can documents which do not contain any keywords of the original query receive a positive similarity coefficient after relevance feedback ?

4. Inverted Files

- Problem: text retrieval algorithms need to find words in documents efficiently
  - Boolean retrieval, vector space retrieval
  - Given index term k_i, find document d_j



- An inverted file is a word-oriented mechanism for indexing a text collection in order to speed up this search task
  - Addressing of documents and word positions within documents
  - Most frequently used indexing technique for large text databases
  - Appropriate when the text collection is large and semi-static

In order to implement text retrieval models efficiently, efficient search for occurrences in documents must be supported. For that purpose different indexing techniques exist, among which inverted files are the by far most widely used. Inverted file are optimized for supporting search on relatively static text collections. For example no database updates are supported with inverted files. This distinguishes inverted files from typical database indexing techniques, such as B+-Trees.

Inverted Files

Inverted list l_k for a term k

$$l_k = \langle f_k : d_{i_1}, \ldots, d_{i_{fk}} \rangle$$

f_k number of documents in which k occurs
$d_{i1},\ldots,d_{ifk}$ list of document identifiers of documents containing k

Inverted File: lexicographically ordered sequence of inverted lists

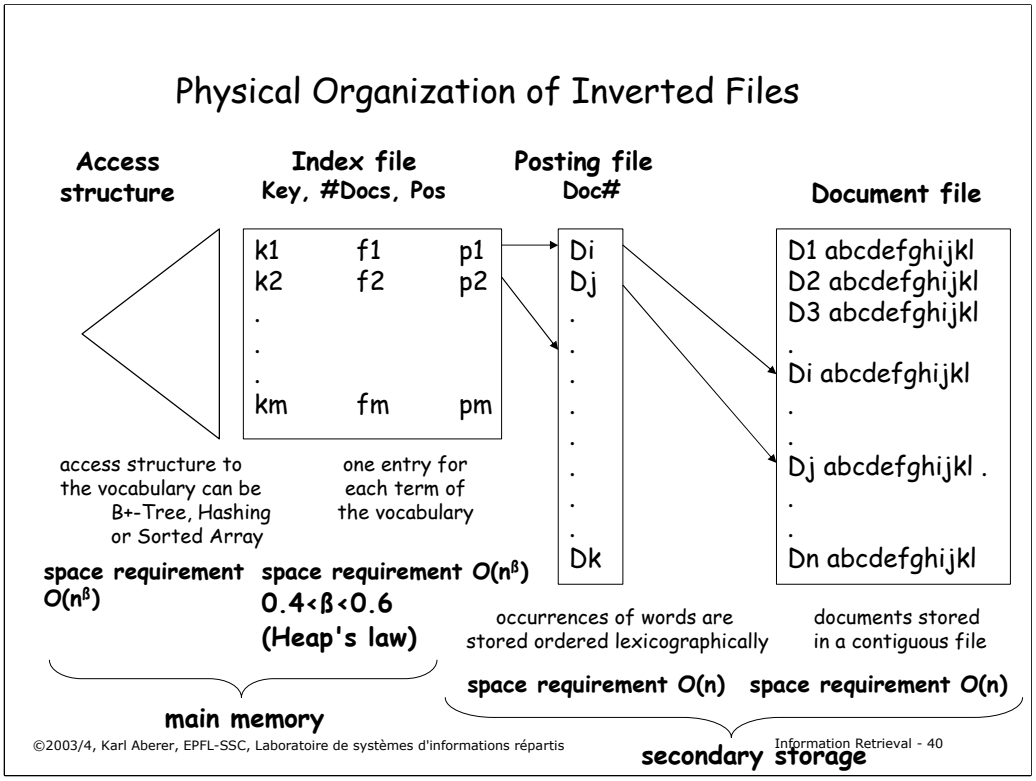
$$IF = \langle i, k_i, l_{k_i} \rangle, i = 1, \ldots, m$$

Inverted files are constructed by concatenating the inverted lists for all keywords occurring in the document collection. The inverted lists enumerate all occurrences of the keyword in documents, by keeping the document identifiers and the frequency of occurrence (this is useful for determining inverse document frequency, for example).

Example

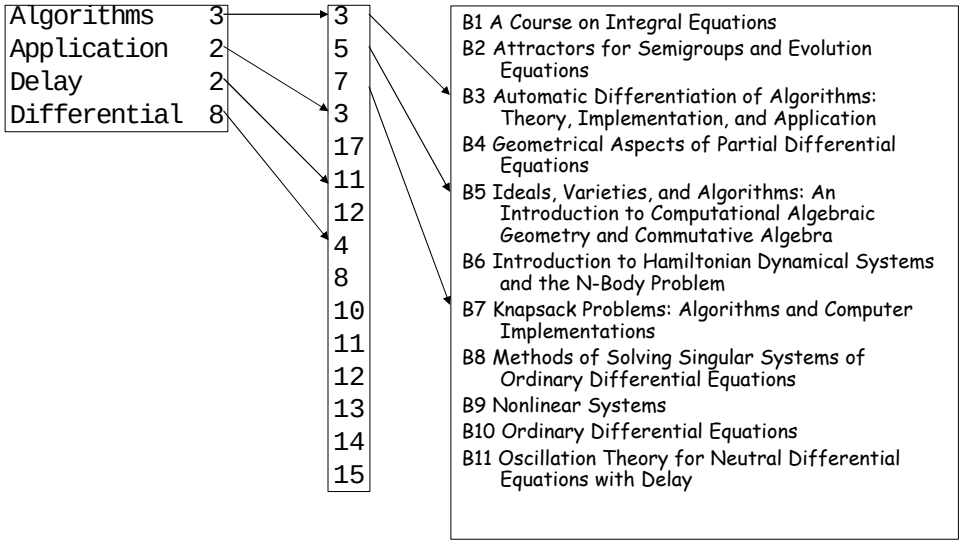
1	Algorithms	3	:	3	5	7						
2	Application	2	:	3	17							
3	Delay	2	:	11	12							
4	Differential	8	:	4	8	10	11	12	13	14	15	
5	Equations	10	:	1	2	4	8	10	11	12	13	14	15
6	Implementation	2	:	3	7							
7	Integral	2	:	16	17							
8	Introduction	2	:	5	6							
9	Methods	2	:									
10	Nonlinear	2	:	9	13							
11	Ordinary	2	:	8	10							
12	Oscillation	2	:	11	12							
13	Partial	2	:	4	13							
14	Problem	2	:	6	7							
15	Systems	3	:	6	8	9						
16	Theory	4	:	3	11	12	17					

Here we display the inverted list that is obtained for our example document collection.



Inverted files as introduced before are a logical data structure for which we have to find a physical storage organization. This physical organization has to take into account the quantitative characteristics of the inverted file structure. The important observation is: the number of references to documents, corresponding to the occurrences of index terms in the documents is much larger than the number of index terms, and thus the number of inverted lists. In fact, the order of magnitude of occurrences of index terms is O(n), whereas the number of index terms is typically O(n^ß), where ß is roughly 0.5. For example, a document collection of size n= 10^6 would have approximate m=10^3 index terms. Therefore the index terms and the corresponding frequencies of occurrences can be kept in main memory, whereas the references to documents are kept in secondary storage. As a result an index file is kept in memory. The access to this index file is supported by any suitable data access structure. Typically binary search, hash tables or tree-based structures, such as B+-Trees, or tries are used for that purpose. The posting files consists of the sequence of all occurrences of the inverted file. The index file is related to the posting file by keeping for each index term a reference to the

Example



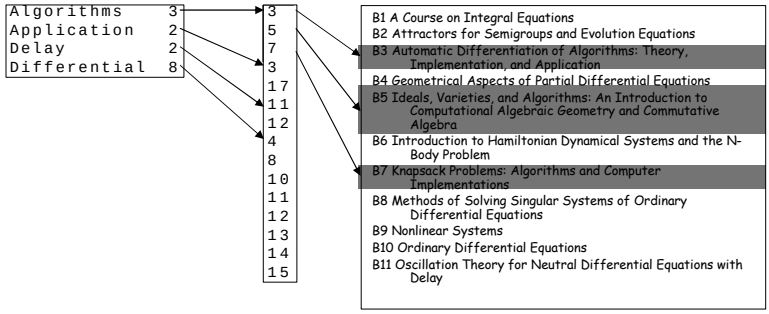
This would be the physical organization of the inverted file for the running example. Note that only part of the data is displayed.

Searching the Inverted File

Step 1: Vocabulary search
- the words present in the query are searched in the index file

Step 2: Retrieval of occurrences
- the lists of the occurrences of all words found are retrieved from the posting file

Step 3: Manipulation of occurrences
- the occurrences are processed in the document file to process the query



Search in an inverted file is a straightforward procedure. Using the data access structure first the index terms occurring in the query are searched in the index file. Then the occurrences can be sequentially retrieved from the postings file. Afterwards the corresponding document portions are accessed and can be processed (e.g. counting frequencies).

Construction of the Inverted File

Step 1: Search phase

- The vocabulary is kept in a trie data structure storing for each word a list of its occurrences
- Each word of the text is read sequentially and searched in the vocabulary
- If it is not found, it is added to the vocabulary with an empty list of occurrences
- The word position is added to the end of its list of occurrences

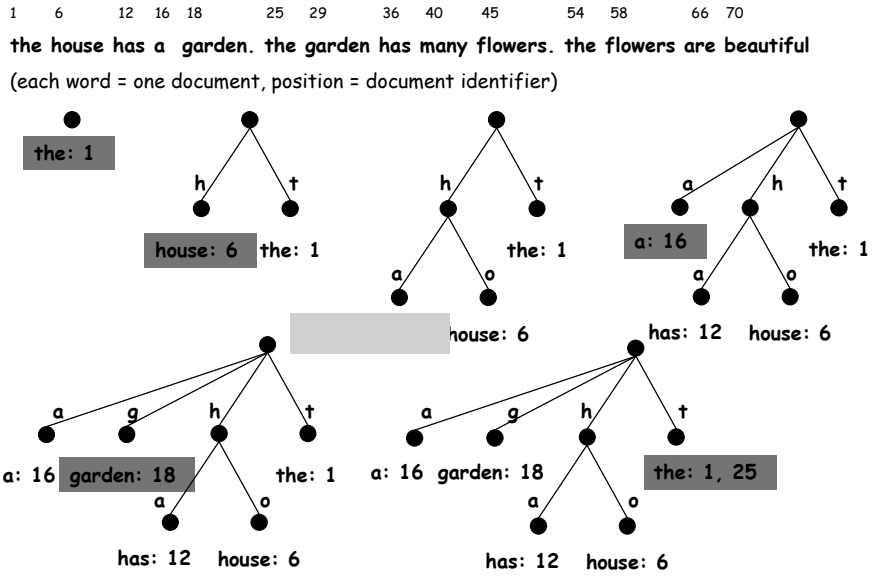
Step 2: Storage phase (once the text is exhausted)

- The list of occurrences is written contiguously to the disk (posting file)
- The vocabulary is stored in lexicographical order (index file) in main memory together with a pointer for each word to its list in the posting file

Overall cost O(n)

The index construction is performed by first constructing dynamically a trie structure, in order to build up a sorted vocabulary and to collect the occurrences of index terms. After the complete document collection has been traversed the trie structure is sequentially traversed and the posting file is written to secondary storage. The trie structure itself can be used as a data access structure for the index file that is kept in main memory.

Example

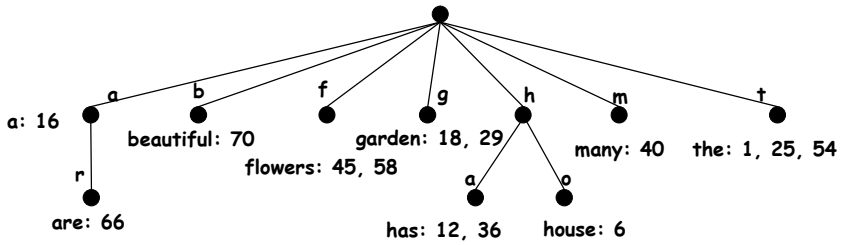


In this example we consider each word of the text as a separate document identified by its position (for space limitations). We demonstrate the initial steps of constructing the trie structure and entering into it the occurrences of index terms. The changes to the trie structure are highlighted for each step. Note that in the last step the tree structure of the trie does not change, since the index term "the" is already present.

Example

1 6 12 16 18 25 29 36 40 45 54 58 66 70

the house has a garden. the garden has many flowers. the flowers are beautiful



inverted file I

a: 16
are: 66
beautiful: 70
flowers: 45, 58
garden: 18, 29
has: 12, 36
house: 6
many: 40
the: 1, 25, 54

16, 66, 70, 45, 58, 18, 29, 12, 36, 6, 40

postings file

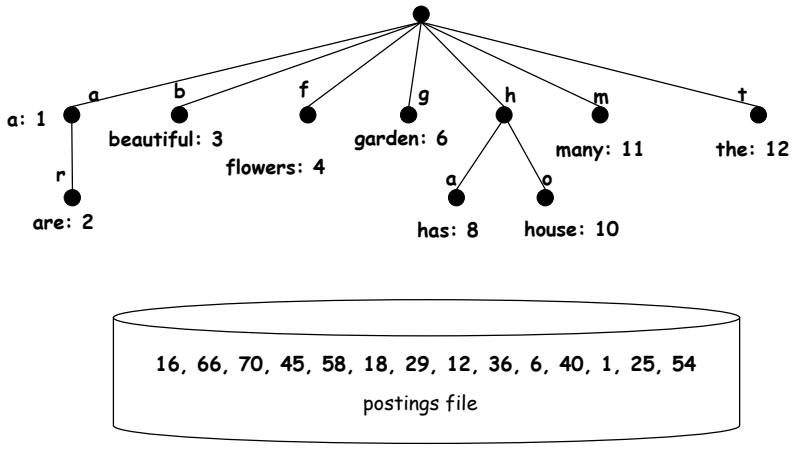
Once the complete trie structure is constructed the inverted file can be derived from it. For doing this the trie is traversed top-down and left-to-right. Whenever an index term is encountered it is added to the end of the inverted file. Note that if a term is prefix of another term (such as "a" is prefix of "are") index terms can occur on internal nodes of the trie.

Analogously to the construction of the inverted file also the posting file can be derived.

Example

1 6 12 16 18 25 29 36 40 45 54 58 66 70

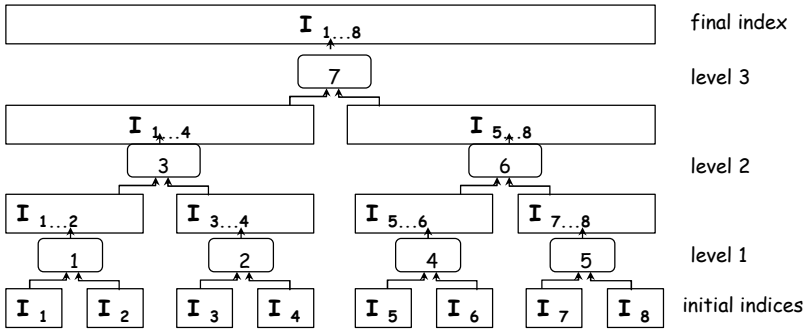
the house has a garden. the garden has many flowers. the flowers are beautiful



Considering the physical organization of the inverted file the result can be displayed as shown. The trie structure constructed is a possible access structure to the index file in main memory. Thus the entries of the index files occur as leaves (or internal nodes) of the trie. Each entry has a reference to the position of the postings file that is held in secondary storage.

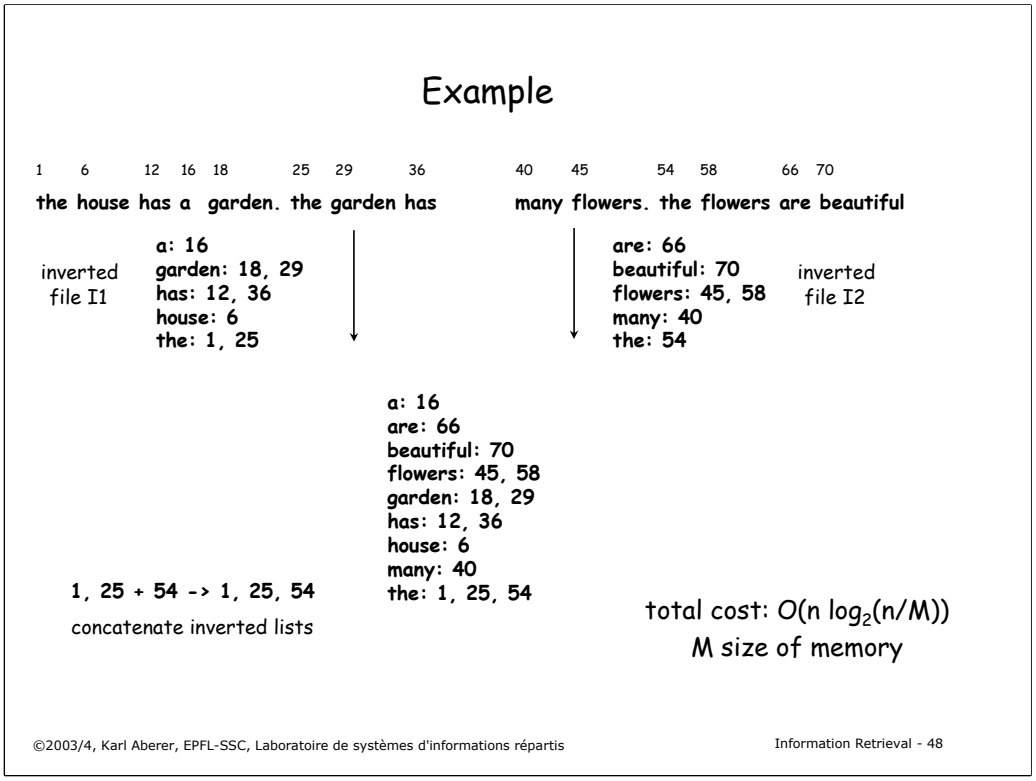
Index Construction in Practice

- In practice not all index information can be kept in main memory
- Index merging
  - When no more memory is available, a partial index I_i is written to disk
  - The main memory is erased before continuing with the rest of the text
  - Once the text is exhausted, a number of partial indices I_i exist on disk
  - The partial indices are merged to obtain the final index



In practice the construction will be inefficient or impossible if the size of the intermediate trie structure constructed exceeds the main memory space. Then the index construction process has to be partitioned in the following way: While the document collection is sequentially traversed, partial indices are written to the disk whenever the main memory is full. This results in a number of partial indices, indexing consecutive partitions of the text. In a second phase the partial indices need to be merged into one index.

This figure illustrates the merging process: in this example 8 partial indices have been constructed. Step by step the indices are merged, by merging two indices into one, until one final index remains. The merging can be performed, such that the two partial indices which are to be merged are scanned in parallel sequentially on the disk, and while scanning the resulting index is written sequentially to the disk.



Merging the indices requires first merging the vocabularies. As we know, the vocabularies are comparably small and thus the merging of the vocabularies can take place in main memory. In case a vocabulary term occurs in both partial indices, their list of occurrences from the posting file need to be combined. Here we can take advantage of the fact that the partial indices have been constructed by sequentially traversing the document file. Therefore these lists can be directly concatenated without sorting.

The total computational complexity of the merging algorithm is $O(n \log_2(n/M))$. This implies that the additional cost of merging as compared to the purely main memory based construction of inverted files is a factor of $O(\log_2(n/M)))$. This is small in practice, e.g. if the database size n is 64 times larger than the main memory size, then this factor would be 6.

This example illustrates how the merging process can be performed for example when the database is partitioned into two parts.

Adressing Granularity

- Documents can be addressed at coarser and finer granularities
  - coarser: text blocks spanning multiple documents
  - finer: paragraph, sentence, word level
- General rule
  - the finer the granularity the less post-processing but the larger the index
- Example: index size in % of document collection size

Index	Small collection (1Mb)	Medium collection (200Mb)	Large collection (2Gb)
Addressing words	73%	64%	63%
Addressing documents	26%	32%	47%
Addressing 256K blocks	25%	2.4%	0.7%

The posting file has the by far largest space requirements. An important factor for the size of an inverted file is the addressing granularity used. The addressing granularity determines of how exactly positions of index terms are recorded in the posting file. There exist three main options:

•Exact word position

•Occurrence within a document

•Occurrence within an arbitrary sized block = equally sized partitions of the document file spanning probably multiple documents

The larger the granularity, the fewer entries occur in the posting file. In turn, with coarser granularity additional postprocessing is required in order to determine exact positions of index terms.

Experiments illustrate the substantial gains that can be obtained with coarser addressing granularities. Coarser granularities lead to a reduction of the index size for two reasons:

•a reduction in pointer size (e.g. from 4 Bytes for word addressing to 1 Byte with block addressing)

•and a lower number of occurrences.

Index Compression

- Documents are ordered and each document identifier d_{ij} is replaced by the difference to the preceding document identifier
  - Document identifiers are encoded using fewer bits for smaller, common numbers

$$l_k = \langle f_k : d_{i_1}, \ldots, d_{i_{fk}} \rangle \rightarrow$$

$$l_k' = \langle f_k : d_{i_1}, d_{i_2} - d_{i_1}, \ldots, d_{i_{fk}} - d_{i_{fk}-1} \rangle$$

- Use of varying length compression further reduces space requirement
- In practice index is reduced to 10- 15% of database size

X	code(X)
1	0
2	10 0
3	10 1
4	110 00
5	110 01
6	110 10
7	110 11
8	1110 000
63	111110 11111

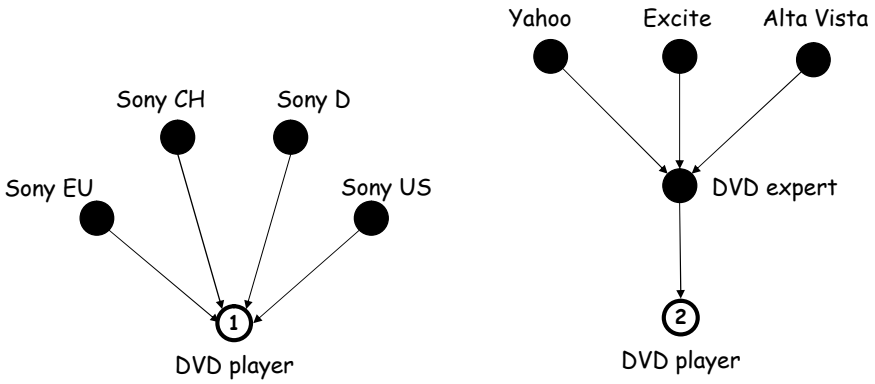
A further reduction of index size can be achieved by applying compression techniques to the inverted lists. In practice the inverted list of a single term can be rather large. A first improvement is achieved by storing only differences among subsequent document identifiers. Since they occur in sequential order the differences are much smaller integers than the absolute values.

In addition number encoding techniques can be applied to the resulting integer values. Since small values will be more frequent than large ones this leads to a further reduction in the size of the posting file.

Summary

- Which aspect of information retrieval is addressed by inverted files ?
- How do inverted files compare to other database indexing approaches ?
- How is an inverted list organized and which are methods to compress it ?
- Which is the file organization of an inverted file ?
- Why are different addressing granularities used in inverted files ?
- Which problem occurs in the construction of an inverted file and how is it addressed ?

5. Web Information Retrieval



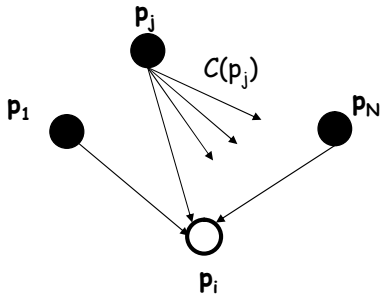
- Full text retrieval result with equal ranking; which page is more relevant ?
  - relevance related to number of referrals (incoming links)
  - relevance related to number of referrals with high relevance

When retrieving documents from the Web the link structure bears important information on the relevance of documents. Generally speaking, a document that is referred more often by other documents by Web links, is likely to be of higher interest and therefore relevance. So a possibility to rank documents is by considering the number of incoming links. Doing this allows to distinguish documents that otherwise would be ranked equally or similarly when relying on text retrieval solely.

However, when doing this, also the importance of documents having a link to the document to be ranked may be different. Therefore not only counting then number of incoming links, but also weighting the links by the relevance of documents from which the links emanate appears to be appropriate. The same reasoning of course again applies then again for evaluating the relevance of documents pointing to the document and so forth.

Random Walker Model

- Assumption: If a random walker visits a page more often it is more relevant: takes into account the number of referrals AND the relevance of referrals



n is the number of Web pages

C(p) is the number of outgoing links of page p

$P(p_i)$ probability to visit page p_i, where page p_i is pointed to by pages p_1 to p_n = **relevance**

$$P(p_i) = \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}$$

In order to capture the process of evaluating the relevance of documents by considering incoming links, assuming the relevance of documents is known (which is a recursive procedure) the random walker model is used. The intuitive idea is that if a random walker on the web graph visits a document more often the document is more relevant. This intuition is captured mathematically in a recursive equation characterizing the probability of visiting a specific page by a random walker. The assumption on the random walker used in this mathematical formulation of the process is that it follows every link of document with equal probability.

Transition Matrix for Random Walker

- The definition of $P(p_i)$ can be reformulated as matrix equation

$$R_{ij} = \begin{cases} \frac{1}{C(p_j)}, if\ p_j \rightarrow p_i \\ 0, otherwise \end{cases}$$

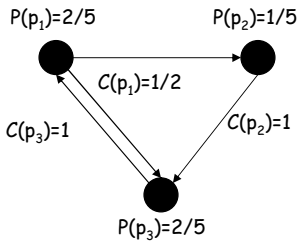
$$\vec{p} = (P(p_1), ..., P(p_n))$$

$$\vec{p} = R.\vec{p}, \quad \left\|\vec{p}\right\|_1 = \sum_{i=1}^{n} p_i = 1$$

- The vector of relevance of pages is an Eigenvector of the matrix R

If we consider the matrix of transition probabilities for the random walker, the recursive definition of the probability to visit a Web page can be formulated as matrix equation. More precisely the resulting equation shows that the vector of probabilities for visiting the Web pages is an Eigen vector of the transition matrix. In fact, it is the one corresponding to the largest Eigen value.

Example



Links from p_1

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Links to p_1

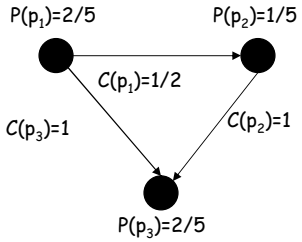
Link Matrix

$$R = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} \frac{2}{5} \\ \frac{1}{5} \\ \end{pmatrix}$$

Ranking p_1, $p_3 > p_2$

This example illustrates the computation of the probabilities for visiting a specific Web page. The values $C(p_i)$ correspond to the transition probabilities. They can be derived from the link matrix, which is the matrix with entries 1 at (i,j) if there exists a link from p_i to p_j, by dividing the values in the columns by the sum of the values found in the column. The probability of a random walker being in a node is then obtained from the Eigen vector of this matrix.

Modified Example



Links from p_1

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Links to p_1

Link Matrix

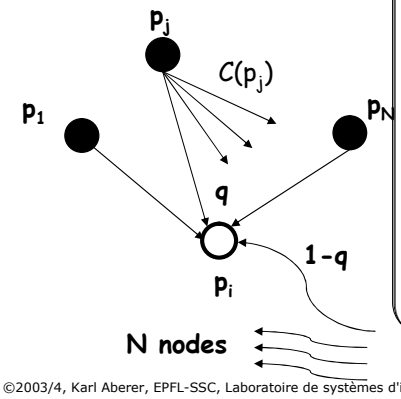
$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

No Ranking

The approach described so far has however a problem. When looking at the modified example we see that there exists a node p_3 that is a "sink of rank". Any random walk ends up in this sink, and therefore the other nodes do not receive any ranking weight. Consequently also the rank of sink does not. Therefore the only solution to the equation p=Rp is the zero vector.

Source of Rank

- Assumption: random walker jumps with probability 1-q to an arbitrary node: thus nodes without incoming links are reached
- PageRank method



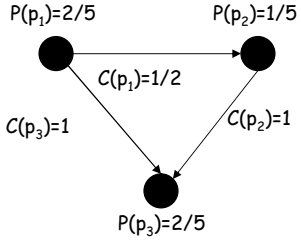
$$P(p_i) = c(\frac{(1-q)}{n} + q \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}), c \le 1$$

$$\vec{p} = c((qR + (1-q)E)).\vec{p}, \quad E = \left[\frac{1}{n}\right]_{nxn}$$

$$\vec{p} = c(qR.\vec{p} + \frac{(1-q)}{n}\vec{e}), \quad \vec{e} = (1,...,1)$$

To avoid the previously described problem we add a "source of rank". The idea is that a random walker in each step can rather than following a link jump to any page with probability 1-q. Therefore also pages without incoming links can be reached by the random walk. In the mathematical formulation of the random walk this means that a term for the source of rank is added. Since at each step the random walker makes a jump with probability 1-q and any of the N pages is reached with the same probability the additional term is (1-q)/N. Reformulating this again as matrix equation means adding a NxN Matrix E with all entries being 1/N. This is equivalent to saying that with probability 1/N transitions among any pairs of nodes (including transition from a node to itself) are performed. Since the vector p has norm 1, i.e. the sum of the components is exactly 1, E.p=e, where e is the unit vector, and the matrix equation can be reformulated in the second form shown. The method described is called PageRank and is used by Google.

Modified Example



$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, E = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}, q = 0.9$$

$$qR + (1-q)E = \begin{pmatrix} \frac{1}{30} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{14}{15} & \frac{1}{30} \end{pmatrix} \quad \vec{p} = \begin{pmatrix} 0.123 \\ 0.275 \\ 0.953 \end{pmatrix}$$

Ranking $p_3 > p_2 > p_1$

With the modification of rank computation using a source of rank, we obtain for our example again a non-trivial ranking which appears to be appropriate.

Practical Computation of PageRank

- Iterative computation

e termination criterion
s arbitrary vector, e.g. s=e

$$\vec{p}_0 \leftarrow \vec{s}$$
$$while d > e$$
$$\vec{p}_{i+1} \leftarrow qR \bullet \vec{p}_i$$
$$\vec{p}_{i+1} \leftarrow \vec{p}_{i+1} + \frac{(1-q)}{n}\vec{e}$$
$$d \leftarrow \left\| \vec{p}_{i+1} - \vec{p}_i \right\|_1$$

For the practical determination of the PageRank ranking an iterative computation is used. It is derived from the second form of the formulation of the visiting probabilities of the random walker that we have given. The vector e used to add a source of rank not necessarily has to assign uniform weights to all pages, but might reflect itself a ranking of Web pages.

Example: ETHZ Page Rank

Doc_ID	Rank_Value	URL
1	0.002536	http://www.ethz.ch/
146	0.002292	http://www.ethz.ch/r_amb/
10	0.000654	http://www.ethz.ch/default_de.asp
35	0.000511	http://www.rereth.ethz.ch/
376124	0.000503	http://computing.ee.ethz.ch/sepp/matlab-5.2-to/helpdesk.html
67378	0.000497	http://computing.ee.ethz.ch/sepp/
59887	0.000485	http://www.computing.ee.ethz.ch/sepp/
89307	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
216716	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
147932	0.000484	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
175544	0.000484	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
186766	0.000478	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
228634	0.000477	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.1ref/docs/api/overview-summary.html
228421	0.000464	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
3161	0.00045	http://www.ethz.ch/r_amb/reto_ambuehler.html
215673	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/files.html
259672	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/globals.html
259671	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/functions.html
259670	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/annotated.html
259669	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/classes.html

Figure 1: Top 20 of ETH Zurich Web Documents

These are the top documents from the PageRank ranking of all Web pages at ETHZ.

Practical Usage of PageRank

- PageRank is part of the ranking method used by Google
  - Compute the global PageRank for all Web pages
  - Given a keyword-based query retrieve a ranked set of documents using standard text retrieval methods
  - Merge the ranking with the result of PageRank to both achieve high precision (text retrieval) and high quality (PageRank)
  - Google uses also other methods to improve ranking (trade secret)
- Main problems
  - Crawling the Web
  - Efficient computation of Page Rank for large link databases
  - Combination with other ranking methods (text)
- Some (old) numbers (1998)
  - 24 million Web pages, 75 million links, 300 MB link database
  - Convergence in roughly 50 iterations on one workstation
  - One iteration takes 6 minutes

PageRank is used as one criterion to rank result documents in Google. Essentially Google uses text retrieval methods to retrieve relevant documents and then applies PageRank to create a more appropriate ranking. Google uses also other methods to improve ranking, e.g. by giving different weights to different parts of Web documents. For example, title elements are given higher weight. The details of the ranking methods are trade secrets of the Web search engine providers.

Other issues of Web search engines are crawling the Web, which requires techniques that can explore the Web without revisiting pages too frequently. Also the enormous size of the document and link database poses implementation challenges in order to keep the ranking computations scalable.

Summary

- Which additional source of information can be used to rank Web pages ?
- What is the informal idea underlying PageRank ?
- Why is a source of rank used in PageRank and what can it be used for ?
- How is PageRank practically computed ?

References

- Course material based on
  - Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Modern Information Retrieval (ACM Press Series), Addison Wesley, 1999.
- Relevant articles
  - Sergey Brin , Lawrence Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, v.30 n.1-7, p.107-117, April 1, 1998.
  - Jon M. Kleinberg: Authoritative Sources in a Hyperlinked Environment. JACM 46(5): 604-632 (1999)