

Query Languages

Keyword-based querying

- Queries are combinations of words.
- The document collection is searched for documents that contain these words.
- Word queries are intuitive, easy to express and provide fast ranking.
- The concept of *word* must be defined.
 - A word is a sequence of *letters* terminated by a *separator* (period, comma, space, etc).
 - Definition of *letter* and *separator* is flexible; e.g., hyphen could be defined as a letter or as a separator.
 - Usually, common words (such as “a”, “the”, “of”, ...) are ignored.

Single-word queries

- A query is a single word
 - Usually used for searching in document images
- Simplest form of query.
- What are the possible documents retrieved as relevant?
 - All documents that include this word are retrieved.
- On what base documents are ranked?
 - Documents may be ranked by the *frequency* of the query word in the document.
 - Documents containing more of the query word are given the highest priority

Phrase queries

- A query is a sequence of words treated as a single unit. Also called “literal string” or “exact phrase” query.
 - Phrase is usually surrounded by quotation marks.
 - All documents that include this phrase are retrieved.
- Usually, separators (commas, colons, ...) & common words (“a”, “the”, “of”, “for”...) in the phrase are ignored
- In effect, this query is for a *set* of words that must appear *in sequence*.
 - Allows users to specify a *context* and thus gain precision.
 - Ex.: “Information Processing for Document Retrieval”.
- What are the possible documents retrieved as relevant?
 - All documents that include phrase query are retrieved.
- On what base documents are ranked?

Multiple-word queries

- A query is a set of words (or phrases).
 - Ex.: what is the result for the query “Data Mining and Intelligent Database Design”?
- What are the possible documents retrieved as relevant?
 - Two options: A document is retrieved if it includes
 - any of the query words, or
 - each of the query words.
- On what bases documents be ranked to list according to best matching principle?
 - Documents are ranked by the number of query words they contain. A document containing n query words is ranked higher than a document containing $m < n$ query words.
 - Documents are ranked in decreasing order:
 - those containing all the query words are ranked at the top, only one query word at bottom.
 - Frequency counts may be used to break ties among documents that contain the same query words.

Boolean queries

- Queries are formulated based on concepts from logic: AND, OR, NOT
 - *It describes the information needed by relating multiple words with Boolean operators.*
- **Semantics:** For each query word w a corresponding set D_w is constructed that includes the documents that contain w .
- The Boolean expression is then interpreted as an expression on the corresponding document sets with corresponding set operators:
 - **AND:** Finds only documents containing all of the specified words or phrases.
 - **OR:** Finds documents containing at least one of the specified words or phrases.
 - **NOT:** Excludes documents containing the specified word or phrase.

Examples: **Boolean queries**

*1. computer **OR** server*

-Finds documents containing either computer, server or both

*2. (computer **OR** server) **NOT** mainframe*

- Select all documents that discuss computers or servers, do not select any documents that discuss mainframes.

*3. computer **NOT** (server **OR** mainframe)*

-Select all documents that discuss computers, and do not discuss either servers or mainframes.

*4. computer **OR** server **NOT** mainframe*

-Select all documents that discuss computers, or documents that discuss servers but do not discuss mainframes.

Pattern queries

- ***What is Pattern?***
 - An expression that defines a *set* of objects. Pattern shows the internal representation of an object.
 - What is the pattern of a word?
- ***Pattern matching:*** A word *matches* a pattern if it is equal to *one* of the words defined by the pattern. In other words,
 - the semantics are of *disjunction*: A pattern P that defines a word $(c_1, c_2, \dots, c_n)$ is interpreted as $c_1 \vee c_2 \vee \dots \vee c_n$.
- ***Similarity pattern.*** Specifies a string and a *radius*
 - *defines* all the words whose *distance from* the string is within the radius.
 - Assume the distance between two strings is measured by the number of one-character changes (insertions, deletions, replacements) required to transform one string into the other.
 - The similarity pattern $(king, 2)$ defines *kin, kong, knig, kings, cling, ...*
- Useful to compensate for typing or scanning (OCR) errors.
 - One of the technique used for pattern matching is **string editing**

String editing

- The problem is given two sequences of symbols, $X = x_1 x_2 \dots x_n$ and $Y = y_1 y_2 \dots y_m$, transform X to Y , based on a sequence of three operations: **Delete**, **Insert** and **Replace**, so that for every operation $\text{COST}(C_{ij})$ is incurred.
- The objective of string editing is to identify a minimum cost sequence of edit operation that will transform X into Y .

Example: consider the sequences

$X = \{a a b a b\}$ and $Y = \{b a b b\}$

Identify a minimum cost sequence of edit operation that transform X into Y . Assume change costs 2 units, delete 1 unit and insert 1 unit.

Dynamic programming

- The minimum cost of any edit sequence that transforms $x_1 x_2 \dots x_i$ into $y_1 y_2 \dots y_j$ (for $i > 0$ and $j > 0$) is the minimum of the three costs: delete, replace, or insert operations.
- The following recurrence equation is used for $COST(i,j)$.

$$COST(i,j) = \begin{cases} 0 & \text{if } i=0, j=0 \\ COST(i-1,0) + D(x_i) & i > 0, j=0 \\ COST(0,j-1) + I(y_j) & i=0, j > 0 \\ COST'(i,j) & i > 0, j > 0 \end{cases}$$

$$\text{where } COST'(i,j) = \min \{ \begin{array}{l} COST(i-1,j) + D(x_i), \\ COST(i-1,j-1) + C(x_i, y_j), \\ COST(i,j-1) + I(y_j) \end{array} \}$$

Example

Transform the sequences

$X_i = \{a \ a \ b \ a \ b\}$ into $Y_j = \{b \ a \ b \ b\}$

with minimum cost sequence of edit operation
using dynamic programming approach,
Assume that change costs 2 units, delete and
insert 1 unit.

j \ i	0	1	2	3	4
0	0	1	2	3	4
1	1	2	1	2	3
2	2	3	2	3	4
3	3	2	3	2	3
4	4	3	2	3	4
5	5	4	3	2	3

The value 3 at (5,4) is
the optimal solution

By tracing back one can
determine which
operations lead to
optimal solution

- Delete x_1 , Delete x_2 and
Insert y_4 Or,

Natural language

- Using natural language for querying is very attractive.
- *Example*: Find all the documents that discuss “ campaign finance reforms, including documents that discuss violations of campaign financing regulations. Do not include documents that discuss campaign contributions by the gun and the tobacco industries”.
- Natural language queries are converted to a formal language for processing against a set of documents.
- Such translation requires ***intelligence*** and is still a challenge

Natural language

- *Pseudo NL processing*: System scans the text and extracts recognized terms and Boolean connectors. The grammaticality of the text is not important.
 - Often used by search engines.
- *Problem*: Recognizing the negation in the search statement (“Do not include...”).
- *Compromise*: Users enter natural language *clauses* connected with Boolean operators.
- In the above example: “campaign finance reforms” **or** “violations of campaign financing regulations” and **not** “campaign contributions by the gun and the tobacco industries”.

Query Operations

Relevance Feedback &
Query Expansion

Problems with Keywords

- May not retrieve relevant documents that include synonymy terms.
 - “restaurant” vs. “café”
 - “Abyssinia” vs. “Ethiopia”
- May retrieve irrelevant documents that include polysomy terms.
 - “Apple” (company vs. fruit vs. sport club)
 - “bit” (unit of data vs. act of eating)
 - “bat” (baseball vs. mammal)

Research area: Intelligent IR

- Take into account the *meaning* of the words used
- Take into account the *order* of words in the query
- Adapt to the user need based on automatic or semi-automatic *feedback*
- *Extend* search with related terms
- Perform automatic *spell checking*

Introduction

- No detailed knowledge of collection and searching environment
 - ⇒ difficult to formulate queries well designed for searching
 - ⇒ need many formulations of queries for effective searching
- ◆ First formulation: often naïve attempt to retrieve relevant information
- ◆ Documents initially retrieved:
 - ⇒ can be examined for relevance information (by the user or automatically by the system) to provide relevance feedback
 - ⇒ improve query formulations for retrieving additional relevant documents (using query reformulation techniques)

Query Reformulation

- Identify terms related to query terms
- Revise query to account for feedback using two basic techniques:
 - **Query Expansion**: Add new terms related to query terms from relevant documents.
 - **Term Reweighting**: modify term weights based on documents relevance for the users query
 - Increase weight of terms in relevant documents and decrease weight of terms in irrelevant documents.
- Several algorithms for query reformulation.

Term Reweighting for Query Reformulation

- Term weight vectors of documents assessed relevant
 - Similarities among themselves
- Term weight vectors of documents assessed non-relevant
 - Dissimilar for those of relevant documents
- Reformulated query:
 - Closer to term weight vectors of relevant documents

Term Reweighting for Query Reformulation: Rochio formula

For query q

- D_r : set of relevant documents among retrieved documents
- D_n : set of non-relevant documents among retrieved documents
- C_r : set of relevant documents among all documents in collection

$$q_{i+1} = \alpha q_i + \frac{\beta}{|D_r|} \sum_{d_j \in D_r} d_j - \frac{\gamma}{|D_n|} \sum_{d_j \in D_n} d_j$$

□ α, β, γ : tuning constants.

- Initial formulation $\alpha=1$
- Usually information in relevant documents is more important than in non-relevant documents ($\gamma \ll \beta$)

Term Reweighting for Query Reformulation: Ide formula

- Initial formulation $\alpha = \beta = \gamma = 1$
 - Same comments as for the Rochio formula

$$q_{i+1} = \alpha q_i + \beta \sum_{d_j \in Dr} d_j - \gamma \sum_{d_j \in Dn} d_j$$

- *Both Ide and Rochio: no optimal criterion*
- E.g. you are given a query vector $q_i = (2, 3, 1, 2, 5)$; and documents; $d_1(3, 3, 2, 0, 9)$; $d_2(2, 2, 1, 0, 12)$; $d_3(3, 2, 1, 0, 9)$; $d_4(1, 0, 0, 7, 2)$; $d_5(0, 1, 0, 8, 3)$
 - 3 documents identified as relevant by a user, (i.e. d_1 - d_3); and 2 documents as irrelevant. Compute the modified query using standard Rochio equation with $\alpha = \beta = \gamma = 1$

Approaches for Query Operations

➤ Users relevance feedback

- Approaches based on feedback from users about relevance of documents retrieved

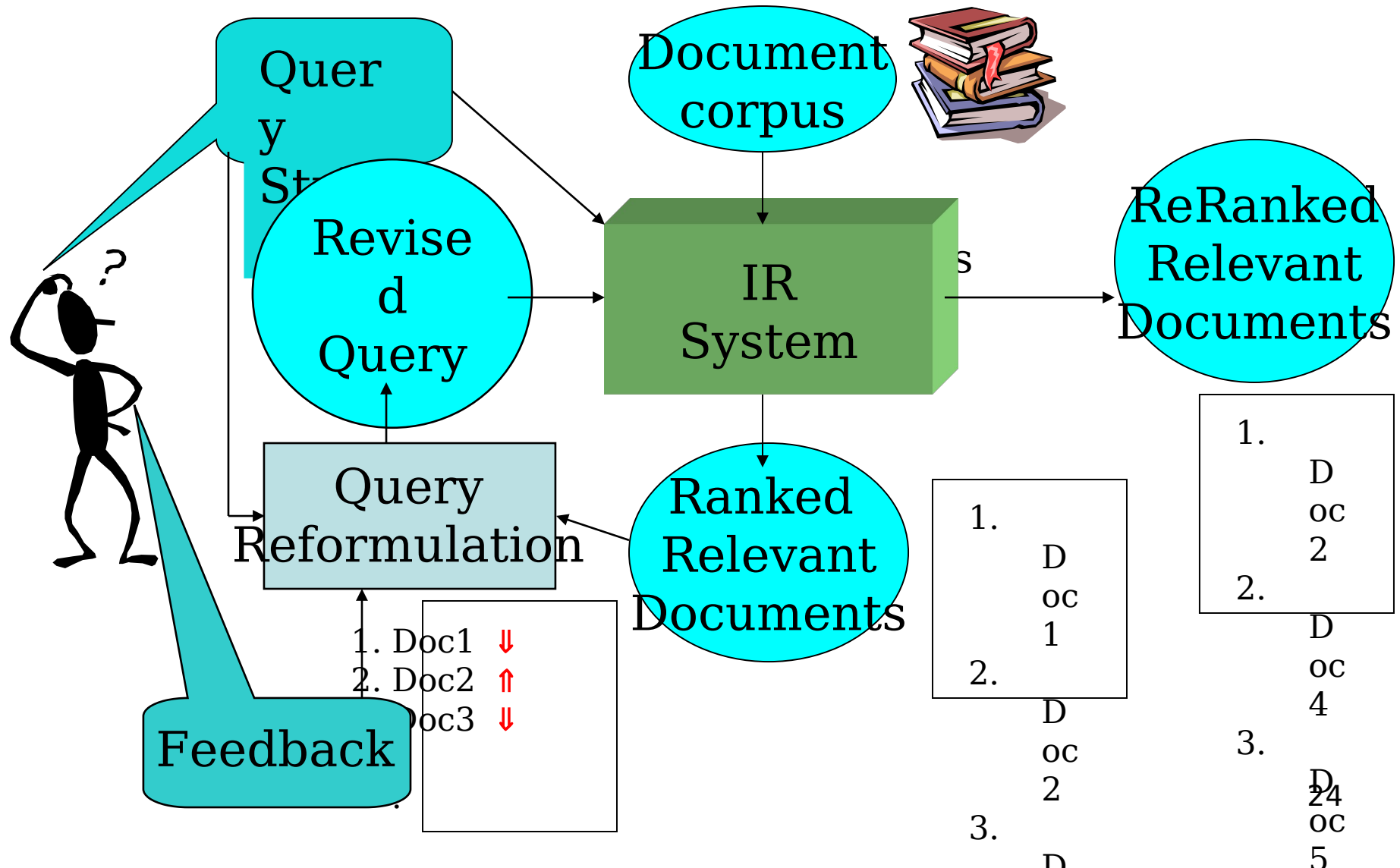
➤ Pseudo-relevance feedback

- Approaches based on information derived from set of initially retrieved documents (local set of documents), which is called Local Analysis
- Approaches based on global information derived from document collection, which is called Global Analysis

Users Relevance Feedback

- Most popular query reformulation strategy
- Cycle:
 - User presented with list of retrieved documents
 - User marks those which are relevant
 - In practice: top 10-20 ranked documents are examined
 - Select **important terms from documents** assessed relevant by users
 - Enhance importance of these terms in a new query
- Expected:
 - New query moves towards relevant documents and away from non-relevant documents

Relevance Feedback Architecture



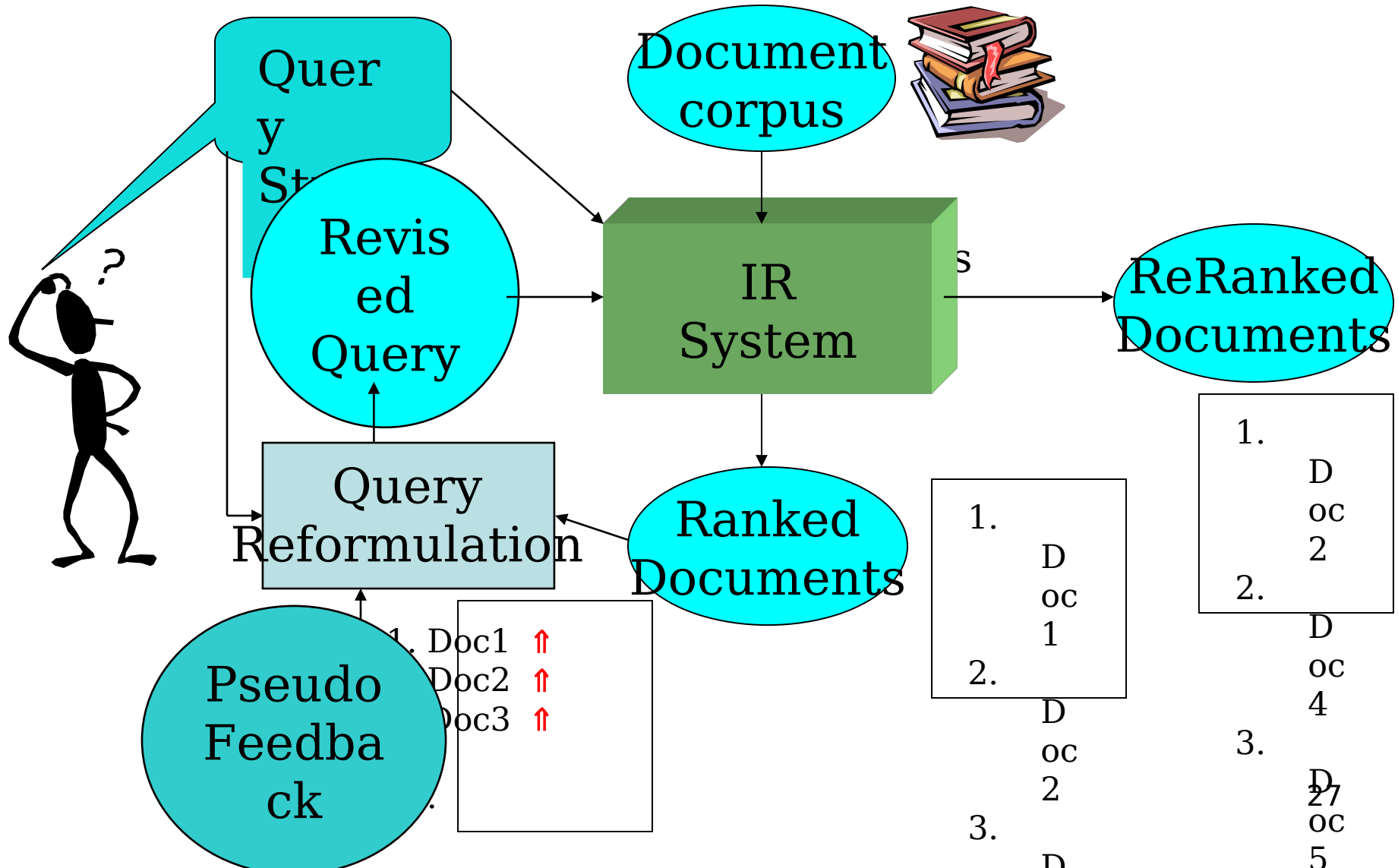
Relevance Feedback

- After initial searching results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
- Use this feedback information to reformulate the query.
- Produce new results based on reformulated query.
- Allows more interactive, multi-pass process.

Pseudo Relevance Feedback

- Use relevance feedback methods without explicit user input.
 - Obtain relevance feedback automatically
 - Identify terms related to query terms (e.g. synonyms, stemming variations, terms close to query terms in text)
- Just **assume** the top m retrieved documents are relevant, and use them to reformulate the query.
- Allows for query expansion that includes terms that are correlated with the query terms.
- Two strategies
 - Local strategies
 - Global strategies

Pseudo Feedback Architecture



Local Analysis

- Examine only documents retrieved automatically for query to determine query expansion
- At query time, dynamically determine similar terms based on analysis of top-ranked retrieved documents.
- Base correlation analysis on only the “local” set of retrieved documents for a specific query.
- Avoids ambiguity by determining similar (correlated) terms only within relevant documents.
 - “Apple computer” → “Apple computer Powerbook laptop”

Global analysis

- Expand query using information from whole set of documents in collection
 - Determine term similarity through a pre-computed statistical analysis of the complete corpus.
- Thesaurus-like structure using all documents
 - Approach to automatically built thesaurus
 - (e.g. similarity thesaurus based on co-occurrence frequency)
 - Approach to select terms for query expansion
- A thesaurus provides information on synonyms and semantically related words and phrases.
- Example:
physician
 similar/synonymous: doctor, medical, MD
 related: general practitioner, surgeon

Thesaurus-based Query Expansion

- For each term, t , in a query, expand the query with synonyms (similar) and related words of t from the thesaurus.
- May weight added to terms less than original query terms.
 - Generally increases recall.
- May significantly decrease precision, particularly with ambiguous terms.
 - “interest rate” → “interest rate fascinate evaluate”

Term Co-occurrence-based Query Expansion

- Compute association matrices which quantify term correlations in terms of how frequently they co-occur.
 - Expand queries with statistically most similar terms.
 - **Synonymy association**: terms that frequently co-occur inside local set of documents

	W_1	W_2	W_3		W_n	Association Matrix
W_1	C_{11}	C_{12}	C_{13}		C_{1n}	
W_2	C_{21}					
.	.					
.	.					
W_n	C_{n1}					

C_{ij} : Correlation factor
 between term i and term j
 tf : Frequency of term i and
 term j in document d

$$C_{i,j} = \sum_{d \in D} tf(t_i, d) \times tf(t_j, d)$$

Normalized Association Matrix

- Frequency based correlation factor favors more frequent terms.
- Normalize association scores:
$$S_{ij} = \frac{C_{ij}}{C_{ii} + C_{jj} - C_{ij}}$$
- Normalized score is $0 \leq S_{ij} \leq 1$
 - 1 if two terms have the same frequency in all documents.
 - 0 if two terms have no correlation
- Term Association
 - For term t_i , take the n largest values $S_{i,j}$
 - The resulting terms t_j form cluster for t_i
- Query q
 - Finding clusters for the $|q|$ query terms
 - Keep clusters small
 - Expand original query

Example

- For the query “**information extraction**”; let say the following documents are retrieved (which contains the words listed below):
 - Doc 1: data, data, information, budget, retrieval, information, budget, retrieval
 - Doc 2: extraction, retrieval, extraction, information, information, data
 - Doc 3: data, retrieving, budget, budget, data, information, budget, retrieval, information
 - Doc 4: information, Internet
- Using term association expand the given query with frequently co-occurring terms

Global vs. Local Analysis

- Global analysis requires intensive term correlation computation only once at system development time.
 - Local analysis requires intensive term correlation computation for every query at run time (although number of terms and documents is less than in global analysis).
- But local analysis gives better results.
 - Term ambiguity may introduce irrelevant statistically correlated terms during global analysis.
 - “Apple computer” → “Apple red fruit computer”

Query Expansion

Conclusions

- Expansion of queries with related terms can improve performance, particularly recall.
- However, must select similar terms very carefully to avoid problems, such as loss of precision.

Programming Assignment (due date: _____)

- Select Local Language that interest you and design an IR system for document collection written in that language.
- Form a group of not more than three members

1. Construct indexing structure

Given text document collection generate index terms and organize them using inverted file indexing, include TF, DF & CF for each index term and position/location information of terms in each document.

2. Develop Vector space retrieval model

Using Vector space model generate any two queries (with two or more words each) and retrieve relevant documents in ranking order.