model itself, novelty is not granted. Instead, the main contribution of the model is the establishment of a formal framework in which dependencies among index terms (induced by co-occurrence patterns inside documents) can be nicely represented.

The usage of index term dependencies to improve retrieval performance continues to be a controversial issue. In fact, despite the introduction in the 1980s of more effective algorithms for incorporating term dependencies (see Chapter 5), there is no consensus that incorporation of term dependencies in the model yields effective improvement with general collections. Thus, it is not clear that the framework of the generalized vector model provides a clear advantage in practical situations. Further, the generalized vector model is more complex and computationally more expensive than the classic vector model.

To determine the index term vector $\vec{k}_i$ associated with the index term k_i, we simply sum up the vectors for all minterms m_r in which the term k_i is in state 1 and normalize. Thus,

$$\vec{k}_i = \frac{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}\ \vec{m}_r}{\sqrt{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}^2}} \qquad (2.5)$$

$$c_{i,r} = \sum_{d_j \mid g_l(\vec{d}_j)=g_l(m_r)\ for\ all\ l} w_{i,j}$$

These equations provide a general definition for the index term vector $\vec{k}_i$ in terms of the $\vec{m}_r$ vectors. The term vector $\vec{k}_i$ collects all the $\vec{m}_r$ vectors in which the index term k_i is in state 1. For each $\vec{m}_r$ vector, a correlation factor $c_{i,r}$ is defined. Such a correlation factor sums up the weights $w_{i,j}$ associated with the index term k_i and each document d_j whose term occurrence pattern coincides exactly with that of the minterm m_r. Thus, a minterm is of interest (in which case it is said to be *active*) only if there is at least one document in the collection which matches its term occurrence pattern. This implies that no more than N minterms can be active, where N is the number of documents in the collection. Therefore, the ranking computation does not depend on an exponential number of minterms as equation 2.5 seems to suggest.

Notice that the internal product $\vec{k}_i \bullet \vec{k}_j$ can now be used to quantify a degree of correlation between the index terms k_i and k_j. For instance,

$$\vec{k}_i \bullet \vec{k}_j = \sum_{\forall r \mid g_i(m_r)=1 \wedge g_j(m_r)=1} c_{i,r} \times c_{j,r}$$

which, as later discussed in Chapter 5, is a good technique for quantifying index term correlations.

In the classic vector model, a document d_j and a user query q are expressed by $\vec{d}_j = \sum_{\forall i} w_{i,j}\ \vec{k}_i$ and $\vec{q} = \sum_{\forall i} w_{i,q}\ \vec{k}_i$, respectively. In the generalized vector space model, these representations can be directly translated to the space of minterm vectors $\vec{m}_r$ by applying equation 2.5. The resultant $\vec{d}_j$ and $\vec{q}$ vectors

are then used for computing the ranking through a standard cosine similarity function.

The ranking that results from the generalized vector space model combines the standard $w_{i,j}$ term-document weights with the correlation factors $c_{i,r}$. However, since the usage of term-term correlations does not necessarily yield improved retrieval performance, it is not clear in which situations the generalized model outperforms the classic vector model. Furthermore, the cost of computing the ranking in the generalized model can be fairly high with large collections because, in this case, the number of active minterms (i.e., those which have to be considered for computing the $\vec{k}_i$ vectors) might be proportional to the number of documents in the collection. Despite these drawbacks, the generalized vector model does introduce new ideas which are of importance from a theoretical point of view.

2.7.2 Latent Semantic Indexing Model

As discussed earlier, summarizing the contents of documents and queries through a set of index terms can lead to poor retrieval performance due to two effects. First, many unrelated documents might be included in the answer set. Second, relevant documents which are not indexed by any of the query keywords are not retrieved. The main reason for these two effects is the inherent vagueness associated with a retrieval process which is based on keyword sets.

The ideas in a text are more related to the concepts described in it than to the index terms used in its description. Thus, the process of matching documents to a given query could be based on concept matching instead of index term matching. This would allow the retrieval of documents even when they are not indexed by query index terms. For instance, a document could be retrieved because it shares concepts with another document which is relevant to the given query. Latent semantic indexing is an approach introduced in 1988 which addresses these issues (for clustering-based approaches which also address these issues, see Chapter 5).

The main idea in the *latent semantic indexing model* [287] is to map each document and query vector into a lower dimensional space which is associated with concepts. This is accomplished by mapping the index term vectors into this lower dimensional space. The claim is that retrieval in the reduced space may be superior to retrieval in the space of index terms. Before proceeding, let us define basic terminology.

Definition *As before, let t be the number of index terms in the collection and N be the total number of documents. Define $\vec{M}=(M_{ij})$ as a term-document association matrix with t rows and N columns. To each element M_{ij} of this matrix is assigned a weight $w_{i,j}$ associated with the term-document pair $[k_i, d_j]$. This $w_{i,j}$ weight could be generated using the tf-idf weighting technique common in the classic vector space model.*

Latent semantic indexing proposes to decompose the $\vec{M}$ association matrix in three components using singular value decomposition as follows.

$$\vec{M} = \vec{K}\vec{S}\vec{D}^t$$

The matrix $\vec{K}$ is the matrix of eigenvectors derived from the term-to-term correlation matrix given by $\vec{M}\vec{M}^t$ (see Chapter 5). The matrix $\vec{D}^t$ is the matrix of eigenvectors derived from the transpose of the document-to-document matrix given by $\vec{M}^t\vec{M}$. The matrix $\vec{S}$ is an $r \times r$ diagonal matrix of singular values where $r = min(t, N)$ is the *rank* of $\vec{M}$.

Consider now that only the s largest singular values of $\vec{S}$ are kept along with their corresponding columns in $\vec{K}$ and $\vec{D}^t$ (i.e., the remaining singular values of $\vec{S}$ are deleted). The resultant $\vec{M}_s$ matrix is the matrix of rank s which is closest to the original matrix $\vec{M}$ in the least square sense. This matrix is given by

$$\vec{M}_s = \vec{K}_s\vec{S}_s\vec{D}_s^t$$

where s, $s < r$, is the dimensionality of a reduced concept space. The selection of a value for s attempts to balance two opposing effects. First, s should be large enough to allow fitting all the structure in the real data. Second, s should be small enough to allow filtering out all the non-relevant representational details (which are present in the conventional index-term based representation).

The relationship between any two documents in the reduced space of dimensionality s can be obtained from the $\vec{M}_s^t\vec{M}_s$ matrix given by

$$\begin{aligned}
\vec{M}_s^t\vec{M}_s &= (\vec{K}_s\vec{S}_s\vec{D}_s^t)^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{K}_s^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{S}_s\vec{D}_s^t \\
&= (\vec{D}_s\vec{S}_s)(\vec{D}_s\vec{S}_s)^t
\end{aligned}$$

In the above matrix, the (i, j) element quantifies the relationship between documents d_i and d_j.

To rank documents with regard to a given user query, we simply model the query as a *pseudo-document* in the original $\vec{M}$ term-document matrix. Assume the query is modeled as the document with number 0. Then, the first row in the matrix $\vec{M}_s^t\vec{M}_s$ provides the ranks of all documents with respect to this query.

Since the matrices used in the latent semantic indexing model are of rank s, $s << t$, and $s << N$, they form an efficient indexing scheme for the documents in the collection. Further, they provide for elimination of noise (present in index term-based representations) and removal of redundancy.

The latent semantic indexing model introduces an interesting conceptualization of the information retrieval problem based on the theory of singular value decomposition. Thus, it has its value as a new theoretical framework. Whether it is superior in practical situations with general collections remains to be verified.

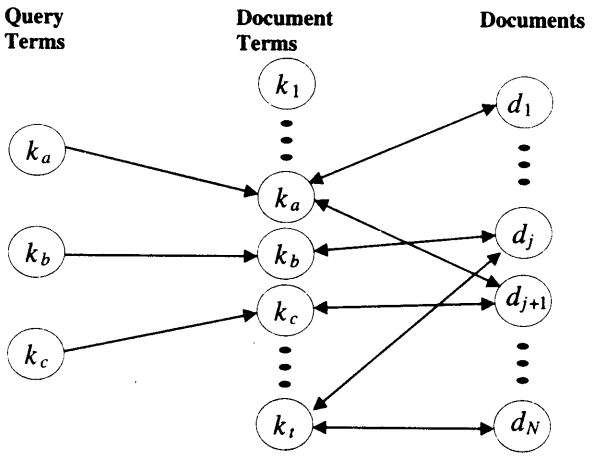

Figure 2.7 A neural network model for information retrieval.

2.7.3 Neural Network Model

In an information retrieval system, document vectors are compared with query vectors for the computation of a ranking. Thus, index terms in documents and queries have to be matched and weighted for computing this ranking. Since neural networks are known to be good pattern matchers, it is natural to consider their usage as an alternative model for information retrieval.

It is now well established that our brain is composed of billions of neurons. Each *neuron* can be viewed as a basic processing unit which, when stimulated by input signals, might emit output signals as a reactive action. The signals emitted by a neuron are fed into other neurons (through synaptic connections) which can themselves emit new output signals. This process might repeat itself through several layers of neurons and is usually referred to as a *spread activation* process. As a result, input information is processed (i.e., analyzed and interpreted) which might lead the brain to command physical reactions (e.g., motor actions) in response.

A *neural network* is an oversimplified graph representation of the mesh of interconnected neurons in a human brain. The nodes in this graph are the processing units while the edges play the role of the synaptic connections. To simulate the fact that the strength of a synaptic connection in the human brain changes over time, a *weight* is assigned to each edge of our neural network. At each instant, the state of a node is defined by its *activation level* (which is a function of its initial state and of the signals it receives as input). Depending on its activation level, a node A might send a signal to a neighbor node B. The strength of this signal at the node B depends on the weight associated with the edge between the nodes A and B.

A neural network for information retrieval can be defined as illustrated in Figure 2.7. The model depicted here is based on the work in [815]. We first

observe that the neural network in Figure 2.7 is composed of three layers: one for the query terms, one for the document terms, and a third one for the documents themselves. Observe the similarity between the topology of this neural network and the topology of the inference and belief networks depicted in Figures 2.9 and 2.10. Here, however, the query term nodes are the ones which initiate the inference process by sending signals to the document term nodes. Following that, the document term nodes might themselves generate signals to the document nodes. This completes a first phase in which a signal travels from the query term nodes to the document nodes (i.e., from the left to the right in Figure 2.7).

The neural network, however, does not stop after the first phase of signal propagation. In fact, the document nodes in their turn might generate new signals which are directed back to the document term nodes (this is the reason for the bidirectional edges between document term nodes and document nodes). Upon receiving this stimulus, the document term nodes might again fire new signals directed to the document nodes, repeating the process. The signals become weaker at each iteration and the spread activation process eventually halts. This process might activate a document d_l even when such a document does not contain any query terms. Thus, the whole process can be interpreted as the activation of a built-in thesaurus.

To the query term nodes is assigned an initial (and fixed) activation level equal to 1 (the maximum). The query term nodes then send signals to the document term nodes which are attenuated by normalized query term weights $\overline{w}_{i,q}$. For a vector-based ranking, these normalized weights can be derived from the weights $w_{i,q}$ defined for the vector model by equation 2.4. For instance,

$$\overline{w}_{i,q} = \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

where the normalization is done using the norm of the query vector.

Once the signals reach the document term nodes, these might send new signals out directed towards the document nodes. These signals are attenuated by normalized document term weights $\overline{w}_{i,j}$ derived from the weights $w_{i,j}$ defined for the vector model by equation 2.3. For instance,

$$\overline{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

where the normalization is done using the norm of the document vector.

The signals which reach a document node are summed up. Thus, after the first round of signal propagation, the activation level of the document node associated to the document d_j is given by

$$\sum_{i=1}^{t} \overline{w}_{i,q}\, \overline{w}_{i,j} = \frac{\sum_{i=1}^{t} w_{i,q}\, w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

which is exactly the ranking provided by the classic vector model.

To improve the retrieval performance, the network continues with the spreading activation process after the first round of propagation. This modifies the initial vector ranking in a process analogous to a user relevance feedback cycle (see Chapter 5). To make the process more effective, a minimum activation threshold might be defined such that document nodes below this threshold send no signals out. Details can be found in [815].

There is no conclusive evidence that a neural network provides superior retrieval performance with general collections. In fact, the model has not been tested extensively with large document collections. However, a neural network does present an alternative modeling paradigm. Further, it naturally allows retrieving documents which are not initially related to the query terms — an appealing functionality.

2.8 Alternative Probabilistic Models

One alternative which has always been considered naturally appealing for quantifying document relevance is the usage of probability theory and its main streams. One such stream which is gaining increased attention concerns the *Bayesian belief networks* which we now discuss.

Bayesian (belief) networks are useful because they provide a clean formalism for combining distinct sources of evidence (past queries, past feedback cycles, and distinct query formulations) in support of the rank for a given document. This combination of distinct evidential sources can be used to improve retrieval performance (i.e., to improve the 'quality' of the ranked list of retrieved documents) as has been demonstrated in the work of Turtle and Croft [771].

In this chapter we discuss two models for information retrieval based on Bayesian networks. The first model is called *inference network* and provides the theoretical basis for the retrieval engine in the Inquery system [122]. Its success has attracted attention to the use of Bayesian networks with information retrieval systems. The second model is called *belief network* and generalizes the first model. At the end, we briefly compare the two models.

Our discussion below uses a style which is quite distinct from that employed by Turtle and Croft in their original writings. Particularly, we pay more attention to probabilistic argumentation during the development of the model. We make a conscious effort of consistently going back to the Bayesian formalism for motivating the major design decisions. It is our view that such an explanation strategy allows for a more precise argumentation which facilitates the task of grasping the subtleties involved.

Before proceeding, we briefly introduce Bayesian networks.

2.8.1 Bayesian Networks

Bayesian networks [630] are directed acyclic graphs (DAGs) in which the nodes represent random variables, the arcs portray causal relationships between these

variables, and the strengths of these causal influences are expressed by conditional probabilities. The *parents* of a node (which is then considered as a *child* node) are those judged to be direct *causes* for it. This causal relationship is represented in the DAG by a link directed from each parent node to the child node. The *roots* of the network are the nodes without parents.

Let x_i be a node in a Bayesian network G and Γ_{x_i} be the set of parent nodes of x_i. The influence of Γ_{x_i} on x_i can be specified by any set of functions $F_i(x_i, \Gamma_{x_i})$ that satisfy

$$\begin{aligned} \sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) &= 1 \\ 0 \leq F_i(x_i, \Gamma_{x_i}) &\leq 1 \end{aligned}$$

where x_i also refers to the states of the random variable associated to the node x_i. This specification is complete and consistent because the product $\prod_{\forall i} F_i(x_i, \Gamma_{x_i})$ constitutes a joint probability distribution for the nodes in G.

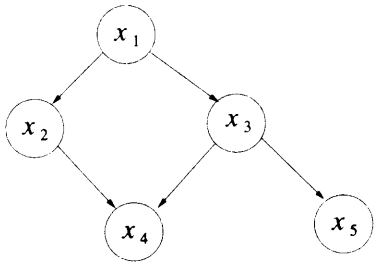

Figure 2.8 An example of a Bayesian network.

Figure 2.8 illustrates a Bayesian network for a joint probability distribution $P(x_1, x_2, x_3, x_4, x_5)$. In this case, the dependencies declared in the network allow the natural expression of the joint probability distribution in terms of local conditional probabilities (a key advantage of Bayesian networks) as follows.

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2|x_1)P(x_3|x_1)P(x_4|x_2, x_3)P(x_5|x_3)$$

The probability $P(x_1)$ is called the *prior* probability for the network and can be used to model previous knowledge about the semantics of the application.

2.8.2 Inference Network Model

The two most traditional schools of thought in probability are based on the *frequentist* view and the *epistemological* view. The frequentist view takes probability as a statistical notion related to the laws of chance. The epistemological

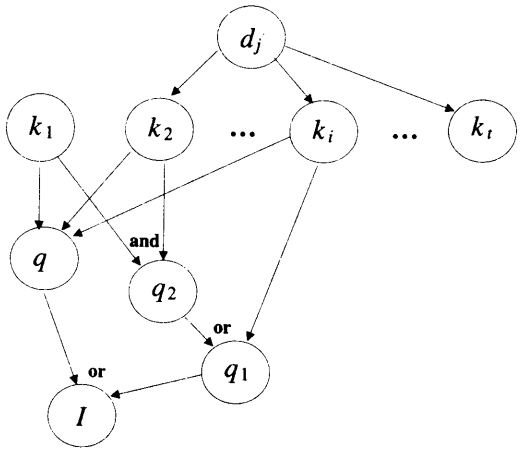

Figure 2.9 Basic inference network model.

view interprets probability as a degree of belief whose specification might be devoid of statistical experimentation. This second viewpoint is important because we frequently refer to probabilities in our daily lives without a clear definition of the statistical experiment which yielded those probabilities.

The inference network model [772, 771] takes an epistemological view of the information retrieval problem. It associates random variables with the index terms, the documents, and the user queries. A random variable associated with a document d_j represents the event of observing that document (i.e., the model assumes that documents are being observed in the search for relevant documents). The observation of the document d_j asserts a belief upon the random variables associated with its index terms. Thus, observation of a document is the *cause* for an increased belief in the variables associated with its index terms.

Index term and document variables are represented as nodes in the network. Edges are directed from a document node to its term nodes to indicate that observation of the document yields improved belief on its term nodes.

The random variable associated with the user query models the event that the information request specified by the query has been met. This random variable is also represented by a node in the network. The belief in this (query) node is a function of the beliefs in the nodes associated with the query terms. Thus, edges are directed from the index term nodes to the query node. Figure 2.9 illustrates an inference network for information retrieval. The document d_j has k_2, k_i, and k_t as its index terms. This is modeled by directing the edges from the node d_j to the nodes k_2, k_i, and k_t. The query q is composed of the index terms k_1, k_2, and k_i. This is modeled by directing the edges from the nodes k_1, k_2, and k_i to the node q. Notice that Figure 2.9 also includes three extra nodes: q_2, q_1, and I. The nodes q_2 and q_1 are used to model an (alternative) Boolean formulation q_1 for the query q (in this case, $q_1 = (k_1 \wedge k_2) \vee k_i$). When such

(additional) information is available, the user information need I is supported by both q and q_1.

In what follows, we concentrate our attention on the support provided to the query node q by the observation of a document d_j. Later on, we discuss the impact of considering multiple query representations for an information need I. This is important because, as Turtle and Croft have demonstrated, a keyword-based query formulation (such as q) can be combined with a Boolean-like query formulation (such as q_1) to yield improved retrieval performance for the same information need.

The complete inference network model also includes text nodes and query concept nodes but the model discussed above summarizes the essence of the approach.

A simplifying assumption is made which states that all random variables in the network are binary. This seems arbitrary but it does simplify the modeling task and is general enough to capture all the important relationships in the information retrieval problem.

Definition *Let $\vec{k}$ be a t-dimensional vector defined by $\vec{k} = (k_1, k_2, \ldots, k_t)$ where $k_1, k_2, \ldots, k_t$ are binary random variables i.e., $k_i \in \{0, 1\}$. These variables define the 2^t possible states for $\vec{k}$. Further, let d_j be a binary random variable associated with a document d_j and let q be a binary random variable associated with the user query.*

Notice that q is used to refer to the query, to the random variable associated with it, and to the respective node in the network. This is also the case for d_j and for each index term k_i. We allow this overloading in syntax because it should always be clear whether we are referring to either the query or to its associated random variable.

The ranking of a document d_j with respect to a query q is a measure of how much evidential support the observation of d_j provides to the query q. In an inference network, the ranking of a document d_j is computed as $P(q \wedge d_j)$ where q and d_j are short representations for $q = 1$ and $d_j = 1$, respectively. In general, such a ranking is given by

$$
\begin{aligned}
P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q \wedge d_j | \vec{k}) \times P(\vec{k}) \\
&= \sum_{\forall \vec{k}} P(q \wedge d_j \wedge \vec{k}) \\
&= \sum_{\forall \vec{k}} P(q | d_j \times \vec{k}) \times P(d_j \times \vec{k}) \\
&= \sum_{\forall \vec{k}} P(q | \vec{k}) \times P(\vec{k} | d_j) \times P(d_j) \qquad \textbf{(2.6)} \\
P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j)
\end{aligned}
$$

which is obtained by basic conditioning and the application of Bayes' rule. Notice that $P(q|d_j \times \vec{k}) = P(q|\vec{k})$ because the k_i nodes separate the query node q from the document node d_j. Also, the notation $\overline{q \wedge d_j}$ is a short representation for $\neg(q \wedge d_j)$.

The instantiation of a document node d_j (i.e., the observation of the document) separates its children index term nodes making them mutually independent (see Bayesian theory for details). Thus, the degree of belief asserted to each index term node k_i by instantiating the document node d_j can be computed separately. This implies that $P(\vec{k}|d_j)$ can be computed in product form which yields (from equation 2.6),

$$
\begin{aligned}
P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q|\vec{k}) \times \\
&\left(\prod_{\forall i | g_i(\vec{k})=1} P(k_i|d_j) \times \prod_{\forall i | g_i(\vec{k})=0} P(\overline{k}_i|d_j) \right) \times P(d_j) \qquad (2.7) \\
P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j)
\end{aligned}
$$

where $P(\overline{k}_i|d_j) = 1 - P(k_i|d_j)$. Through proper specification of the probabilities $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$, we can make the inference network cover a wide range of useful information retrieval ranking strategies. Later on, we discuss how to use an inference network to subsume the Boolean model and tf-idf ranking schemes. Let us first cover the specification of the $P(d_j)$ probabilities.

Prior Probabilities for Inference Networks

Since the document nodes are the root nodes in an inference network, they receive a *prior probability distribution* which is of our choosing. This prior probability reflects the probability associated to the event of observing a given document d_j (to simplify matters, a single document node is observed at a time). Since we have no prior preferences for any document in particular, we usually adopt a prior probability distribution which is uniform. For instance, in the original work on inference networks [772, 771], the probability of observing a document d_j is set to $1/N$ where N is the total number of documents in the system. Thus,

$$
\begin{aligned}
P(d_j) &= \frac{1}{N} \\
P(\overline{d}_j) &= 1 - \frac{1}{N}
\end{aligned}
$$

The choice of the value $1/N$ for the prior probability $P(d_j)$ is a simple and natural specification given that our collection is composed of N documents. However, other specifications for $P(d_j)$ might also be interesting. For instance, in the cosine formula of the vector model, the contribution of an index term to

the rank of the document d_j is inversely proportional to the norm of the vector $\vec{d}_j$. The larger the norm of the document vector, the smaller is the relative contribution of its index terms to the document final rank. This effect can be taken into account through proper specification of the prior probabilities $P(d_j)$ as follows.

$$\begin{aligned} P(d_j) &= \frac{1}{|\vec{d}_j|} \\ P(\overline{d}_j) &= 1 - P(d_j) \end{aligned}$$

where $|\vec{d}_j|$ stands for the norm of the vector $\vec{d}_j$. Therefore, in this case, the larger the norm of a document vector, the smaller its associated prior probability. Such specification reflects a prior knowledge that we have about the behavior of vector-based ranking strategies (which normalize the ranking in the document space). As commanded by Bayesian postulates, previous knowledge of the application domain should be asserted in the specification of the priors in the network, as we have just done.

Inference Network for the Boolean Model

Here we demonstrate how an inference network can be tuned to subsume the Boolean model. First, for the Boolean model, the prior probabilities $P(d_j)$ are all set to $1/N$ because the Boolean model makes no prior distinction on documents. Thus,

$$\begin{aligned} P(d_j) &= \frac{1}{N} \\ P(\overline{d}_j) &= 1 - P(d_j) \end{aligned}$$

Regarding the conditional probabilities $P(k_i|d_j)$ and $P(q|\vec{k})$, the specification is as follows.

$$\begin{aligned} P(k_i|d_j) &= \begin{cases} 1 & \text{if } g_i(d_j) = 1 \\ 0 & \text{otherwise} \end{cases} \\ P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j) \end{aligned}$$

which basically states that, when the document d_j is being observed, only the nodes associated with the index terms of the document d_j are *active* (i.e., have an induced probability greater than 0). For instance, observation of a document node d_j whose term vector is composed of exactly the index terms k_2, k_i, and k_t (see Figure 2.9) activates the index term nodes $\{k_2,k_i,k_t\}$ and no others.

Once the beliefs in the index term nodes have been computed, we can use them to compute the evidential support they provide to the user query q as

follows.

$$
\begin{aligned}
P(q|\vec{k}) &= \begin{cases} 1 & \text{if } \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i},\ g_i(\vec{k}) = g_i(\vec{q}_{cc})) \\ 0 & \text{otherwise} \end{cases} \\
P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k})
\end{aligned}
$$

where $\vec{q}_{cc}$ and $\vec{q}_{dnf}$ are as defined for the classic Boolean model. The above equation basically states that one of the conjunctive components of the user query (expressed in disjunctive normal form) must be matched by the set of active terms in $\vec{k}$ (in this case, those activated by the document observed) exactly.

Substituting the above definitions for $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$ into equation 2.7, it can be easily shown that the set of documents retrieved is exactly the set of documents returned by the Boolean model as defined in section 2.5.2. Thus, an inference network can be used to subsume the Boolean model without difficulties.

Inference Network for tf-idf Ranking Strategies

For tf-idf ranking strategies (i.e., those related to the vector model), we adopt prior probabilities which reflect our prior knowledge of the importance of document normalization. Thus, we set the prior $P(d_j)$ to $1/|\vec{d}_j|$ as follows.

$$
\begin{aligned}
P(d_j) &= \frac{1}{|\vec{d}_j|} \\
P(\overline{d}_j) &= 1 - P(d_j)
\end{aligned}
\tag{2.8}
$$

Further, we have to decide where to introduce the tf (term-frequency) and the idf (inverse-document-frequency) factors in the network. For that purpose, we consider that the tf and idf factors are normalized (as in equation 2.1) and that these normalized factors are strictly smaller than 1.

We first focus on capturing the impact of the tf factors in the network. Normalized tf factors are taken into account through the beliefs asserted upon the index term nodes as follows.

$$
\begin{aligned}
P(k_i|d_j) &= f_{i,j} \\
P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j)
\end{aligned}
\tag{2.9}
$$

These equations simply state that, according to the observed document d_j, the relevance of a term k_i is determined by its normalized term-frequency factor.

We are now in a position to consider the influence of idf factors. They are taken into account through the specification of the impact of index term nodes

on the query node. Define a vector $\vec{k}_i$ given by,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\ g_j(\vec{k}) = 0)$$

The vector $\vec{k}_i$ is a reference to the state of the vector $\vec{k}$ in which the node k_i is active and all others are inactive. The motivation is that tf-idf ranking strategies sum up the *individual* contributions of index terms and $\vec{k}_i$ allows us to consider the influence of the term k_i in isolation. We are now ready to define the influence of the index term nodes in the query node q as

$$\begin{aligned} P(q|\vec{k}) &= \begin{cases} idf_i & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{q}) = 1 \\ 0 & \text{if } \vec{k} \neq \vec{k}_i \vee g_i(\vec{q}) = 0 \end{cases} \\ P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k}) \end{aligned} \tag{2.10}$$

where idf_i here is a normalized version of the idf factor defined in equation 2.2.

By applying equations 2.8, 2.9, and 2.10 to equation 2.7, we can then write

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}_i} P(q|\vec{k}_i) \times P(k_i|d_j) \times \left(\prod_{\forall\ l \neq i} P(\overline{k}_l|d_j) \right) \times P(d_j) \\ &= \left(\prod_{\forall\ i} P(\overline{k}_i|d_j) \right) \times P(d_j) \times \sum_{\forall \vec{k}_i} P(k_i|d_j) \times P(q|\vec{k}_i) \times \frac{1}{P(\overline{k}_i|d_j)} \\ &= C_j \times \frac{1}{|\vec{d}_j|} \times \sum_{\forall i | g_i(\vec{d}_j)=1 \wedge g_i(\vec{q})=1} f_{i,j} \times idf_i \times \frac{1}{1 - f_{i,j}} \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

which provides a tf-idf-like ranking. Unfortunately, C_j depends on a product of the various probabilities $P(\overline{k}_i|d_j)$ which vary from document to document and thus the ranking is distinct from the one provided by the vector model. Despite this peculiarity in the tf-idf ranking generated, it has been shown that an inference network is able to provide good retrieval performance with general collections. The reason is that the network allows us to consistently combine evidence from distinct evidential sources to improve the final ranking, as we now discuss.

Combining Evidential Sources

In Figure 2.9, the first query node q is the standard keyword-based query formulation for the user information need I. The second query q_1 is a Boolean-like query formulation for the same information need (i.e., an additional evidential source collected from a specialist). The joint support these two query formulations provide to the information need node I can be modeled through, for

instance, an OR operator (i.e., $I = q \vee q_1$). In this case, the ranking provided by the inference network is computed as,

$$\begin{aligned} P(I \wedge d_j) &= \sum_{\vec{k}} P(I|\vec{k}) \times P(\vec{k}|d_j) \times P(d_j) \\ &= \sum_{\vec{k}} (1 - P(\overline{q}|\vec{k})\ P(\overline{q}_1|\vec{k})) \times P(\vec{k}|d_j) \times P(d_j) \end{aligned}$$

which might yield a retrieval performance which surpasses the retrieval performance obtained with each of the query nodes in isolation as demonstrated in [771].

2.8.3 Belief Network Model

The belief network model, introduced in 1996 by Ribeiro-Neto and Muntz [674], is also based on an epistemological interpretation of probabilities. However, it departs from the inference network model by adopting a clearly defined sample space. As a result, it yields a slightly different network topology which provides a separation between the document and the query portions of the network. This is the main difference between the two models and one which has theoretical implications.

The Probability Space

The probability space adopted was first introduced by Wong and Yao [830] and works as follows. All documents in the collection are indexed by *index terms* and the universe of discourse U is the set K of all index terms.

Definition *The set $K = \{k_1, \ldots, k_t\}$ is the universe of discourse and defines the sample space for the belief network model. Let $u \subset K$ be a subset of K. To each subset u is associated a vector $\vec{k}$ such that $g_i(\vec{k}) = 1 \Longleftrightarrow k_i \in u$.*

The introduction of the vector $\vec{k}$ is useful to keep the notation compatible with the one which has been used throughout this chapter.

Each index term is viewed as an *elementary concept* and K as a concept space. A concept u is a subset of K and might represent a document in the collection or a user query. In a belief network, set relationships are specified using random variables as follows.

Definition *To each index term k_i is associated a binary random variable which is also referred to as k_i. The random variable k_i is set to 1 to indicate that the index k_i is a member of a concept/set represented by $\vec{k}$.*

This association of concepts with subsets is useful because it allows us to express the logical notions of conjunction, disjunction, negation, and implication as the more familiar set-theoretic notions of intersection, union, complementation, and inclusion. Documents and user queries can be defined as concepts in the sample space K as follows.

Definition *A document d_j in the collection is represented as a concept (i.e., a set) composed of the terms which are used to index d_j. Analogously, a user query q is represented as a concept composed of the terms which are used to index q.*

A probability distribution P is defined over K as follows. Let c be a generic concept in the space K representing a document or user query. Then,

$$P(c) = \sum_u P(c|u) \times P(u) \tag{2.11}$$

$$P(u) = \left(\frac{1}{2}\right)^t \tag{2.12}$$

Equation 2.11 defines $P(c)$ as the *degree of coverage* of the space K by c. Such a coverage is computed by contrasting each of the concepts in K with c (through $P(c|u)$) and by summing up the individual contributions. This sum is weighted by the probability $P(u)$ with which u occurs in K. Since at the beginning the system has no knowledge of the probability with which a concept u occurs in the space K, we can assume that each u is equally likely which results in equation 2.12.

Belief Network Model

In the belief network model, the user query q is modeled as a network node to which is associated a binary random variable (as in the inference network model) which is also referred to as q. This variable is set to 1 whenever q completely covers the concept space K. Thus, when we assess $P(q)$ we compute the degree of coverage of the space K by q. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that q completely covers all possible concepts in K?

A document d_j is modeled as a network node with which is associated a binary random variable which is also referred to as d_j. This variable is 1 to indicate that d_j completely covers the concept space K. When we assess $P(d_j)$, we compute the degree of coverage of the space K by d_j. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that d_j completely covers all possible concepts in K?

According to the above formalism, the user query and the documents in the collection are modeled as subsets of index terms. Each of these subsets is interpreted as a *concept* embedded in the concept space K which works as a common *sample space*. Furthermore, user queries and documents are modeled

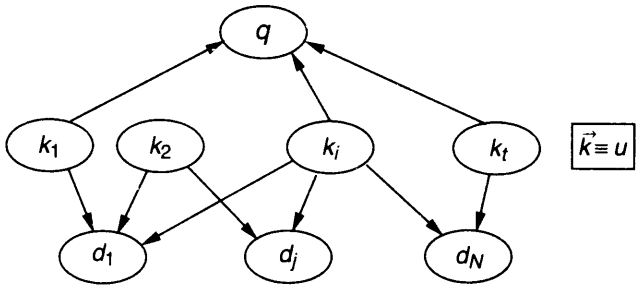

Figure 2.10 Basic belief network model.

identically. This is an important observation because it defines the topology of the belief network.

Figure 2.10 illustrates our belief network model. As in the inference network model, a query q is modeled as a binary random variable which is pointed to by the index term nodes which compose the query concept. Documents are treated analogously to user queries (i.e., both are concepts in the space K). Thus, contrary to the inference network model, a document node is pointed to by the index term nodes which compose the document. This is the topological difference between the two models and one which has more implications than it seems at first glance.

The ranking of a document d_j relative to a given query q is interpreted as a concept matching relationship and reflects the degree of coverage provided to the concept d_j by the concept q.

Assumption In the belief network model, $P(d_j|q)$ is adopted as the rank of the document d_j with respect to the query q.

By the application of Bayes' theorem, we can write $P(d_j|q) = P(d_j \wedge q)/P(q)$. Since $P(q)$ is a constant for all documents in the collection, we can write $P(d_j|q) \sim P(d_j \wedge q)$ i.e., the rank assigned to a document d_j is directly proportional to $P(d_j \wedge q)$. This last probability is computed through the application of equation 2.11 which yields

$$P(d_j|q) \sim \sum_{\forall u} P(d_j \wedge q|u) \times P(u)$$

In the belief network of Figure 2.10, instantiation of the index term variables logically separates the nodes q and d making them mutually independent (i.e., the document and query portions of the network are logically separated by in-

stantiation of the index term nodes). Therefore,

$$P(d_j|q) \sim \sum_{\forall u} P(d_j|u) \times P(q|u) \times P(u)$$

which can be rewritten as

$$P(d_j|q) \sim \sum_{\forall \vec{k}} P(d_j|\vec{k}) \times P(q|\vec{k}) \times P(\vec{k}) \quad \textbf{(2.13)}$$

To complete the belief network we need to specify the conditional probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$. Distinct specifications of these probabilities allow the modeling of different ranking strategies (corresponding to different IR models). We now discuss how to specify these probabilities to subsume the vector model.

For the vector model, the probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$ are specified as follows. Let,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\ g_j(\vec{k}) = 0)$$

as before. Also,

$$P(q|\vec{k}) = \begin{cases} \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(q) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{q}|\vec{k}) = 1 - P(q|\vec{k})$$

Further, define

$$P(d_j|\vec{k}) = \begin{cases} \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{d}_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{d_j}|\vec{k}) = 1 - P(d_j|\vec{k})$$

Then, the ordering of the retrieved documents (i.e., the ranking) defined by $P(d_j|q)$ coincides with the ordering generated by the vector model as specified in section 2.5.3. Thus, the belief network model can be tuned to subsume the vector model which cannot be accomplished with the inference network model.

2.8.4 Comparison of Bayesian Network Models

There is a close resemblance between the belief network model and the inference network model. However, this resemblance hides important differences between the two models. First, the belief network model is based on a set-theoretic view

of the IR ranking problem and adopts a clearly defined sample space. The inference network model takes a purely epistemological view of the IR problem which is more difficult to grasp (because, for instance, the sample space is not clearly defined). Second, the belief network model provides a separation between the document and the query portions of the network which facilitates the modeling of additional evidential sources such as past queries and past relevance information. Third, as a result of this document-query space separation, the belief network model is able to reproduce any ranking strategy generated by the inference network model while the converse is not true.

To see that the belief network ranking subsumes any ranking generated by an inference network, compare equations 2.6 and 2.13. The key distinction is between the terms $P(d_j|\vec{k})$ and $P(\vec{k}|d_j)$. For the latter, instantiation of the document node d_j separates the index term nodes making them mutually independent. Thus, the joint probability $P(\vec{k}|d_j)$ can always be computed as the product of the individual probabilities $P(k_i|d_j)$. However, the computation of $P(d_j|\vec{k})$ might be non-decomposable in a product of term-based probabilities. As a result, $P(d_j|\vec{k})$ can express any probability function defined with $P(\vec{k}|d_j)$ while the converse is not true.

One should not infer from the above comparison that the inference network model is not a good model. On the contrary, it has been shown in the literature that the inference network model allows top retrieval performance to be accomplished with general collections. Further, it is the retrieval model used by the Inquery system. The point of the comparison is that, from a theoretical point of view, the belief network model is more general. Also, it provides a separation between the document space and the query space which simplifies the modeling task.

2.8.5 Computational Costs of Bayesian Networks

In the inference network model, according to equation 2.6, only the states which have a single document active node are considered. Thus, the cost of computing the ranking is linear on the number of documents in the collection. As with conventional collections, index structures such as inverted files (see Chapter 8) are used to restrict the ranking computation to those documents which have terms in common with the query. Thus, the cost of computing an inference network ranking has the same complexity as the cost of computing a vectorial ranking.

In the belief network model, according to equation 2.13, the only states (of the roots nodes) considered (for computing the rank of a document d_j) are the ones in which the active nodes are exactly those associated with the query terms. Thus, again, the cost of computing the ranking is linear on the number of documents in the collection. If index structures are used, the cost of computing a belief network ranking has the same complexity as the cost of computing a vectorial ranking.

Therefore, the Bayesian network models discussed here do not impose significant additional costs for ranking computation. This is so because the networks presented do not include cycles, which implies that belief propagation can be done in a time proportional to the number of nodes in the network.

2.8.6 The Impact of Bayesian Network Models

The classic Boolean model is based on a neat formalism but is not very effective for information retrieval. The classic vector model provides improved answer sets but lacks a more formal framework. Many attempts have been made in the past to combine the best features of each model. The extended Boolean model and the generalized vector space model are two well known examples. These past attempts are grounded in the belief that the combination of selected properties from distinct models is a promising approach towards improved retrieval.

Bayesian network models constitute modern variants of probabilistic reasoning whose major strength (for information retrieval) is a framework which allows the neat combination of distinct evidential sources to support a relevance judgement (i.e., a numerical rank) on a given document. In this regard, belief networks seem more appropriate than previous approaches and more promising. Further, besides allowing the combination of Boolean and vector features, a belief network can be naturally extended to incorporate evidential information derived from past user sessions [674] and feedback cycles [332].

The inference network model has been successfully implemented in the Inquery retrieval system [122] and compares favorably with other retrieval systems. However, despite these promises, whether Bayesian networks will become popular and widely used for information retrieval remains to be seen.

2.9 Structured Text Retrieval Models

Consider a user with a superior visual memory. Such a user might then recall that the specific document he is interested in contains a page in which the string '*atomic holocaust*' appears in italic in the text surrounding a Figure whose label contains the word 'earth.' With a classic information retrieval model, this query could be expressed as ['atomic holocaust' and 'earth'] which retrieves all the documents containing both strings. Clearly, however, this answer contains many more documents than desired by this user. In this particular case, the user would like to express his query through a richer expression such as

same-page (near ('*atomic holocaust*,' Figure (label ('earth'))))

which conveys the details in his visual recollection. Further, the user might be interested in an advanced interface which simplifies the task of specifying this (now complex) query. This example illustrates the appeal of a query language which allows us to combine the specification of strings (or patterns) with the

specification of structural components of the document. Retrieval models which combine information on text content with information on the document structure are called *structured text retrieval* models.

For a query such as the one illustrated above, a structured text retrieval system searches for all the documents which *satisfy* the query. Thus, there is no notion of relevance attached to the retrieval task. In this sense, the current structured text retrieval models are data (instead of information) retrieval models. However, the retrieval system could search for documents which match the query conditions only partially. In this situation, the matching would be approximate and some ranking would have to be used for ordering the approximate answers. Thus, a structured text retrieval algorithm can be seen as an information retrieval algorithm for which the issue of appropriate ranking is not well established. In fact, this is an actual, interesting, and open research problem.

At the end of the 1980s and throughout the 1990s, various structured text retrieval models have appeared in the literature. Usually, the more expressive the model, the less efficient is its query evaluation strategy. Thus, selection of a structured model for a given application must be exercised with care. A good policy is to select the most efficient model which supports the functionality required by the application in view.

Here, we do not survey all the structured text retrieval models. Instead, we briefly discuss the main features of two of them, namely, a model based on *non-overlapping lists* and a model based on *proximal nodes*. These two models should provide a good overview of the main issues and tradeoffs in structured text retrieval.

We use the term *match point* to refer to the position in the text of a sequence of words which matches (or satisfies) the user query. Thus, if the user specifies the simple query ['atomic holocaust in Hiroshima'] and this string appears in three positions in the text of a document d_j, we say that the document d_j contains three match points. Further, we use the term *region* to refer to a contiguous portion of the text and the term *node* to refer to a structural component of the document such as a chapter, a section, a subsection, etc. Thus, a node is a region with predefined topological properties which are known both to the author of the document and to the user who searches the document system.

2.9.1 Model Based on Non-Overlapping Lists

Burkowski [132, 133] proposes to divide the whole text of each document in non-overlapping text regions which are collected in a *list*. Since there are multiple ways to divide a text in non-overlapping regions, multiple lists are generated. For instance, we might have a list of all chapters in the document, a second list of all sections in the document, and a third list of all subsections in the document. These lists are kept as separate and distinct data structures. While the text regions in the same (flat) list have no overlapping, text regions from distinct lists might overlap. Figure 2.11 illustrates four separate lists for the same document.

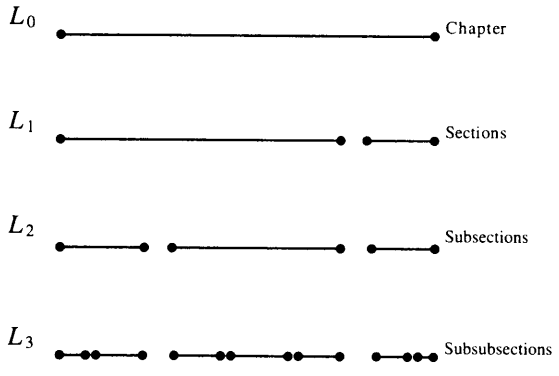

Figure 2.11 Representation of the structure in the text of a document through four separate (flat) indexing lists.

To allow searching for index terms and for text regions, a single inverted file (see Chapter 8 for a definition of inverted files) is built in which each structural component stands as an entry in the index. Associated with each entry, there is a list of text regions as a list of occurrences. Moreover, such a list could be easily merged with the traditional inverted file for the words in the text. Since the text regions are non-overlapping, the types of queries which can be asked are simple: (a) select a region which contains a given word (and does not contain other regions); (b) select a region A which does not contain any other region B (where B belongs to a list distinct from the list for A); (c) select a region not contained within any other region, etc.

2.9.2 Model Based on Proximal Nodes

Navarro and Baeza-Yates [41, 589, 590] propose a model which allows the definition of independent hierarchical (non-flat) indexing structures over the same document text. Each of these indexing structures is a strict hierarchy composed of chapters, sections, paragraphs, pages, and lines which are called *nodes* (see Figure 2.12). To each of these nodes is associated a text region. Further, two distinct hierarchies might refer to overlapping text regions.

Given a user query which refers to distinct hierarchies, the compiled answer is formed by nodes which all come from only one of them. Thus, an answer cannot be composed of nodes which come from two distinct hierarchies (which allows for faster query processing at the expense of less expressiveness). Notice, however, that due to the hierarchical structure, nested text regions (coming from the same hierarchy) are allowed in the answer set.

Figure 2.12 illustrates a hierarchical indexing structure composed of four

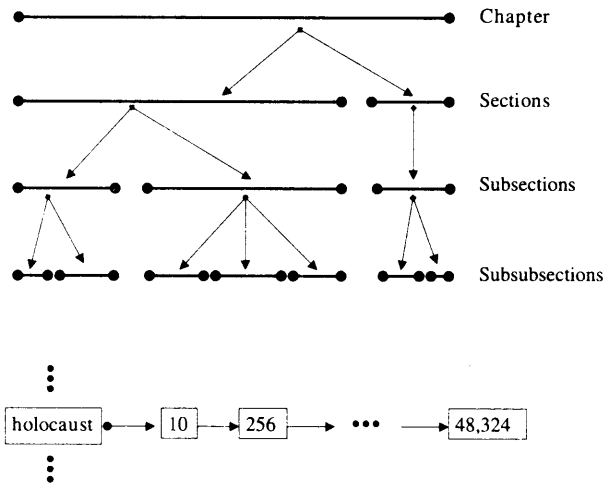

Figure 2.12 Hierarchical index for structural components and flat index for words.

levels (corresponding to a chapter, sections, subsections, and subsubsections of the same document) and an inverted list for the word 'holocaust.' The entries in this inverted list indicate all the positions in the text of the document in which the word 'holocaust' occurs. In the hierarchy, each node indicates the position in the text of its associated structural component (chapter, section, subsection, or subsubsection).

The query language allows the specification of regular expressions (to search for strings), the reference to structural components by name (to search for chapters, for instance), and a combination of these. In this sense, the model can be viewed as a compromise between expressiveness and efficiency. The somewhat limited expressiveness of the query language allows efficient query processing by first searching for the components which match the strings specified in the query and, subsequently, evaluating which of these components satisfy the structural part of the query.

Consider, for instance, the query [(*section) with ('holocaust')] which searches for sections, subsections, or subsubsections which contain the word 'holocaust.' A simple query processing strategy is to traverse the inverted list for the term 'holocaust' and, for each entry in the list (which indicates an occurrence of the term 'holocaust' in the text), search the hierarchical index looking for sections, subsections, and subsubsections containing that occurrence of the term. A more sophisticated query processing strategy is as follows. For the first entry in the list for 'holocaust,' search the hierarchical index as before. This implies traversing down the hierarchy until no more successful matches occur (or the bottom of the hierarchy is reached). Let the last matching structural component be referred to as the innermost matching component. Once this first search is concluded, do not start all over again for the following entry in the

inverted list. Instead, verify whether the innermost matching component also matches the second entry in the list. If it does, we immediately conclude that the larger structural components above it (in the hierarchy) also do. Proceed then to the third entry in the list, and so on. Notice that the query processing is accelerated because only the nearby (or proximal) nodes in the list need to be searched at each time. This is the reason for the label *proximal nodes*.

The model based on proximal nodes allows us to formulate queries which are more complex than those which can be formulated in the model based on non-overlapping lists. To speed up query processing, however, only nearby (proximal) nodes are looked at which imposes restrictions on the answer set retrieved (all nodes must come from the same hierarchy). More complex models for structured retrieval have been proposed in the literature as discussed in [41, 590].

2.10 Models for Browsing

As already observed, the user might not be interested in posing a specific query to the system. Instead, he might be willing to invest some time in exploring the document space looking for interesting references. In this situation, we say that the user is *browsing* the space instead of searching. Both with browsing and searching, the user has goals which he is pursuing. However, in general, the goal of a searching task is clearer in the mind of the user than the goal of a browsing task. As is obvious, this is not a distinction which is valid in all scenarios. But, since it is simple and provides a clear separation between the tasks of searching and browsing, it is adopted here. We distinguish three types of browsing namely, flat, structure guided, and hypertext.

2.10.1 Flat Browsing

The idea here is that the user explores a document space which has a flat organization. For instance, the documents might be represented as dots in a (two-dimensional) plan or as elements in a (single dimension) list. The user then glances here and there looking for information within the documents visited. For instance, he might look for correlations among neighbor documents or for keywords which are of interest to him. Such keywords could then be added to the original query in an attempt to provide better contextualization. This is a process called *relevance feedback* which is discussed in detail in Chapter 5. Also, the user could explore a single document in a flat manner. For example, he could use a browser to look into a Web page, using the arrows and the scroll bar. One disadvantage is that in a given page or screen there may not be any indication about the context where the user is. For example, if he opens a novel at a random page, he might not know in which chapter that page is.

Web search engines such as 'Yahoo!' provide, besides the standard search interface, a hierarchical directory which can be used for browsing (and frequently, for searching). However, the organization is not flat as discussed below.

2.10.2 Structure Guided Browsing

To facilitate the task of browsing, the documents might be organized in a structure such as a directory. Directories are hierarchies of classes which group documents covering related topics. Such hierarchies of classes have been used to classify document collections for many centuries now. Thus, it seems natural to adapt them for use with modern browsing interfaces. In this case, we say that the user performs a structure guided type of browsing. The same idea can be applied to a single document. For example, if we are browsing an electronic book, a first level of content could be the chapters, the second level, all sections, and so on. The last level would be the text itself (flat). A good user interface could go down or up those levels in a focused manner, assisting the user with the task of keeping track of the context.

Besides the structure which directs the browsing task, the interface can also include facilities such as a history map which identifies classes recently visited. This might be quite useful for dealing with very large structures – an issue discussed in Chapters 10 and 13. When searching, the occurrences can also be displayed showing just the structure (for example, using the table of contents). This allows us to see the occurrences in a global context instead of in a page of text that may have no indication of where we are.

2.10.3 The Hypertext Model

One fundamental concept related to the task of writing down text is the notion of *sequencing*. Written text is usually conceived to be read sequentially. The reader should not expect to fully understand the message conveyed by the writer by randomly reading pieces of text here and there. One might rely on the text structure to skip portions of the text but this might result in miscommunication between reader and writer. Thus, a sequenced organizational structure lies underneath most written text. When the reader fails to perceive such a structure and abide by it, he frequently is unable to capture the essence of the writer's message.

Sometimes, however, we are looking for information which is subsumed by the whole text but which cannot be easily captured through sequential reading. For instance, while glancing at a book about the history of the wars fought by man, we might be momentarily interested solely in the regional wars in Europe. We know that this information is in the book, but we might have a hard time finding it because the writer did not organize his writings with this purpose (he might have organized the wars chronologically). In such a situation, a different organization of the text is desired. However, there is no point in rewriting the whole text. Thus, the solution is to define a new organizational structure besides the one already in existence. One way to accomplish such a goal is through the design of a hypertext.

Hypertext Definition and the Navigational Task

A *hypertext* is a high level interactive navigational structure which allows us to browse text non-sequentially on a computer screen. It consists basically of nodes which are correlated by directed links in a graph structure.

To each node is associated a text region which might be a chapter in a book, a section in an article, or a Web page. Two nodes A and B might be connected by a *directed link* l_{AB} which correlates the texts associated with these two nodes. In this case, the reader might move to the node B while reading the text associated with the node A.

In its most conventional form, a hypertext link l_{AB} is attached to a specific string inside the text for node A. Such a string is marked specially (for instance, its characters might appear in a different color or underlined) to indicate the presence of the underlying link. While reading the text, the user might come across a marked string. If the user clicks on that string, the underlying directed link is followed, and a new text region (associated with the node at the destination) is displayed on the screen.

The process of navigating the hypertext can be understood as a traversal of a directed graph. The linked nodes of the graph represent text nodes which are semantically related. While traversing this graph the reader visualizes a flow of information which was conceived by the designer of the hypertext. Consider our previous example regarding a book on the wars fought by man. One might design a hypertext composed of two distinct *webs* (here, a web is simply a connected component formed by a subset of all links in the hypertext). While the first web might be designed to provide access to the local wars fought in Europe in chronological order, the second web might be designed to provide access to the local wars fought by each European country. In this way, the user of this hypertext can access the information according to his particular need.

When the hypertext is large, the user might lose track of the organizational structure of the hypertext. The effect is that the user might start to take bad navigational decisions which might sidetrack him from his main goal (which usually consists of finding a piece of information in the hypertext). When this happens, the user is said to be *lost in hyperspace* [604]. To avoid this problem, it is desirable that the hypertext include a hypertext map which shows where the user is at all times. In its simplest form, this map is a directed graph which displays the current node being visited. Additionally, such a map could include information on the paths the user has traveled so far. This can be used to remind the user of the uselessness of following paths which have been explored already.

While navigating a hypertext, the user is restricted to the intended flow of information previously conceived by the hypertext designer. Thus, the task of designing a hypertext should take into account the needs of its potential users. This implies the execution of a requirement analysis phase before starting the actual implementation of the hypertext. Such a requirement analysis is critically important but is frequently overlooked.

Furthermore, during the hypertext navigation, the user might find it difficult to orient himself. This difficulty arises even in the presence of a guiding

tool such as the hypertext map discussed above. One possible reason is an excessively complex hypertext organization with too many links which allow the user to travel back and forth. To avoid this problem, the hypertext can have a simpler structure which can be quickly remembered by the user at all times. For instance, the hypertext can be organized hierarchically to facilitate the navigational task.

Definition of the structure of the hypertext should be accomplished in a domain modeling phase (done after a requirement analysis phase). Further, after the modeling of the domain, a user interface design should be concluded prior to implementation. Only then, can we say that we have a proper hypertext structure for the application at hand. In the Web, however, pages are usually implemented with no attention paid to requirement analysis, domain modeling, and user interface design. As a result, Web pages are frequently poorly conceived and often fail to provide the user with a proper hypertext structure for assistance with the information seeking task.

With large hypertexts, it might be difficult for the user to position himself in the part of the whole graph which is of most interest to him. To facilitate this initial positioning step, a search based on index terms might be used. In [540], Manber discusses the advantages of this approach.

Hypertexts provided the basis for the conception and design of the hypertext markup language (HTML) and the hypertext transfer protocol (HTTP) which originated the *World Wide Web* (which we simply refer to as the Web). In Chapter 13, we discuss the Web in detail. We briefly discuss some of its features below.

About the Web

When one talks about the Web, the first concept which comes to mind is that of a hypertext. In fact, we frequently think of the Web as a huge distributed hypertext domain. However, the Web is not exactly a proper hypertext because it lacks an underlying data model, it lacks a navigational plan, and it lacks a consistently designed user interface. Each one of the millions of Web page designers devises his own interface with its own peculiar characteristics. Many times we visit a Web site simply looking for a phone number and cannot find it because it is buried in the least expected place of the local hypertext structure. Thus, the Web user has no underlying metaphor to assist him in the search for information of interest.

Instead of saying that the Web is a hypertext, we prefer to say that it is a pool of (partially) interconnected webs. Some of these webs might be characterized as a local hypertext (in the sense that they have an underlying structure which enjoys some consistency) but others might be simply a collection of pages designed separately (for instance, the web of a university department whose professors design their own pages). Despite not being exactly a hypertext, the Web has provided us with a new dimension in communication functionality because it is easily accessible world wide at very low cost. And maybe most important, the Web has no control body setting up regulations and censorship rules. As a

result, for the first time in the history of mankind, any one person can publish his writings through a large medium without being subjected to the filtering of an editorial board.

For a more thorough discussion of these and many other issues related to the Web, the user is referred to Chapter 13.

2.11 Trends and Research Issues

There are three main types of products and systems which can benefit directly from research in models for information retrieval: library systems, specialized retrieval systems, and the Web.

Regarding library systems, there is currently much interest in cognitive and behavioral issues oriented particularly at a better understanding of which criteria the users adopt to judge relevance. From the point of view of the computer scientist, a main question is how this knowledge about the user affects the ranking strategies and the user interface implemented by the system. A related issue is the investigation of how models other than the Boolean model (which is still largely adopted by most large commercial library systems) affect the user of a library.

A specialized retrieval system is one which is developed with a particular application in mind. For instance, the LEXIS-NEXIS retrieval system (see Chapter 14), which provides access to a very large collection of legal and business documents, is a good example of a specialized retrieval system. In such a system, a key problem is how to retrieve (almost) all documents which might be relevant to the user information need without also retrieving a large number of unrelated documents. In this context, sophisticated ranking algorithms are highly desirable. Since ranking based on single evidential sources is unlikely to provide the appropriate answers, research on approaches for combining several evidential sources seems highly relevant (as demonstrated at the various TREC conferences, see Chapter 3 for details).

In the Web, the scenario is quite distinct and unique. In fact, the user of the Web frequently does not know what he wants or has great difficulty in properly formulating his request. Thus, research in advanced user interfaces is highly desirable. From the point of view of the ranking engine, an interesting problem is to study how the paradigm adopted for the user interface affects the ranking. Furthermore, it is now well established that the indexes maintained by the various Web search engines are almost disjoint (e.g., the ten most popular search engines have indexes whose intersection corresponds to less than 2% of the total number of pages indexed). In this scenario, research on meta-search engines (i.e., engines which work by fusing the rankings generated by other search engines) seems highly promising.

2.12 Bibliographic Discussion

Early in 1960, Maron and Kuhns [547] had already discussed the issues of relevance and probabilistic indexing in information retrieval. Twenty-three years

later, Salton and McGill wrote a book [698] which became a classic in the field. The book provides a thorough coverage of the three classic models in information retrieval namely, the Boolean, the vector, and the probabilistic models. Another landmark reference is the book by van Rijsbergen [785] which, besides also covering the three classic models, presents a thorough and enjoyable discussion on the probabilistic model. The book edited by Frakes and Baeza-Yates [275] presents several data structures and algorithms for IR and is more recent. Further, it includes a discussion of ranking algorithms by Harman [340] which provides interesting insights into the history of information retrieval from 1960 to 1990.

Boolean operations and their implementation are covered in [803]. The inadequacy of Boolean queries for information retrieval was characterized early on by Verhoeff, Goffman, and Belzer [786]. The issue of adapting the Boolean formalism to operate with other frameworks received great attention. Bookstein discusses the problems related with merging Boolean and weighted retrieval systems [101] and the implications of Boolean structure for probabilistic retrieval [103]. Losee and Bookstein [522] cover the usage of Boolean queries with probabilistic retrieval. Anick *et al.* [21] propose an interface based on natural language for Boolean retrieval. A thesaurus-based Boolean retrieval system is proposed in [493].

The vector model is maybe the most popular model among the research community in information retrieval. Much of this popularity is due to the long-term research of Salton and his associates [697, 704]. Most of this research revolved around the SMART retrieval system developed at Cornell University [695, 842, 696]. Term weighting for the vector model has also been investigated thoroughly. Simple term weighting was used early on by Salton and Lesk [697]. Sparck Jones introduced the idf factor [409, 410] and Salton and Yang verified its effectiveness for improving retrieval [704]. Yu and Salton [842] further studied the effects of term weighting in the final ranking. Salton and Buckley [696] summarize 20 years of experiments in term weighting with the SMART system. Raghavan and Wong [665] provide a critical analysis of the vector model.

The probabilistic model was introduced by Robertson and Sparck Jones [677] and is thoroughly discussed in [785]. Experimental studies with the model were conducted by Sparck Jones [411, 412] which used feedback from the user to estimate the initial probabilities. Croft and Harper [199] proposed a method to estimate these probabilities without feedback from the user. Croft [198] later on added within-document frequency weights into the model. Fuhr discusses probabilistic indexing through polynomial retrieval functions [281, 284]. Cooper, Gey, and Dabney [186] and later on Gey [295] propose the use of logistic regression with probabilistic retrieval. Lee and Kantor [494] study the effect of inconsistent expert judgements on probabilistic retrieval. Fuhr [282] reviews various variants of the classic probabilistic model. Cooper [187], in a seminal paper, raises troubling questions on the utilization of the probabilistic ranking principle in information retrieval.

The inference network model was introduced by Turtle and Croft [772, 771] in 1990. Haines and Croft [332] discuss the utilization of inference networks for user relevance feedback (see Chapter 5). Callan, Lu, and Croft [139] use an

inference network to search distributed document collections. Callan [138], in his turn, discusses the application of inference networks to information filtering. The belief network model, due to Ribeiro-Neto and Muntz [674], generalizes the inference network model.

The extended Boolean model was introduced by Salton, Fox, and Wu [703]. Lee, Kim, Kim, and Lee [496] discuss the evaluation of Boolean operators with the extended Boolean model, while properties of the model are discussed in [495]. The generalized vector space model was introduced in 1985 by Wong, Ziarko, and Wong [832, 831]. Latent semantic indexing was introduced in 1988 by Furnas, Deerwester, Dumais, Landauer, Harshman, Streeter, and Lochbaum [287]. In a subsequent paper, Bartell, Cottrell, and Belew [62] show that latent semantic indexing can be interpreted as a special case of multidimensional scaling.

Regarding neural network models for information retrieval, our discussion in this book is based mainly on the work by Wilkinson and Hingston [815]. But we also benefited from the writings of Kwok on the subject and related topics [466, 467, 469, 468].

The fuzzy set model (for information retrieval) covered in this book is due to Ogawa, Morita, and Kobayashi [616]. The utilization of fuzzy theory in information retrieval goes back to the 1970s with the work of Radecki [658, 659, 660, 661], of Sachs [691], and of Tahani [755]. Bookstein [102] proposes the utilization of fuzzy operators to deal with weighted Boolean searches. Kraft and Buel [461] utilize fuzzy sets to generalize a Boolean system. Miyamoto, Miyake, and Nakayama [567] discuss the generation of a pseudothesaurus using co-occurrences and fuzzy operators. Subsequently, Miyamoto and Nakayama [568] discuss the utilization of this thesaurus with information retrieval systems.

Our discussion on structured text is based on the survey by [41]. Another survey of interest (an older one though) is the work by MacLeod [533]. Burkowski [132, 133] proposed a model based on non-overlapping regions. Clarke, Cormack, and Burkowski [173] extended this model with overlapping capabilities. The model based on proximal nodes was proposed by Navarro and Baeza-Yates [589, 590]. In [534], MacLeod introduced a model based on a single hierarchy which also associates attributes with nodes in the hierarchy (for database-like querying) and hypertext links with pairs of nodes. Kilpelainen and Mannila [439] discuss the retrieval from hierarchical texts through the specification of partial patterns. In [183], Consens and Milo discuss algebras for querying text regions.

A classic reference on hypertexts is the book by Nielsen [604]. Another popular reference is the book by Shneiderman and Kearsley [727]. Conklin [181] presents an introductory survey of the area. The *Communications of the ACM* dedicated an special edition [177] to hypermedia which discusses in detail the Dexter model — a reference standard on the terminology and semantics of basic hypermedia concepts. A subsequent edition [178] was dedicated to the presentation of various models for supporting the design of hypermedia applications.

Chapter 3
Retrieval Evaluation

3.1 Introduction

Before the final implementation of an information retrieval system, an evaluation of the system is usually carried out. The type of evaluation to be considered depends on the objectives of the retrieval system. Clearly, any software system has to provide the functionality it was conceived for. Thus, the first type of evaluation which should be considered is a functional analysis in which the specified system functionalities are tested one by one. Such an analysis should also include an error analysis phase in which, instead of looking for functionalities, one behaves erratically trying to make the system fail. It is a simple procedure which can be quite useful for catching programming errors. Given that the system has passed the functional analysis phase, one should proceed to evaluate the performance of the system.

The most common measures of system performance are time and space. The shorter the response time, the smaller the space used, the better the system is considered to be. There is an inherent tradeoff between space complexity and time complexity which frequently allows trading one for the other. In Chapter 8 we discuss this issue in detail.

In a system designed for providing data retrieval, the response time and the space required are usually the metrics of most interest and the ones normally adopted for evaluating the system. In this case, we look for the performance of the indexing structures (which are in place to accelerate the search), the interaction with the operating system, the delays in communication channels, and the overheads introduced by the many software layers which are usually present. We refer to such a form of evaluation simply as *performance evaluation*.

In a system designed for providing information retrieval, other metrics, besides time and space, are also of interest. In fact, since the user query request is inherently vague, the retrieved documents are not exact answers and have to be ranked according to their relevance to the query. Such relevance ranking introduces a component which is not present in data retrieval systems and which plays a central role in information retrieval. Thus, information retrieval systems require the evaluation of how precise is the answer set. This type of evaluation is referred to as *retrieval performance evaluation*.

In this chapter, we discuss retrieval performance evaluation for information retrieval systems. Such an evaluation is usually based on a test reference collection and on an evaluation measure. The test reference collection consists of a collection of documents, a set of example information requests, and a set of relevant documents (provided by specialists) for each example information request. Given a retrieval strategy S, the evaluation measure quantifies (for each example information request) the *similarity* between the set of documents retrieved by S and the set of relevant documents provided by the specialists. This provides an estimation of the *goodness* of the retrieval strategy S.

In our discussion, we first cover the two most used retrieval evaluation measures: recall and precision. We also cover alternative evaluation measures such as the E measure, the harmonic mean, satisfaction, frustration, etc. Following that, we cover four test reference collections namely, TIPSTER/TREC, CACM, CISI, and Cystic Fibrosis.

3.2 Retrieval Performance Evaluation

When considering retrieval performance evaluation, we should first consider the retrieval task that is to be evaluated. For instance, the retrieval task could consist simply of a query processed in batch mode (i.e., the user submits a query and receives an answer back) or of a whole interactive session (i.e., the user specifies his information need through a series of interactive steps with the system). Further, the retrieval task could also comprise a combination of these two strategies. Batch and interactive query tasks are quite distinct processes and thus their evaluations are also distinct. In fact, in an interactive session, user effort, characteristics of the interface design, guidance provided by the system, and duration of the session are critical aspects which should be observed and measured. In a batch session, none of these aspects is nearly as important as the quality of the answer set generated.

Besides the nature of the query request, one has also to consider the setting where the evaluation will take place and the type of interface used. Regarding the setting, evaluation of experiments performed in a laboratory might be quite distinct from evaluation of experiments carried out in a real life situation. Regarding the type of interface, while early bibliographic systems (which still dominate the commercial market as discussed in Chapter 14) present the user with interfaces which normally operate in batch mode, newer systems (which are been popularized by the high quality graphic displays available nowadays) usually present the user with complex interfaces which often operate interactively.

Retrieval performance evaluation in the early days of computer-based information retrieval systems focused primarily on laboratory experiments designed for batch interfaces. In the 1990s, a lot more attention has been paid to the evaluation of real life experiments. Despite this tendency, laboratory experimentation is still dominant. Two main reasons are the repeatability and the scalability provided by the closed setting of a laboratory.

In this book, we focus mainly on experiments performed in laboratories. In this chapter in particular we discuss solely the evaluation of systems which operate in batch mode. Evaluation of systems which operate interactively is briefly discussed in Chapter 10.

3.2.1 Recall and Precision

Consider an example information request I (of a test reference collection) and its set R of relevant documents. Let $|R|$ be the number of documents in this set. Assume that a given retrieval strategy (which is being evaluated) processes the information request I and generates a document answer set A. Let $|A|$ be the number of documents in this set. Further, let $|Ra|$ be the number of documents in the intersection of the sets R and A. Figure 3.1 illustrates these sets.

The recall and precision measures are defined as follows.

- **Recall** is the fraction of the relevant documents (the set R) which has been retrieved i.e.,

$$Recall = \frac{|Ra|}{|R|}$$

- **Precision** is the fraction of the retrieved documents (the set A) which is relevant i.e.,

$$Precision = \frac{|Ra|}{|A|}$$

Recall and precision, as defined above, assume that all the documents in the answer set A have been examined (or seen). However, the user is not usually presented with all the documents in the answer set A at once. Instead, the

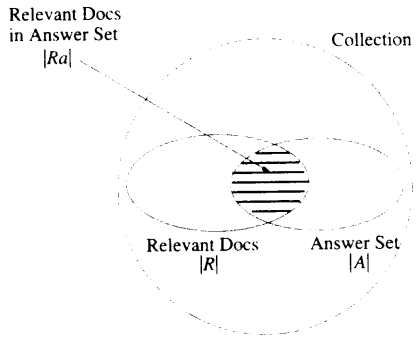

Figure 3.1 Precision and recall for a given example information request.

documents in A are first sorted according to a degree of relevance (i.e., a ranking is generated). The user then examines this ranked list starting from the top document. In this situation, the recall and precision measures vary as the user proceeds with his examination of the answer set A. Thus, proper evaluation requires plotting a precision versus recall curve as follows.

As before, consider a reference collection and its set of example information requests. Let us focus on a given example information request for which a query q is formulated. Assume that a set R_q containing the relevant documents for q has been defined. Without loss of generality, assume further that the set R_q is composed of the following documents

$$R_q = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\} \quad \textbf{(3.1)}$$

Thus, according to a group of specialists, there are ten documents which are relevant to the query q.

Consider now a new retrieval algorithm which has just been designed. Assume that this algorithm returns, for the query q, a ranking of the documents in the answer set as follows.

Ranking for query q:

1. d_{123} •	6. d_9 •	11. d_{38}
2. d_{84}	7. d_{511}	12. d_{48}
3. d_{56} •	8. d_{129}	13. d_{250}
4. d_6	9. d_{187}	14. d_{113}
5. d_8	10. d_{25} •	15. d_3 •

The documents that are relevant to the query q are marked with a bullet after the document number. If we examine this ranking, starting from the top document, we observe the following points. First, the document d_{123} which is ranked as number 1 is relevant. Further, this document corresponds to 10% of all the relevant documents in the set R_q. Thus, we say that we have a precision of 100% at 10% recall. Second, the document d_{56} which is ranked as number 3 is the next relevant document. At this point, we say that we have a precision of roughly 66% (two documents out of three are relevant) at 20% recall (two of the ten relevant documents have been seen). Third, if we proceed with our examination of the ranking generated we can plot a curve of precision versus recall as illustrated in Figure 3.2. The precision at levels of recall higher than 50% drops to 0 because not all relevant documents have been retrieved. This precision versus recall curve is usually based on 11 (instead of ten) *standard* recall levels which are 0%, 10%, 20%, ..., 100%. For the recall level 0%, the precision is obtained through an interpolation procedure as detailed below.

In the above example, the precision and recall figures are for a single query. Usually, however, retrieval algorithms are evaluated by running them for several distinct queries. In this case, for each query a distinct precision versus recall curve is generated. To evaluate the retrieval performance of an algorithm over

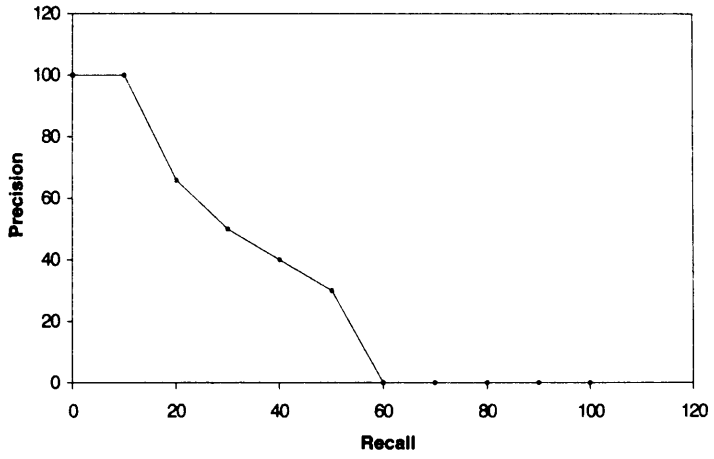

Figure 3.2 Precision at 11 standard recall levels.

all test queries, we average the precision figures at each recall level as follows.

$$\overline{P}(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q} \tag{3.2}$$

where $\overline{P}(r)$ is the average precision at the recall level r, N_q is the number of queries used, and $P_i(r)$ is the precision at recall level r for the i-th query.

Since the recall levels for each query might be distinct from the 11 standard recall levels, utilization of an interpolation procedure is often necessary. For instance, consider again the set of 15 ranked documents presented above. Assume that the set of relevant documents for the query q has changed and is now given by

$$R_q = \{d_3, d_{56}, d_{129}\} \tag{3.3}$$

In this case, the first relevant document in the ranking for query q is d_{56} which provides a recall level of 33.3% (with precision also equal to 33.3%) because, at this point, one-third of all relevant documents have already been seen. The second relevant document is d_{129} which provides a recall level of 66.6% (with precision equal to 25%). The third relevant document is d_3 which provides a recall level of 100% (with precision equal to 20%). The precision figures at the 11 standard recall levels are interpolated as follows.

Let r_j, $j \in \{0, 1, 2, \ldots, 10\}$, be a reference to the j-th standard recall level (i.e., r_5 is a reference to the recall level 50%). Then,

$$P(r_j) = max_{\,r_j \leq r \leq r_{j+1}} \; P(r) \tag{3.4}$$

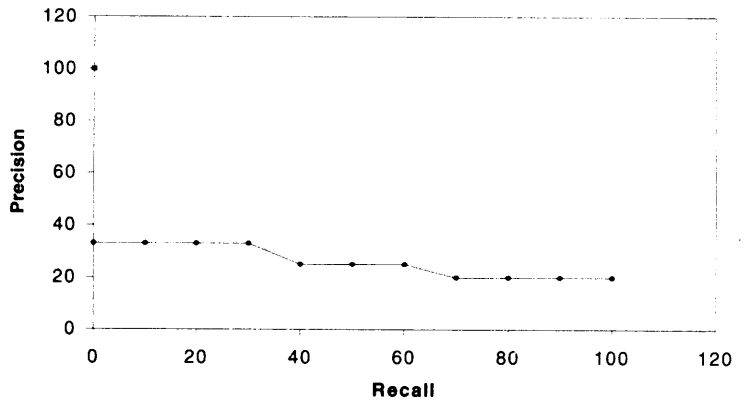

Figure 3.3 Interpolated precision at 11 standard recall levels relative to $R_q = \{d_3, d_{56}, d_{129}\}$.

which states that the interpolated precision at the j-th standard recall level is the maximum known precision at any recall level between the j-th recall level and the $(j+1)$-th recall level.

In our last example, this interpolation rule yields the precision and recall figures illustrated in Figure 3.3. At recall levels 0%, 10%, 20%, and 30%, the interpolated precision is equal to 33.3% (which is the known precision at the recall level 33.3%). At recall levels 40%, 50%, and 60%, the interpolated precision is 25% (which is the precision at the recall level 66.6%). At recall levels 70%, 80%, 90%, and 100%, the interpolated precision is 20% (which is the precision at recall level 100%).

The curve of precision versus recall which results from averaging the results for various queries is usually referred to as precision versus recall figures. Such average figures are normally used to compare the retrieval performance of distinct retrieval algorithms. For instance, one could compare the retrieval performance of a newly proposed retrieval algorithm with the retrieval performance of the classic vector space model. Figure 3.4 illustrates average precision versus recall figures for two distinct retrieval algorithms. In this case, one algorithm has higher precision at lower recall levels while the second algorithm is superior at higher recall levels.

One additional approach is to compute average precision at given *document cutoff values*. For instance, we can compute the average precision when 5, 10, 15, 20, 30, 50, or 100 relevant documents have been seen. The procedure is analogous to the computation of average precision at 11 standard recall levels but provides additional information on the retrieval performance of the ranking algorithm.

Average precision versus recall figures are now a standard evaluation strategy for information retrieval systems and are used extensively in the information retrieval literature. They are useful because they allow us to evaluate

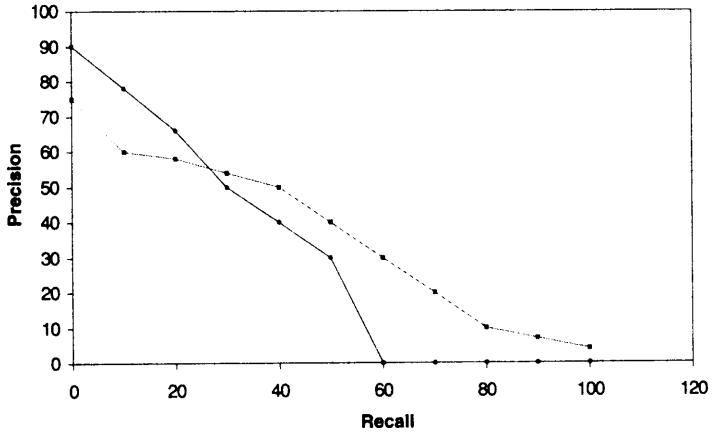

Figure 3.4 Average recall versus precision figures for two distinct retrieval algorithms.

quantitatively both the quality of the overall answer set and the breadth of the retrieval algorithm. Further, they are simple, intuitive, and can be combined in a single curve. However, precision versus recall figures also have their disadvantages and their widespread usage has been criticized in the literature. We return to this point later on. Before that, let us discuss techniques for summarizing precision versus recall figures by a single numerical value.

Single Value Summaries

Average precision versus recall figures are useful for comparing the retrieval performance of distinct retrieval algorithms over a set of example queries. However, there are situations in which we would like to compare the retrieval performance of our retrieval algorithms for the individual queries. The reasons are twofold. First, averaging precision over many queries might disguise important anomalies in the retrieval algorithms under study. Second, when comparing two algorithms, we might be interested in investigating whether one of them outperforms the other for each query in a given set of example queries (notice that this fact can be easily hidden by an average precision computation). In these situations, a single precision value (for each query) can be used. This single value should be interpreted as a summary of the corresponding precision versus recall curve. Usually, this single value summary is taken as the precision at a specified recall level. For instance, we could evaluate the precision when we observe the first relevant document and take this precision as the single value summary. Of course, as seems obvious, this is not a good approach. More interesting strategies can be adopted as we now discuss.

Average Precision at Seen Relevant Documents

The idea here is to generate a single value summary of the ranking by averaging the precision figures obtained after each new relevant document is observed (in the ranking). For instance, consider the example in Figure 3.2. The precision figures after each new relevant document is observed are 1, 0.66, 0.5, 0.4, and 0.3. Thus, the *average precision at seen relevant documents* is given by (1+0.66+0.5+0.4+0.3)/5 or 0.57. This measure favors systems which retrieve relevant documents quickly (i.e., early in the ranking). Of course, an algorithm might present a good average precision at seen relevant documents but have a poor performance in terms of overall recall.

R-Precision

The idea here is to generate a single value summary of the ranking by computing the precision at the R-th position in the ranking, where R is the total number of relevant documents for the current query (i.e., number of documents in the set R_q). For instance, consider the examples in Figures 3.2 and 3.3. The value of R-precision is 0.4 for the first example (because $R = 10$ and there are four relevant documents among the first ten documents in the ranking) and 0.33 for the second example (because $R = 3$ and there is one relevant document among the first three documents in the ranking). The R-precision measure is a useful parameter for observing the behavior of an algorithm for each individual query in an experiment. Additionally, one can also compute an average R-precision figure over all queries. However, using a single number to summarize the full behavior of a retrieval algorithm over several queries might be quite imprecise.

Precision Histograms

The R-precision measures for several queries can be used to compare the retrieval history of two algorithms as follows. Let $RP_A(i)$ and $RP_B(i)$ be the R-precision values of the retrieval algorithms A and B for the i-th query. Define, for instance, the difference

$$RP_{A/B}(i) = RP_A(i) - RP_B(i) \tag{3.5}$$

A value of $RP_{A/B}(i)$ equal to 0 indicates that both algorithms have equivalent performance (in terms of R-precision) for the i-th query. A positive value of $RP_{A/B}(i)$ indicates a better retrieval performance by algorithm A (for the i-th query) while a negative value indicates a better retrieval performance by algorithm B. Figure 3.5 illustrates the $RP_{A/B}(i)$ values (labeled *R-Precision A/B*) for two hypothetical retrieval algorithms over ten example queries. The algorithm A is superior for eight queries while the algorithm B performs better for the two other queries (numbered 4 and 5). This type of bar graph is called a *precision histogram* and allows us to quickly compare the retrieval performance history of two algorithms through visual inspection.

Summary Table Statistics

Single value measures can also be stored in a table to provide a statistical summary regarding the set of all the queries in a retrieval task. For instance, these

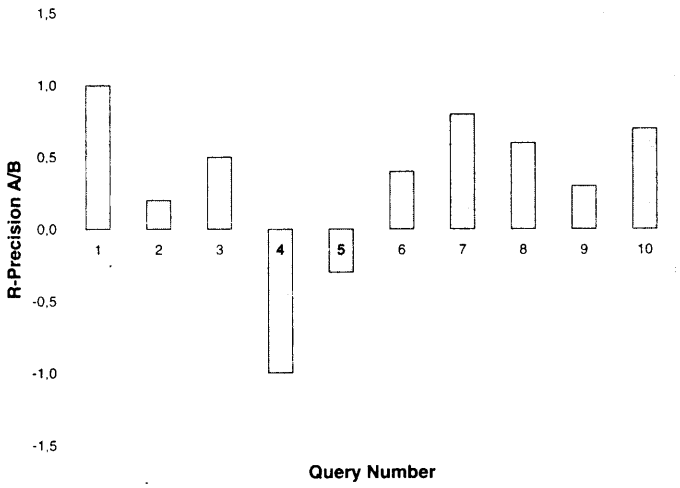

Figure 3.5 A precision histogram for ten hypothetical queries.

summary table statistics could include: the number of queries used in the task, the total number of documents retrieved by all queries, the total number of relevant documents which were effectively retrieved when all queries are considered, the total number of relevant documents which could have been retrieved by all queries, etc.

Precision and Recall Appropriateness

Precision and recall have been used extensively to evaluate the retrieval performance of retrieval algorithms. However, a more careful reflection reveals problems with these two measures [451, 664, 754]. First, the proper estimation of maximum recall for a query requires detailed knowledge of all the documents in the collection. With large collections, such knowledge is unavailable which implies that recall cannot be estimated precisely. Second, recall and precision are related measures which capture different aspects of the set of retrieved documents. In many situations, the use of a single measure which combines recall and precision could be more appropriate. Third, recall and precision measure the effectiveness over a set of queries processed in batch mode. However, with modern systems, interactivity (and not batch processing) is the key aspect of the retrieval process. Thus, measures which quantify the *informativeness* of the retrieval process might now be more appropriate. Fourth, recall and precision are easy to define when a linear ordering of the retrieved documents is enforced. For systems which require a weak ordering though, recall and precision might be inadequate.

3.2.2 Alternative Measures

Since recall and precision, despite their popularity, are not always the most appropriate measures for evaluating retrieval performance, alternative measures have been proposed over the years. A brief review of some of them is as follows.

The Harmonic Mean

As discussed above, a single measure which combines recall and precision might be of interest. One such measure is the harmonic mean F of recall and precision [422] which is computed as

$$F(j) = \frac{2}{\frac{1}{r(j)} + \frac{1}{P(j)}} \tag{3.6}$$

where $r(j)$ is the recall for the j-th document in the ranking, $P(j)$ is the precision for the j-th document in the ranking, and $F(j)$ is the harmonic mean of $r(j)$ and $P(j)$ (thus, relative to the j-th document in the ranking). The function F assumes values in the interval $[0, 1]$. It is 0 when no relevant documents have been retrieved and is 1 when all ranked documents are relevant. Further, the harmonic mean F assumes a high value only when both recall and precision are high. Therefore, determination of the maximum value for F can be interpreted as an attempt to find the best possible compromise between recall and precision.

The E Measure

Another measure which combines recall and precision was proposed by van Rijsbergen [785] and is called the E evaluation measure. The idea is to allow the user to specify whether he is more interested in recall or in precision. The E measure is defined as follows.

$$E(j) = 1 - \frac{1 + b^2}{\frac{b^2}{r(j)} + \frac{1}{P(j)}}$$

where $r(j)$ is the recall for the j-th document in the ranking, $P(j)$ is the precision for the j-th document in the ranking, $E(j)$ is the E evaluation measure relative to $r(j)$ and $P(j)$, and b is a user specified parameter which reflects the relative importance of recall and precision. For $b = 1$, the $E(j)$ measure works as the complement of the harmonic mean $F(j)$. Values of b greater than 1 indicate that the user is more interested in precision than in recall while values of b smaller than 1 indicate that the user is more interested in recall than in precision.

User-Oriented Measures

Recall and precision are based on the assumption that the set of relevant documents for a query is the same, independent of the user. However, different users might have a different interpretation of which document is relevant and which one is not. To cope with this problem, *user-oriented* measures have been proposed such as coverage ratio, novelty ratio, relative recall, and recall effort [451].

As before, consider a reference collection, an example information request I, and a retrieval strategy to be evaluated. Let R be the set of relevant documents for I and A be the answer set retrieved. Also, let U be the subset of R which is known to the user. The number of documents in U is $|U|$. The intersection of the sets A and U yields the documents known to the user to be relevant which were retrieved. Let $|Rk|$ be the number of documents in this set. Further, let $|Ru|$ be the number of relevant documents previously unknown to the user which were retrieved. Figure 3.6 illustrates the situation. The *coverage ratio* is defined as the fraction of the documents known (to the user) to be relevant which has actually been retrieved i.e.,

$$coverage = \frac{|Rk|}{|U|}$$

The *novelty ratio* is defined as the fraction of the relevant documents retrieved which was unknown to the user i.e.,

$$novelty = \frac{|Ru|}{|Ru| + |Rk|}$$

A high coverage ratio indicates that the system is finding most of the relevant documents the user expected to see. A high novelty ratio indicates that the system is revealing (to the user) many new relevant documents which were previously unknown.

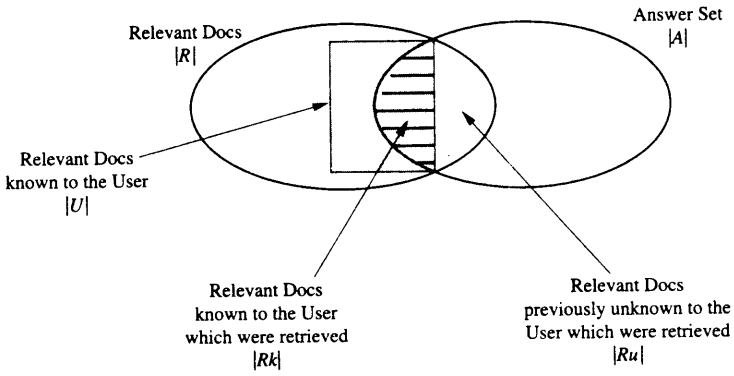

Figure 3.6 Coverage and novelty ratios for a given example information request:

Additionally, two other measures can be defined as follows. The *relative recall* is given by the ratio between the number of relevant documents found (by the system) and the number of relevant documents the user expected to find. In the case when the user finds as many relevant documents as he expected, he stops searching and the relative recall is equal to 1. The *recall effort* is given by the ratio between the number of relevant documents the user expected to find and the number of documents examined in an attempt to find the expected relevant documents.

Other Measures

Other measures which might be of interest include the *expected search length*, which is good for dealing with sets of documents weakly ordered, the *satisfaction*, which takes into account only the relevant documents, and the *frustration*, which takes into account only the non-relevant documents [451].

3.3 Reference Collections

In this section we discuss various reference collections which have been used throughout the years for the evaluation of information retrieval systems. We first discuss the TIPSTER/TREC collection which, due to its large size and thorough experimentation, is usually considered to be the *reference* test collection in information retrieval nowadays. Following that, we cover the CACM and ISI collections due to their historical importance in the area of information retrieval. We conclude this section with a brief discussion of the Cystic Fibrosis collection. It is a small collection whose example information requests were extensively studied by four groups of specialists before generation of the relevant document sets.

3.3.1 The TREC Collection

Research in information retrieval has frequently been criticized on two fronts. First, that it lacks a solid formal framework as a basic foundation. Second, that it lacks robust and consistent testbeds and benchmarks. The first of these criticisms is difficult to dismiss entirely due to the inherent degree of psychological *subjectiveness* associated with the task of deciding on the relevance of a given document (which characterizes information, as opposed to data, retrieval). Thus, at least for now, research in information retrieval will have to proceed without a solid formal underpinning. The second of these criticisms, however, can be acted upon. For three decades, experimentation in information retrieval was based on relatively small test collections which did not reflect the main issues present in a large bibliographical environment. Further, comparisons between various retrieval systems were difficult to make because distinct groups conducted experiments focused on distinct aspects of retrieval (even when the same test collection was used) and there were no widely accepted benchmarks.

In the early 1990s, a reaction to this state of disarray was initiated under the leadership of Donna Harman at the National Institute of Standards and Technology (NIST), in Maryland. Such an effort consisted of promoting a yearly conference, named TREC for Text REtrieval Conference, dedicated to experimentation with a large test collection comprising over a million documents. For each TREC conference, a set of reference experiments is designed. The research groups which participate in the conference use these reference experiments for comparing their retrieval systems.

A clear statement of the purpose of the TREC conferences can be found in the NIST TREC Web site [768] and reads as follows.

> The TREC conference series is co-sponsored by the National Institute of Standards and Technology (NIST) and the Information Technology Office of the Defense Advanced Research Projects Agency (DARPA) as part of the TIPSTER Text Program. The goal of the conference series is to encourage research in information retrieval from large text applications by providing a large test collection, uniform scoring procedures, and a forum for organizations interested in comparing their results. Attendance at TREC conferences is restricted to those researchers and developers who have performed the TREC retrieval tasks and to selected government personnel from sponsoring agencies.
>
> Participants in a TREC conference employ a wide variety of retrieval techniques, including methods using automatic thesauri, sophisticated term weighting, natural language techniques, relevance feedback, and advanced pattern matching. Each system works with the same test collection that consists of about 2 gigabytes of text (over 1 million documents) and a given set of information needs called 'topics.' Results are run through a common evaluation package so that groups can compare the effectiveness of different techniques and can determine how differences between systems affect performance.

Since the collection was built under the TIPSTER program, it is frequently referred to as the TIPSTER or the TIPSTER/TREC test collection. Here, however, for simplicity we refer to it as the TREC collection.

The first TREC conference was held at NIST in November 1992, while the second TREC conference occurred in August 1993. In November 1997, the sixth TREC conference was held (also at NIST) and counted the following participating organizations (extracted from [794]):

Apple Computer
AT&T Labs Research
Australian National Univ.
Carnegie Mellon Univ.
CEA (France)
Center for Inf. Res., Russia
City Univ., London
CLARITECH Corporation
Cornell Univ./SaBIR Research, Inc.
CSIRO (Australia)
Daimler Benz Res. Center, Ulm
Dublin Univ. Center

Duke Univ./Univ. of Colorado/Bellcore
ETH (Switzerland)
FS Consulting, Inc.
GE Corp./Rutgers Univ.
George Mason Univ./NCR Corp.
Harris Corp.
IBM T.J. Watson Res. (2 groups)
ISS (Singapore)
ITI (Singapore)
APL, Johns Hopkins Univ.
LEXIS-NEXIS
MDS at RMIT, Australia
MIT/IBM Almaden Res. Center
MSI/IRIT/Univ. Toulouse
NEC Corporation
New Mexico State Univ. (2 groups)
NSA (Speech Research Group)
Open Text Corporation
Oregon Health Sciences Univ.
Queens College, CUNY
Rutgers Univ. (2 groups)
Siemens AG
SRI International
TwentyOne
Univ. California, Berkeley
Univ. California, San Diego
Univ. Glasgow
Univ. Maryland, College Park
Univ. Massachusetts, Amherst
Univ. Montreal
Univ. North Carolina (2 groups)
Univ. Sheffield/Univ. Cambridge
Univ. Waterloo
Verity, Inc.
Xerox Res. Centre Europe

The seventh TREC conference was held again at NIST in November of 1998.

In the following, we briefly discuss the TREC document collection and the (benchmark) tasks at the TREC conferences. As with most test collections, the TREC collection is composed of three parts: the documents, the example information requests (called *topics* in the TREC nomenclature), and a set of relevant documents for each example information request. Further, the TREC conferences also include a set of tasks to be used as a benchmark.

The Document Collection

The TREC collection has been growing steadily over the years. At TREC-3, the collection size was roughly 2 gigabytes while at TREC-6 it had gone up to roughly 5.8 gigabytes. In the beginning, copyright restrictions prevented free distribution of the collection and, as a result, the distribution CD-ROM disks had to be bought. In 1998, however, an arrangement was made which allows free access to the documents used in the most recent TREC conferences. As a result, TREC disk 4 and TREC disk 5 are now available from NIST at a small fee (US$200 in 1998) to cover distribution costs. Information on how to obtain the collection (which comes with the disks) and the topics with their relevant document sets (which have to be retrieved through the network) can be obtained directly from the NIST TREC Web site [768].

The TREC collection is distributed in six CD-ROM disks of roughly 1 gigabyte of compressed text each. The documents come from the following sources:

WSJ	→ *Wall Street Journal*
AP	→ Associated Press (news wire)
ZIFF	→ Computer Selects (articles), Ziff-Davis
FR	→ Federal Register

DOE → US DOE Publications (abstracts)
SJMN → *San Jose Mercury News*
PAT → US Patents
FT → *Financial Times*
CR → Congressional Record
FBIS → Foreign Broadcast Information Service
LAT → *LA Times*

Table 3.1 illustrates the contents of each disk and some simple statistics regarding the collection (extracted from [794]). Documents from all subcollections are

Disk	*Contents*	*Size Mb*	*Number Docs*	*Words/Doc. (median)*	*Words/Doc. (mean)*
1	WSJ, 1987-1989	267	98,732	245	434.0
	AP, 1989	254	84,678	446	473.9
	ZIFF	242	75,180	200	473.0
	FR, 1989	260	25,960	391	1315.9
	DOE	184	226,087	111	120.4
2	WSJ, 1990-1992	242	74,520	301	508.4
	AP, 1988	237	79,919	438	468.7
	ZIFF	175	56,920	182	451.9
	FR, 1988	209	19,860	396	1378.1
3	SJMN, 1991	287	90,257	379	453.0
	AP, 1990	237	78,321	451	478.4
	ZIFF	345	161,021	122	295.4
	PAT, 1993	243	6,711	4,445	5391.0
4	FT, 1991-1994	564	210,158	316	412.7
	FR, 1994	395	55,630	588	644.7
	CR, 1993	235	27,922	288	1373.5
5	FBIS	470	130,471	322	543.6
	LAT	475	131,896	351	526.5
6	FBIS	490	120,653	348	581.3

Table 3.1 Document collection used at TREC-6. Stopwords are not removed and no stemming is performed (see Chapter 7 for details on stemming).

tagged with SGML (see Chapter 6) to allow easy parsing (which implies simple coding for the groups participating at TREC conferences). Major structures such as a field for the document number (identified by <DOCNO>) and a field for the document text (identified by <TEXT>) are common to all documents. Minor structures might be different across subcollections to preserve parts of the structure in the original document. This has been the philosophy for formatting decisions at NIST: preserve as much of the original structure as possible while providing a common framework which allows simple decoding of the data.

An example of a TREC document is the document numbered 880406-0090

```
<doc>

<docno> WSJ880406-0090 </docno>
<hl> AT&T Unveils Services to Upgrade Phone Networks Under
Global Plan </hl>
<author> Janet Guyon (WSJ Staff) </author>
<dateline> New York </dateline>

<text>
American Telephone & Telegraph Co. introduced the first of a new
generation of phone services with broad ...
</text>

</doc>
```

Figure 3.7 TREC document numbered WSJ880406-0090.

in the *Wall Street Journal* subcollection which is shown in Figure 3.7 (extracted from [342]). Further details on the TREC document collection can be obtained from [794, 768].

The Example Information Requests (Topics)

The TREC collection includes a set of example *information requests* which can be used for testing a new ranking algorithm. Each request is a description of an information need in natural language. In the TREC nomenclature, each test information request is referred to as a *topic*. An example of an information request in TREC is the topic numbered 168 (prepared for the TREC-3 conference) which is illustrated in Figure 3.8 (extracted from [342]).

The task of converting an information request (topic) into a system query (i.e., a set of index terms, a Boolean expression, a fuzzy expression, etc.) must be done by the system itself and is considered to be an integral part of the evaluation procedure.

The number of topics prepared for the first six TREC conferences goes up to 350. The topics numbered 1 to 150 were prepared for use with the TREC-1 and TREC-2 conferences. They were written by people who were experienced users of real systems and represented long-standing information needs. The topics numbered 151 to 200 were prepared for use with the TREC-3 conference, are shorter, and have a simpler structure which includes only three subfields (named Title, Description, and Narrative as illustrated in the topic 168 above). The topics numbered 201 to 250 were prepared for use with the TREC-4 conference and are even shorter. At the TREC-5 (which included topics 251-300) and TREC-6 (which included topics 301-350) conferences, the topics were prepared with a composition similar to the topics in TREC-3 (i.e., they were expanded with respect to the topics in TREC-4 which were considered to be too short).

<top>

<num> Number: 168
<title> Topic: Financing AMTRAK

<desc> Description:
A document will address the role of the Federal Government in financing the operation of the National Railroad Transportation Corporation (AMTRAK).

<narr> Narrative: A relevant document must provide information on the government's responsibility to make AMTRAK an economically viable entity. It could also discuss the privatization of AMTRAK as an alternative to continuing government subsidies. Documents comparing government subsidies given to air and bus transportation with those provided to AMTRAK would also be relevant.

</top>

Figure 3.8 Topic numbered 168 in the TREC collection.

The Relevant Documents for Each Example Information Request

At the TREC conferences, the set of relevant documents for each example information request (topic) is obtained from a pool of possible relevant documents. This pool is created by taking the top K documents (usually, $K = 100$) in the rankings generated by the various participating retrieval systems. The documents in the pool are then shown to human assessors who ultimately decide on the relevance of each document.

This technique for assessing relevance is called the *pooling method* [794] and is based on two assumptions. First, that the vast majority of the relevant documents is collected in the assembled pool. Second, that the documents which are not in the pool can be considered to be not relevant. Both assumptions have been verified to be accurate in tests done at the TREC conferences. A detailed description of these relevance assessments can be found in [342, 794].

The (Benchmark) Tasks at the TREC Conferences

The TREC conferences include two main information retrieval tasks [342]. In the first, called *ad hoc* task, a set of new (conventional) requests are run against a fixed document database. This is the situation which normally occurs in a library where a user is asking new queries against a set of static documents. In the second, called *routing* task, a set of fixed requests are run against a database whose documents are continually changing. This is like a filtering task in which the same questions are always being asked against a set of dynamic documents (for instance, news clipping services). Unlike a pure filtering task, however, the retrieved documents must be ranked.

For the ad hoc task, the participant systems receive the test information requests and execute them on a pre-specified document collection. For the routing task, the participant systems receive the test information requests and two distinct document collections. The first collection is used for training and allows the tuning of the retrieval algorithm. The second collection is used for testing the tuned retrieval algorithm.

Starting at the TREC-4 conference, new secondary tasks, besides the ad hoc and routing tasks, were introduced with the purpose of allowing more specific comparisons among the various systems. At TREC-6, eight (specific) secondary tasks were added in as follows.

- **Chinese** Ad hoc task in which both the documents and the topics are in Chinese.
- **Filtering** Routing task in which the retrieval algorithm has only to decide whether a new incoming document is relevant (in which case it is taken) or not (in which case it is discarded). No ranking of the documents taken needs to be provided. The test data (incoming documents) is processed in time-stamp order.
- **Interactive** Task in which a human searcher interacts with the retrieval system to determine the relevant documents. Documents are ruled relevant or not relevant (i.e., no ranking is provided).
- **NLP** Task aimed at verifying whether retrieval algorithms based on natural language processing offer advantages when compared to the more traditional retrieval algorithms based on index terms.
- **Cross languages** Ad hoc task in which the documents are in one language but the topics are in a different language.
- **High precision** Task in which the user of a retrieval system is asked to retrieve ten documents that answer a given (and previously unknown) information request within five minutes (wall clock time).
- **Spoken document retrieval** Task in which the documents are written transcripts of radio broadcast news shows. Intended to stimulate research on retrieval techniques for spoken documents.
- **Very large corpus** Ad hoc task in which the retrieval systems have to deal with collections of size 20 gigabytes (7.5 million documents).

For TREC-7, the NLP and the Chinese secondary tasks were discontinued. Additionally, the routing task was retired as a main task because there is a consensus that the filtering task is a more realistic type of routing task. TREC-7 also included a new task called *Query Task* in which several distinct query versions were created for each example information request [794]. The main goal of this task is to allow investigation of query-dependent retrieval strategies, a well known problem with the TREC collection due to the sparsity of the given information requests (which present very little overlap) used in past TREC conferences.

Besides providing detailed descriptions of the tasks to be executed, the TREC conferences also make a clear distinction between two basic techniques for transforming the information requests (which are in natural language) into query statements (which might be in vector form, in Boolean form, etc.). In the TREC-6 conference, the allowable query construction methods were divided into *automatic* methods, in which the queries were derived completely automatically from the test information requests, and *manual* methods, in which the queries were derived using any means other than the fully automatic method [794].

Evaluation Measures at the TREC Conferences

At the TREC conferences, four basic types of evaluation measures are used: summary table statistics, recall-precision averages, document level averages, and average precision histograms. Briefly, these measures can be described as follows (see further details on these measures in Section 3.2).

- **Summary table statistics** Consists of a table which summarizes statistics relative to a given task. The statistics included are: the number of topics (information requests) used in the task, the number of documents retrieved over all topics, the number of relevant documents which were effectively retrieved for all topics, and the number of relevant documents which could have been retrieved for all topics.
- **Recall-precision averages** Consists of a table or graph with average precision (over all topics) at 11 standard recall levels. Since the recall levels of the individual queries are seldom equal to the standard recall levels, interpolation is used to define the precision at the standard recall levels. Further, a non-interpolated average precision over seen relevant documents (and over all topics) might be included.
- **Document level averages** In this case, average precision (over all topics) is computed at specified document cutoff values (instead of standard recall levels). For instance, the average precision might be computed when 5, 10, 20, 100 relevant documents have been seen. Further, the average R-precision value (over all queries) might also be provided.
- **Average precision histogram** Consists of a graph which includes a single measure for each separate topic. This measure (for a topic t_i) is given, for instance, by the difference between the R-precision (for topic t_i) for a target retrieval algorithm and the average R-precision (for topic t_i) computed from the results of all participating retrieval systems.

3.3.2 The CACM and ISI Collections

The TREC collection is a large collection which requires time consuming preparation before experiments can be carried out effectively at a local site. Further,

the testing itself is also time consuming and requires much more effort than that required to execute the testing in a small collection. For groups who are not interested in making this investment, an alternative approach is to use a smaller test collection which can be installed and experimented with in a much shorter time. Further, a small collection might include features which are not present in the larger TREC collection. For instance, it is well known that the example information requests at TREC present very little overlap among themselves and thus are not very useful for investigating the impact of techniques which take advantage of information derived from dependencies between the current and past user queries (an issue which received attention at the TREC-7 conference). Further, the TREC collection does not provide good support for experimenting with algorithms which combine distinct evidential sources (such as co-citations, bibliographic coupling, etc.) to generate a ranking. In these situations, alternative (and smaller) test collections might be more appropriate.

For the experimental studies in [271], five different (small) test collections were developed: ADI (documents on information science), CACM, INSPEC (abstracts on electronics, computer, and physics), ISI, and Medlars (medical articles). In this section we cover two of them in detail: the CACM and the ISI test collections. Our discussion is based on the work by Fox [272].

The CACM Collection

The documents in the CACM test collection consist of all the 3204 articles published in the *Communications of the ACM* from the first issue in 1958 to the last number of 1979. Those documents cover a considerable range of computer science literature due to the fact that the CACM served for many years as the premier periodical in the field.

Besides the text of the documents, the collection also includes information on structured *subfields* (called *concepts* by Fox) as follows:

- author names
- date information
- word stems from the title and abstract sections
- categories derived from a hierarchical classification scheme
- direct references between articles
- bibliographic coupling connections
- number of co-citations for each pair of articles.

The subfields 'author names' and 'date information' provide information on authors and date of publication. The subfield 'word stems' provides, for each document, a list of indexing terms (from the title and abstract sections) which have been stemmed (i.e., reduced to their grammatical roots as explained in Chapter 7). The subfield 'categories' assigns a list of classification categories (from the Computing Reviews category scheme) to each document. Since the

categories are fairly broad, the number of categories for any given document is usually smaller than five. The subfield 'direct references' provides a list of pairs of documents $[d_a,d_b]$ in which each pair identifies a document d_a which includes a direct reference to a document d_b. The subfield 'bibliographic coupling' provides a list of triples $[d_1,d_2,n_{cited}]$ in which the documents d_1 and d_2 both include a direct reference to a same third document d_j and the factor n_{cited} counts the number of documents d_j cited by both d_1 and d_2. The subfield 'co-citations' provides a list of triples $[d_1,d_2,n_{citing}]$ in which the documents d_1 and d_2 are both cited by a same third document d_j and the factor n_{citing} counts the number of documents d_j citing both d_1 and d_2. Thus, the CACM collection provides a unique environment for testing retrieval algorithms which are based on information derived from cross-citing patterns — a topic which has attracted much attention in the past.

The CACM collection also includes a set of 52 test information requests. For instance, the information request numbered 1 reads as follows.

> What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?

For each information request, the collection also includes two Boolean query formulations and a set of relevant documents. Since the information requests are fairly specific, the average number of relevant documents for each information request is small and around 15. As a result, precision and recall figures tend to be low.

The ISI Collection

The 1460 documents in the ISI (often referred to as CISI) test collection were selected from a previous collection assembled by Small [731] at the Institute of Scientific Information (ISI). The documents selected (which are about information sciences) were those most cited in a cross-citation study done by Small. The main purpose of the ISI collection is to support investigation of similarities based on terms and on cross-citation patterns.

The documents in the ISI collection include three types of subfields as follows.

- author names
- word stems from the title and abstract sections
- number of co-citations for each pair of articles.

The meaning of each of these subfields is as in the CACM collection.

The ISI collection includes a total of 35 test information requests (in natural language) for which there are Boolean query formulations. It also includes 41 additional test information requests for which there is no Boolean query formulation (only the version in natural language). The information requests are

fairly general which resulted in a larger number of relevant documents to each request (around 50). However, many of these relevant documents have no terms in common with the information requests which implies that precision and recall figures tend to be low.

Statistics for the CACM and ISI Collections

Tables 3.2 and 3.3 provide comparative summary statistics for the CACM and the ISI test collections.

Collection	*Num. Docs*	*Num. Terms*	*Terms/Docs.*
CACM	3204	10,446	40.1
ISI	1460	7392	104.9

Table 3.2 Document statistics for the CACM and ISI collections.

Collection	*Number Queries*	*Terms per Query*	*Relevants per Query*	*Relevants in Top 10*
CACM	52	11.4	15.3	1.9
ISI	35 & 76	8.1	49.8	1.7

Table 3.3 Query statistics for the CACM and ISI collections.

We notice that, compared to the size of the collection, the ISI collection has a much higher percentage of relevant documents per query (3.4%) than the CACM collection (0.5%). However, as already discussed, many of the relevant documents in the ISI collection have no terms in common with the respective information requests which usually yields low precision.

Related Test Collections

At the Virginia Polytechnic Institute and State University, Fox has assembled together nine small test collections in a CD-ROM. These test collections have sizes comparable to those of the CACM and ISI collections, but include their own particularities. Since they have been used throughout the years for evaluation of information retrieval systems, they provide a good setting for the preliminary testing of information retrieval algorithms. A list of these nine test collections is provided in Table 3.4.

3.3.3 The Cystic Fibrosis Collection

The cystic fibrosis (CF) collection [721] is composed of 1239 documents indexed with the term 'cystic fibrosis' in the National Library of Medicine's MEDLINE database. Each document contains the following fields:

Collection	*Subject*	*Num. Docs*	*Num. Queries*
ADI	Information Science	82	35
CACM	Computer Science	3200	64
ISI	Library Science	1460	76
CRAN	Aeronautics	1400	225
LISA	Library Science	6004	35
MED	Medicine	1033	30
NLM	Medicine	3078	155
NPL	Elec. Engineering	11,429	100
TIME	General Articles	423	83

Table 3.4 Test collections related to the CACM and ISI collections.

- MEDLINE accession number
- author
- title
- source
- major subjects
- minor subjects
- abstract (or extract)
- references
- citations.

The collection also includes 100 information requests (generated by an expert with two decades of clinical and research experience with cystic fibrosis) and the documents relevant to each query. Further, 4 separate relevance scores are provided for each relevant document. These relevance scores can be 0 (which indicates non-relevance), 1 (which indicates marginal relevance), and 2 (which indicates high relevance). Thus, the overall relevance score for a document (relative to a given query) varies from 0 to 8. Three of the relevance scores were provided by subject experts while the fourth relevance score was provided by a medical bibliographer.

Table 3.5 provides some statistics regarding the information requests in the CF collection. We notice that the number of queries with at least one relevant document is close to the total number of queries in the collection. Further, for various relevance thresholds (the minimum value of relevance score used to characterize relevance), the average number of relevant documents per query is between 10 and 30.

The CF collection, despite its small size, has two important characteristics. First, its set of relevance scores was generated directly by human experts through a careful evaluation strategy. Second, it includes a good number of information requests (relative to the collection size) and, as a result, the respective query vectors present overlap among themselves. This allows experimentation

Relevance Threshold	*Queries At Least 1 Rel. Doc*	*Min. Num. Rel. Docs*	*Max. Num. Rel. Docs*	*Avg. Num. Rel. Docs*
1	100	2	189	31.9
2	100	1	130	18.1
3	99	1	119	14.9
4	99	1	114	14.1
5	99	1	93	10.7
6	94	1	53	6.4

Table 3.5 Summary statistics for the information requests in the CF collection.

with retrieval strategies which take advantage of past query sessions to improve retrieval performance.

3.4 Trends and Research Issues

A major trend today is research in interactive user interfaces. The motivation is a general belief that effective retrieval is highly dependent on obtaining proper feedback from the user. Thus, evaluation studies of interactive interfaces will tend to become more common in the near future. The main issues revolve around deciding which evaluation measures are most appropriate in this scenario. A typical example is the informativeness measure [754] introduced in 1992.

Furthermore, the proposal, the study, and the characterization of alternative measures to recall and precision, such as the harmonic mean and the E measures, continue to be of interest.

3.5 Bibliographic Discussion

A nice chapter on retrieval performance evaluation appeared in the book by Salton and McGill [698]. Even if outdated, it is still interesting reading. The book by Khorfage [451] also includes a full chapter on retrieval evaluation. A recent paper by Mizzaro [569] presents a very complete survey of relevance studies throughout the years. About 160 papers are discussed in this paper.

Two recent papers by Shaw, Burgin, and Howel [422, 423] discuss standards and evaluations in test collections for cluster-based and vector-based retrieval models. These papers also discuss the advantages of the harmonic mean (of recall and precision) as a single alternative measure for recall and precision. Problems with recall and precision related to systems which require a weak document ordering are discussed by Raghavan, Bollmann, and Jung [664, 663]. Tague-Sutcliffe proposes a measure of informativeness for evaluating interactive user sessions [754].

Our discussion of the TREC collection is based on the papers by Harman [342] and by Vorhees and Harman [794]. The TREC collection is the most important reference collection nowadays for evaluation of complex information requests which execute on a large collection. Our coverage of the CACM and ISI collections is based on the work by Fox [272]. These collections are small, require short setup time, and provide a good environment for testing retrieval algorithms which are based on information derived from cross-citing patterns — a topic which has attracted much attention in the past [94, 435, 694, 730, 732, 809] and which might flourish again in the context of the Web. The discussion on the Cystic Fibrosis (CF) collection is based on the work by Shaw, Wood, Wood, and Tibbo [721]. The CF collection is also small but includes a set of relevance scores carefully generated by human experts. Furthermore, its example information requests present overlap among themselves which allows the testing of retrieval algorithms that take advantage of past user sessions to improve retrieval performance.

Chapter 4
Query Languages

with Gonzalo Navarro

4.1 Introduction

We cover in this chapter the different kinds of queries normally posed to text retrieval systems. This is in part dependent on the retrieval model the system adopts, i.e., a full-text system will not answer the same kinds of queries as those answered by a system based on keyword ranking (as Web search engines) or on a hypertext model. In Chapter 8 we explain *how* the user queries are solved, while in this chapter we show *which* queries can be formulated. The type of query the user might formulate is largely dependent on the underlying information retrieval model. The different models for text retrieval systems are covered in Chapter 2.

As in previous chapters, we want to distinguish between information retrieval and data retrieval, as we use this dichotomy to classify different query languages. We have chosen to distinguish first languages that allow the answer to be ranked, that is, languages for information retrieval. As covered in Chapter 2, for the basic information retrieval models, keyword-based retrieval is the main type of querying task. For query languages not aimed at information retrieval, the concept of ranking cannot be easily defined, so we consider them as languages for data retrieval. Furthermore, some query languages are not intended for final users and can be viewed as languages that a higher level software package should use to query an on-line database or a CD-ROM archive. In that case, we talk about *protocols* rather than query languages. Depending on the user experience, a different query language will be used. For example, if the user knows exactly what he wants, the retrieval task is easier and ranking may not even be needed.

An important issue is that most query languages try to use the content (i.e., the semantics) and the structure of the text (i.e., the text syntax) to find relevant documents. In that sense, the system may fail to find the relevant answers (see Chapter 3). For this reason, a number of techniques meant to enhance the usefulness of the queries exist. Examples include the expansion of a word to the set of its synonyms or the use of a thesaurus and stemming to

put together all the derivatives of the same word. Moreover, some words which are very frequent and do not carry meaning (such as 'the'), called *stopwords*, may be removed. This subject is covered in Chapter 7. Here we assume that all the query preprocessing has already been done. Although these operations are usually done for information retrieval, many of them can also be useful in a data retrieval context. When we want to emphasize the difference between words that can be retrieved by a query and those which cannot, we call the former 'keywords.'

Orthogonal to the kind of queries that can be asked is the subject of the *retrieval unit* the information system adopts. The retrieval unit is the basic element which can be retrieved as an answer to a query (normally a set of such basic elements is retrieved, sometimes ranked by relevance or other criterion). The retrieval unit can be a file, a document, a Web page, a paragraph, or some other structural unit which contains an answer to the search query. From this point on, we will simply call those retrieval units 'documents,' although as explained this can have different meanings (see also Chapter 2).

This chapter is organized as follows. We first show the queries that can be formulated with keyword-based query languages. They are aimed at information retrieval, including simple words and phrases as well as Boolean operators which manipulate sets of documents. In the second section we cover pattern matching, which includes more complex queries and is generally aimed at complementing keyword searching with more powerful data retrieval capabilities. Third, we cover querying on the structure of the text, which is more dependent on the particular text model. Finally, we finish with some standard protocols used on the Internet and by CD-ROM publishers.

4.2 Keyword-Based Querying

A query is the formulation of a user information need. In its simplest form, a query is composed of keywords and the documents containing such keywords are searched for. Keyword-based queries are popular because they are intuitive, easy to express, and allow for fast ranking. Thus, a query can be (and in many cases is) simply a word, although it can in general be a more complex combination of operations involving several words.

In the rest of this chapter we will refer to single-word and multiple-word queries as *basic queries*. Patterns, which are covered in section 4.3, are also considered as basic queries.

4.2.1 Single-Word Queries

The most elementary query that can be formulated in a text retrieval system is a word. Text documents are assumed to be essentially long sequences of words. Although some models present a more general view, virtually all models allow us

to see the text in this perspective and to search words. Some models are also able to see the internal division of words into letters. These latter models permit the searching of other types of patterns, which are covered in section 4.3. The set of words retrieved by these extended queries can then be fed into the word-treating machinery, say to perform thesaurus expansion or for ranking purposes.

A word is normally defined in a rather simple way. The alphabet is split into 'letters' and 'separators,' and a word is a sequence of letters surrounded by separators. More complex models allow us to specify that some characters are not letters but do not split a word, e.g. the hyphen in '`on-line`.' It is good practice to leave the choice of what is a letter and what is a separator to the manager of the text database.

The division of the text into words is not arbitrary, since words carry a lot of meaning in natural language. Because of that, many models (such as the vector model) are completely structured on the concept of words, and words are the only type of queries allowed (moreover, some systems only allow a small set of words to be extracted from the documents). The result of word queries is the set of documents containing at least one of the words of the query. Further, the resulting documents are ranked according to a degree of similarity to the query. To support ranking, two common statistics on word occurrences inside texts are commonly used: 'term frequency' which counts the number of times a word appears inside a document and 'inverse document frequency' which counts the number of documents in which a word appears. See Chapter 2 for more details.

Additionally, the exact positions where a word appears in the text may be required for instance, by an interface which highlights each occurrence of that word.

4.2.2 Context Queries

Many systems complement single-word queries with the ability to search words in a given *context*, that is, near other words. Words which appear near each other may signal a higher likelihood of relevance than if they appear apart. For instance, we may want to form phrases of words or find words which are proximal in the text. Therefore, we distinguish two types of queries:

- **Phrase** is a sequence of single-word queries. An occurrence of the phrase is a sequence of words. For instance, it is possible to search for the word '`enhance`,' and then for the word '`retrieval`.' In phrase queries it is normally understood that the separators in the text need not be the same as those in the query (e.g., two spaces versus one space), and uninteresting words are not considered at all. For instance, the previous example could match a text such as '...enhance the retrieval...'. Although the notion of a phrase is a very useful feature in most cases, not all systems implement it.
- **Proximity** A more relaxed version of the phrase query is the proximity query. In this case, a sequence of single words or phrases is given, together

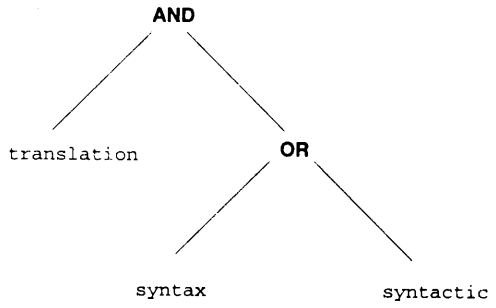

Figure 4.1 An example of a query syntax tree. It will retrieve all the documents which contain the word '**translation**' as well as either the word '**syntax**' or the word '**syntactic**'.

with a maximum allowed distance between them. For instance, the above example could state that the two words should occur within four words, and therefore a match could be '**...enhance the power of retrieval...**' This distance can be measured in characters or words depending on the system. The words and phrases may or may not be required to appear in the same order as in the query.

Phrases can be ranked in a fashion somewhat analogous to single words (see Chapters 2 and 5 for details). Proximity queries can be ranked in the same way if the parameters used by the ranking technique do not depend on physical proximity. Although it is not clear how to do better ranking, physical proximity has semantic value. This is because in most cases the proximity means that the words are in the same paragraph, and hence related in some way.

4.2.3 Boolean Queries

The oldest (and still heavily used) form of combining keyword queries is to use Boolean operators. A *Boolean query* has a syntax composed of *atoms* (i.e., basic queries) that retrieve documents, and of *Boolean operators* which work on their operands (which are sets of documents) and deliver sets of documents. Since this scheme is in general *compositional* (i.e., operators can be composed over the results of other operators), a *query syntax tree* is naturally defined, where the leaves correspond to the basic queries and the internal nodes to the operators. The query syntax tree operates on an algebra over sets of documents (and the final answer of the query is also a set of documents). This is much as, for instance, the syntax trees of arithmetic expressions where the numbers and variables are the leaves and the operations form the internal nodes. Figure 4.1 shows an example.

The operators most commonly used, given two basic queries or Boolean

subexpressions e_1 and e_2, are:

- **OR** The query (e_1 OR e_2) selects all documents which satisfy e_1 or e_2. Duplicates are eliminated.
- **AND** The query (e_1 AND e_2) selects all documents which satisfy both e_1 and e_2.
- **BUT** The query (e_1 BUT e_2) selects all documents which satisfy e_1 but not e_2. Notice that classical Boolean logic uses a NOT operation, where (NOT e_2) is valid whenever e_2 is not. In this case all documents not satisfying e_2 should be delivered, which may retrieve a huge amount of text and is probably not what the user wants. The BUT operator, instead, restricts the universe of retrievable elements to the result of e_1.†

Besides selecting the appropriate documents, the IR system may also sort the documents by some criterion, highlight the occurrences within the documents of the words mentioned in the query, and allow feedback by taking the answer set as a basis to reformulate the query.

With classic Boolean systems, no ranking of the retrieved documents is normally provided. A document either satisfies the Boolean query (in which case it is retrieved) or it does not (in which case it is not retrieved). This is quite a limitation because it does not allow for partial matching between a document and a user query. To overcome this limitation, the condition for retrieval must be relaxed. For instance, a document which partially satisfies an AND condition might be retrieved.

In fact, it is widely accepted that users not trained in mathematics find the meaning of Boolean operators difficult to grasp. With this problem in mind, a 'fuzzy Boolean' set of operators has been proposed. The idea is that the meaning of AND and OR can be relaxed, such that instead of forcing an element to appear in *all* the operands (AND) or at least in *one* of the operands (OR), they retrieve elements appearing in *some* operands (the AND may require it to appear in more operands than the OR). Moreover, the documents are ranked higher when they have a larger number of elements in common with the query (see Chapter 2).

4.2.4 Natural Language

Pushing the fuzzy Boolean model even further, the distinction between AND and OR can be completely blurred, so that a query becomes simply an enumeration of words and context queries. All the documents matching a portion of the user query are retrieved. Higher ranking is assigned to those documents matching more parts of the query. The negation can be handled by letting the user express

† Notice that the same problem arises in the relational calculus, which is shown similar to the relational algebra only when 'unsafe' expressions are avoided. Unsafe expressions are those that make direct or indirect reference to a universe of elements, as NOT does.

that some words are not desired, so that the documents containing them are penalized in the ranking computation. A threshold may be selected so that the documents with very low weights are not retrieved. Under this scheme we have completely eliminated any reference to Boolean operations and entered into the field of natural language queries. In fact, one can consider that Boolean queries are a simplified abstraction of natural language queries.

A number of new issues arise once this model is used, especially those related to the proper way to rank an element with respect to a query. The search criterion can be re-expressed using a different model, where documents and queries are considered just as a vector of 'term weights' (with one coordinate per interesting keyword or even per existing text word) and queries are considered in exactly the same way (context queries are not considered in this case). Therefore, the query is now internally converted into a vector of term weights and the aim is to retrieve all the vectors (documents) which are *close* to the query (where closeness has to be defined in the model). This allows many interesting possibilities, for instance a complete document can be used as a query (since it is also a vector), which naturally leads to the use of relevance feedback techniques (i.e., the user can select a document from the result and submit it as a new query to retrieve documents similar to the selected one). The algorithms for this model are totally different from those based on searching patterns (it is even possible that not every text word needs to be searched but only a small set of hopefully representative keywords extracted from each document). Natural language querying is also covered in Chapter 14.

4.3 Pattern Matching

In this section we discuss more specific query formulations (based on the concept of a *pattern*) which allow the retrieval of pieces of text that have some property. These data retrieval queries are useful for linguistics, text statistics, and data extraction. Their result can be fed into the composition mechanism described above to form phrases and proximity queries, comprising what we have called *basic queries*. Basic queries can be combined using Boolean expressions. In this sense we can view these data retrieval capabilities as enhanced tools for information retrieval. However, it is more difficult to rank the result of a pattern matching expression.

A *pattern* is a set of syntactic features that must occur in a text segment. Those segments satisfying the pattern specifications are said to 'match' the pattern. We are interested in documents containing segments which match a given search pattern. Each system allows the specification of some types of patterns, which range from very simple (for example, words) to rather complex (such as regular expressions). In general, as more powerful is the set of patterns allowed, more involved are the queries that the user can formulate and more complex is the implementation of the search. The most used types of patterns are:

- **Words** A string (sequence of characters) which must be a word in the text (see section 4.2). This is the most basic pattern.
- **Prefixes** A string which must form the beginning of a text word. For instance, given the prefix 'comput' all the documents containing words such as 'computer,' 'computation,' 'computing,' etc. are retrieved.
- **Suffixes** A string which must form the termination of a text word. For instance, given the suffix 'ters' all the documents containing words such as 'computers,' 'testers,' 'painters,' etc. are retrieved.
- **Substrings** A string which can appear within a text word. For instance, given the substring 'tal' all the documents containing words such as 'coastal,' 'talk,' 'metallic,' etc. are retrieved. This query can be restricted to find the substrings inside words, or it can go further and search the substring anywhere in the text (in this case the query is not restricted to be a sequence of letters but can contain word separators). For instance, a search for 'any flow' will match in the phrase '...many flowers....'
- **Ranges** A pair of strings which matches any word lying between them in lexicographical order. Alphabets are normally sorted, and this induces an order into the strings which is called *lexicographical order* (this is indeed the order in which words in a dictionary are listed). For instance, the range between words 'held' and 'hold' will retrieve strings such as 'hoax' and 'hissing.'
- **Allowing errors** A word together with an error threshold. This search pattern retrieves all text words which are 'similar' to the given word. The concept of similarity can be defined in many ways. The general concept is that the pattern or the text may have errors (coming from typing, spelling, or from optical character recognition software, among others), and the query should try to retrieve the given word and what are likely to be its erroneous variants. Although there are many models for similarity among words, the most generally accepted in text retrieval is the *Levenshtein distance*, or simply *edit distance*. The edit distance between two strings is the minimum number of character insertions, deletions, and replacements needed to make them equal (see Chapter 6). Therefore, the query specifies the maximum number of allowed errors for a word to match the pattern (i.e., the maximum allowed edit distance). This model can also be extended to search substrings (not only words), retrieving any text segment which is at the allowed edit distance from the search pattern. Under this extended model, if a typing error splits 'flower' into 'flo wer' it could still be found with one error, while in the restricted case of words it could not (since neither 'flo' nor 'wer' are at edit distance 1 from 'flower'). Variations on this distance model are of use in computational biology for searching on DNA or protein sequences as well as in signal processing.
- **Regular expressions** Some text retrieval systems allow searching for *regular expressions*. A regular expression is a rather general pattern built

up by simple strings (which are meant to be matched as substrings) and the following operators:

- union: if e_1 and e_2 are regular expressions, then $(e_1|e_2)$ matches what e_1 or e_2 matches.
- concatenation: if e_1 and e_2 are regular expressions, the occurrences of $(e_1\ e_2)$ are formed by the occurrences of e_1 immediately followed by those of e_2 (therefore simple strings can be thought of as a concatenation of their individual letters).
- repetition: if e is a regular expression, then (e^*) matches a sequence of zero or more contiguous occurrences of e.

For instance, consider a query like '`pro (blem | tein) (s | ε) (0 | 1 | 2)*`' (where ε denotes the empty string). It will match words such as '`problem02`' and '`proteins.`' As in previous cases, the matches can be restricted to comprise a whole word, to occur inside a word, or to match an arbitrary text segment. This can also be combined with the previous type of patterns to search a regular expression allowing errors.

- **Extended patterns** It is normal to use a more user-friendly query language to represent some common cases of regular expressions. Extended patterns are subsets of the regular expressions which are expressed with a simpler syntax. The retrieval system can internally convert extended patterns into regular expressions, or search them with specific algorithms. Each system supports its own set of extended patterns, and therefore no formal definition exists. Some examples found in many new systems are:

  - classes of characters, i.e. one or more positions within the pattern are matched by any character from a pre-defined set. This involves features such as case-insensitive matching, use of ranges of characters (e.g., specifying that some character must be a digit), complements (e.g., some character must not be a letter), enumeration (e.g., a character must be a vowel), wild cards (i.e., a position within the pattern matches with anything), among others.
  - conditional expressions, i.e., a part of the pattern may or may not appear.
  - wild characters which match any sequence in the text, e.g. any word which starts as '`flo`' and ends with '`ers`,' which matches '`flowers`' as well as '`flounders.`'
  - combinations that allow some parts of the pattern to match exactly and other parts with errors.

4.4 Structural Queries

Up to now we have considered the text collection as a set of documents which can be queried with regard to their text content. This model is unable to take advantage of novel text features which are becoming commonplace, such as the

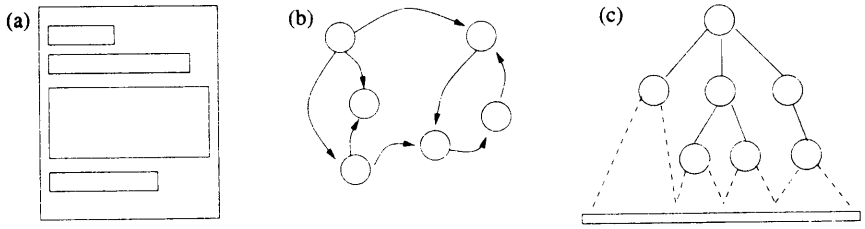

Figure 4.2 The three main structures: (*a*) form-like fixed structure, (*b*) hypertext structure, and (*c*) hierarchical structure.

text structure. The text collections tend to have some structure built into them, and allowing the user to query those texts based on their structure (and not only their content) is becoming attractive. The standardization of languages to represent structured texts such as HTML has pushed forward in this direction (see Chapter 6).

As discussed in Chapter 2, mixing contents and structure in queries allows us to pose very powerful queries, which are much more expressive than each query mechanism by itself. By using a query language that integrates both types of queries, the retrieval quality of textual databases can be improved.

This mechanism is built on top of the basic queries, so that they select a set of documents that satisfy certain constraints on their content (expressed using words, phrases, or patterns that the documents must contain). On top of this, some structural constraints can be expressed using containment, proximity, or other restrictions on the structural elements (e.g., chapters, sections, etc.) present in the documents. The Boolean queries can be built on top of the structural queries, so that they combine the sets of documents delivered by those queries. In the Boolean syntax tree (recall the example of Figure 4.1) the structural queries form the leaves of the tree. On the other hand, structural queries can themselves have a complex syntax.

We divide this section according to the type of structures found in text databases. Figure 4.2 illustrates them. Although structured query languages should be amenable for ranking, this is still an open problem.

In what follows it is important to distinguish the difference between the structure that a text may *have* and what can be *queried* about that structure. In general, natural language texts may have any desired structure. However, different models allow the querying of only some aspects of the real structure. When we say that the structure allowed is restricted in some way, we mean that only the aspects which follow this restriction can be queried, albeit the text may have more structural information. For instance, it is possible that an article has a nested structure of sections and subsections, but the query model does not accept recursive structures. In this case we will not be able to query for sections included in others, although this may be the case in the texts documents under consideration.

4.4.1 Fixed Structure

The structure allowed in texts was traditionally quite restrictive. The documents had a fixed set of *fields*, much like a filled form. Each field had some text inside. Some fields were not present in all documents. Only rarely could the fields appear in any order or repeat across a document. A document could not have text not classified under any field. Fields were not allowed to nest or overlap. The retrieval activity allowed on them was restricted to specifying that a given basic pattern was to be found only in a given field. Most current commercial systems use this model.

This model is reasonable when the text collection has a fixed structure. For instance, a mail archive could be regarded as a set of mails, where each mail has a sender, a receiver, a date, a subject, and a body field. The user can thus search for the mails sent to a given person with `'football'` in the subject field. However, the model is inadequate to represent the hierarchical structure present in an HTML document, for instance.

If the division of the text into fields is rigid enough, the content of some fields can even be interpreted not as text but as numbers, dates, etc. thereby allowing different queries to be posed on them (e.g., month ranges in dates). It is not hard to see that this idea leads naturally to the relational model, each field corresponding to a column in the database table. Looking at the database as a text allows us to query the textual fields with much more power than is common in relational database systems. On the other hand, relational databases may make better use of their knowledge on the data types involved to build specialized and more efficient indices. A number of approaches towards combining these trends have been proposed in recent years, their main problem being that they do not achieve optimal performance because the text is usually stored together with other types of data. Nevertheless, there are several proposals that extend SQL (Structured Query Language) to allow full-text retrieval. Among them we can mention proposals by leading relational database vendors such as Oracle and Sybase, as well as SFQL, which is covered in section 4.5.

4.4.2 Hypertext

Hypertexts probably represent the maximum freedom with respect to structuring power. A hypertext is a directed graph where the nodes hold some text and the links represent connections between nodes or between positions inside the nodes (see Chapter 2). Hypertexts have received a lot of attention since the explosion of the Web, which is indeed a gigantic hypertext-like database spread across the world.

However, retrieval from a hypertext began as a merely navigational activity. That is, the user had to manually traverse the hypertext nodes following links to search what he wanted. It was not possible to query the hypertext based on its structure. Even in the Web one can search by the text contents of the nodes, but not by their structural connectivity.

An interesting proposal to combine browsing and searching on the Web is WebGlimpse. It allows classical navigation plus the ability to search by content in the neighborhood of the current node. Currently, some query tools have appeared that achieve the goal of querying hypertexts based on their content and their structure. This problem is covered in detail in Chapter 13.

4.4.3 Hierarchical Structure

An intermediate structuring model which lies between fixed structure and hypertext is the hierarchical structure. This model represents a recursive decomposition of the text and is a natural model for many text collections (e.g., books, articles, legal documents, structured programs, etc.). Figure 4.3 shows an example of such a hierarchical structure.

The simplification from hypertext to a hierarchy allows the adoption of faster algorithms to solve queries. As a general rule, the more powerful the model, the less efficiently it can be implemented.

Our aim in this section is to analyze and discuss the different approaches presented by the hierarchical models. We first present a selection of the most representative models and then discuss the main subjects of this area.

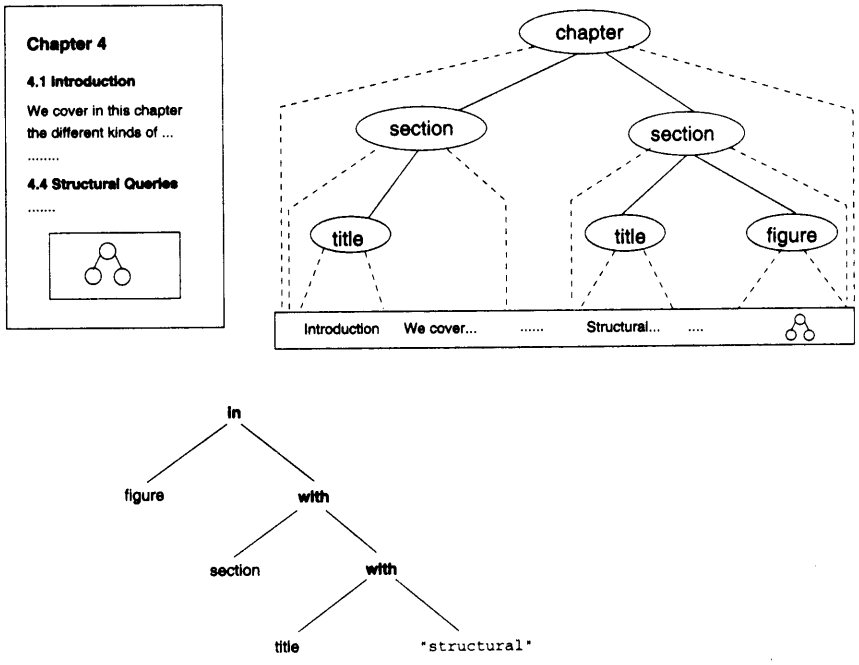

Figure 4.3 An example of a hierarchical structure: the page of a book, its schematic view, and a parsed query to retrieve the figure.

A Sample of Hierarchical Models

PAT Expressions

These are built on the same index as the text index, i.e. there is no special separate index on the structure. The structure is assumed to be marked in the text by *tags* (as in HTML), and therefore is defined in terms of initial and final tags. This allows a dynamic scheme where the structure of interest is not fixed but can be determined at query time. For instance, since tags need not to be especially designed as normal tags, one can define that the end-of-lines are the marks in order to define a structure on lines. This also allows for a very efficient implementation and no additional space overhead for the structure.

Each pair of initial and final tags defines a *region*, which is a set of contiguous text areas. Externally computed regions are also supported. However, the areas of a region cannot nest or overlap, which is quite restrictive. There is no restriction on areas of different regions.

Apart from text searching operations, it is possible to select areas containing (or not) other areas, contained (or not) in other areas, or followed (or not) by other areas.

A disadvantage is that the algebra mixes regions and sets of text positions which are incompatible and force complex conversion semantics. For instance, if the result of a query is going to generate overlapping areas (a fact that cannot be determined beforehand) then the result is converted to positions. Also, the dynamic definition of regions is flexible but requires the structure to be expressable using tags (also called 'markup', see Chapter 6), which for instance does not occur in some structured programming languages.

Overlapped Lists

These can be seen as an evolution of PAT Expressions. The model allows for the areas of a region to overlap, but not to nest. This elegantly solves the problems of mixing regions and sets of positions. The model considers the use of an inverted list (see Chapter 8) where not only the words but also the regions are indexed.

Apart from the operations of PAT Expressions, the model allows us to perform set union, and to combine regions. Combination means selecting the minimal text areas which include any two areas taken from two regions. A 'followed by' operator imposes the additional restriction that the first area must be before the second one. An 'n words' operator generates the region of all (overlapping) sequences of n words of the text (this is further used to retrieve elements close to each other). If an operation produces a region with nested areas, only the minimal areas are selected. An example is shown in Figure 2.11.

The implementation of this model can also be very efficient. It is not clear, however, whether overlapping is good or not for capturing the structural properties that information has in practice. A new proposal allows the structure to be nested *and* overlapped, showing that more interesting operators can still be implemented.

Lists of References

These are an attempt to make the definition and querying of structured text uniform, using a common language. The language goes beyond querying structured text, so we restrict our attention to the subset in which we are interested.

The structure of documents is fixed and hierarchical, which makes it impossible to have overlapping results. All possible regions are defined at indexing time. The answers delivered are more restrictive, since nesting is not allowed (only the top-level elements qualify) and all elements must be of the same type, e.g. only sections, or only paragraphs. In fact, there are also hypertext links but these cannot be queried (the model also has navigational features).

A static hierarchical structure makes it possible to speak in terms of direct ancestry of nodes, a concept difficult to express when the structure is dynamic. The language allows for querying on 'path expressions,' which describe paths in the structure tree.

Answers to queries are seen as lists of 'references.' A reference is a pointer to a region of the database. This integrates in an elegant way answers to queries and hypertext links, since all are lists of references.

Proximal Nodes

This model tries to find a good compromise between expressiveness and efficiency. It does not define a specific language, but a model in which it is shown that a number of useful operators can be included achieving good efficiency.

The structure is fixed and hierarchical. However, many independent structures can be defined on the same text, each one being a strict hierarchy but allowing overlaps between areas of different hierarchies. An example is shown in Figure 2.12.

A query can relate different hierarchies, but returns a subset of the nodes of one hierarchy only (i.e., nested elements are allowed in the answers, but no overlaps). Text matching queries are modeled as returning nodes from a special 'text hierarchy.'

The model specifies a fully compositional language where the leaves of the query syntax tree are formed by basic queries on contents or names of structural elements (e.g., all chapters). The internal nodes combine results. For efficiency, the operations defined at the internal nodes must be implementable looking at the identity and text areas of the operands, and must relate nodes which are close in the text.

It has been shown that many useful operators satisfy this restriction: selecting elements that (directly or transitively) include or are included in others; that are included at a given position (e.g., the third paragraph of each chapter); that are shortly before or after others; set manipulation; and many powerful variations. Operations on content elements deliver a set of regions with no nesting, and those results can be fully integrated into any query. This ability to integrate the text into the model is very useful. On the other hand, some queries requiring non-proximal operations are not allowed, for instance *semijoins*. An example of a semijoin is 'give me the titles of all the chapters referenced in this chapter.'

Tree Matching
This model relies on a single primitive: tree inclusion, whose main idea is as follows. Interpreting the structure both of the text database and of the query (which is defined as a pattern on the structure) as trees, determine an embedding of the query into the database which respects the hierarchical relationships between nodes of the query.

Two variants are studied. *Ordered inclusion* forces the embedding to respect the left-to-right relations among siblings in the query, while *unordered inclusion* does not. The leaves of the query can be not only structural elements but also text patterns, meaning that the ancestor of the leaf must contain that pattern.

Simple queries return the roots of the matches. The language is enriched by Prolog-like variables, which can be used to express requirements on equality between parts of the matched substructure and to retrieve another part of the match, not only the root. Logical variables are also used for union and intersection of queries, as well as to emulate tuples and join capabilities.

Although the language is set oriented, the algorithms work by sequentially obtaining each match. The use of logical variables and unordered inclusion makes the search problem intractable (NP-hard in many cases). Even the good cases have an inefficient solution in practice.

Discussion

A survey of the main hierarchical models raises a number of interesting issues, most of them largely unresolved up to now. Some of them are listed below.

Static or dynamic structure
As seen, in a static structure there are one or more explicit hierarchies (which can be queried, e.g., by ancestry), while in a dynamic structure there is not really a hierarchy, but the required elements are built on the fly. A dynamic structure is implemented over a normal text index, while a static one may or may not be. A static structure is independent of the text markup, while a dynamic one is more flexible for building arbitrary structures.

Restrictions on the structure
The text or the answers may have restrictions about nesting and/or overlapping. In some cases these restrictions exist for efficiency reasons. In other cases, the query language is restricted to avoid restricting the structure. This choice is largely dependent on the needs of each application.

Integration with text
In many structured models, the text content is merely seen as a secondary source of information which is used only to restrict the matches of structural elements. In classic IR models, on the other side, information on the structure is the secondary element which is used only to restrict text matches. For an effective

integration of queries on text content with queries on text structure, the query language must provide for full expressiveness of both types of queries and for effective means of combining them.

Query language
Typical queries on structure allow the selection of areas that contain (or not) other areas, that are contained (or not) in other areas, that follow (or are followed by) other areas, that are close to other areas, and set manipulation. Many of them are implemented in most models, although each model has unique features. Some kind of standardization, expressiveness taxonomy, or formal categorization would be highly desirable but does not exist yet.

4.5 Query Protocols

In this section we briefly cover some query languages that are used automatically by software applications to query text databases. Some of them are proposed as standards for querying CD-ROMs or as intermediate languages to query library systems. Because they are not intended for human use, we refer to them as protocols rather than languages. More information on protocols can be found in Chapters 14 and 15. The most important query protocols are:

- **Z39.50** is a protocol approved as a standard in 1995 by ANSI and NISO. This protocol is intended to query bibliographical information using a standard interface between the client and the host database manager which is independent of the client user interface and of the query database language at the host. The database is assumed to be a text collection with some fixed fields (although it is more flexible than usual). The Z39.50 protocol is used broadly and is part, for instance, of WAIS (see below). The protocol does not only specify the query language and its semantics, but also the way in which client and server establish a session, communicate and exchange information, etc. Although originally conceived only to operate on bibliographical information (using the Machine Readable Cataloging Record (MARC) format), it has been extended to query other types of information as well.
- **WAIS** (Wide Area Information Service) is a suite of protocols that was popular at the beginning of the 1990s before the boom of the Web. The goal of WAIS was to be a network publishing protocol and to be able to query databases through the Internet.

In the CD-ROM publishing arena, there are several proposals for query protocols. The main goal of these protocols is to provide 'disk interchangeability.' This means more flexibility in data communication between primary information providers and end users. It also enables significant cost savings since it allows access to diverse information without the need to buy, install, and train users for different data retrieval applications. We briefly cover three of these proposals:

- **CCL** (Common Command Language) is a NISO proposal (Z39.58 or ISO 8777) based on Z39.50. It defines 19 commands that can be used interactively. It is more popular in Europe, although very few products use it. It is based on the classical Boolean model.
- **CD-RDx** (Compact Disk Read only Data exchange) uses a client-server architecture and has been implemented in most platforms. The client is generic while the server is designed and provided by the CD-ROM publisher who includes it with the database in the CD-ROM. It allows fixed-length fields, images, and audio, and is supported by such US national agencies as the CIA, NASA, and GSA.
- **SFQL** (Structured Full-text Query Language) is based on SQL and also has a client-server architecture. SFQL has been adopted as a standard by the aerospace community (the Air Transport Association/Aircraft Industry Association). Documents are rows in a relational table and can be tagged using SGML. The language defines the format of the answer, which has a header and a variable length message area. The language does not define any specific formatting or markup. For example, a query in SFQL is:

  ```
  Select abstract from journal.papers where title
  contains "text search"
  ```

  The language supports Boolean and logical operators, thesaurus, proximity operations, and some special characters such as wild cards and repetition. For example:

  ```
  ... where paper contains "retrieval" or like "info
  %" and date > 1/1/98
  ```

  Compared with CCL or CD-RDx, SFQL is more general and flexible, although it is based on a relational model, which is not always the best choice for a document database.

4.6 Trends and Research Issues

We reviewed in this chapter the main aspects of the query languages that retrieve information from textual databases. Our discussion covered from the most classic tools to the most novel capabilities that are emerging, from searching words to extended patterns, from the Boolean model to querying structures. Table 4.1 shows the different basic queries allowed in the different models. Although the probabilistic and the Bayesian belief network (BBN) models are based on word queries, they can incorporate set operations.

We present in Figure 4.4 the types of operations we covered and how they can be structured (not all of them exist in all models and not all of them have to be used to form a query). The figure shows, for instance, that we can form a query using Boolean operations over phrases (skipping structural queries), which

Model	Queries allowed
Boolean	word, set operations
Vector	words
Probabilistic	words
BBN	words

Table 4.1 Relationship between types of queries and models.

can be formed by words and by regular expressions (skipping the ability to allow errors).

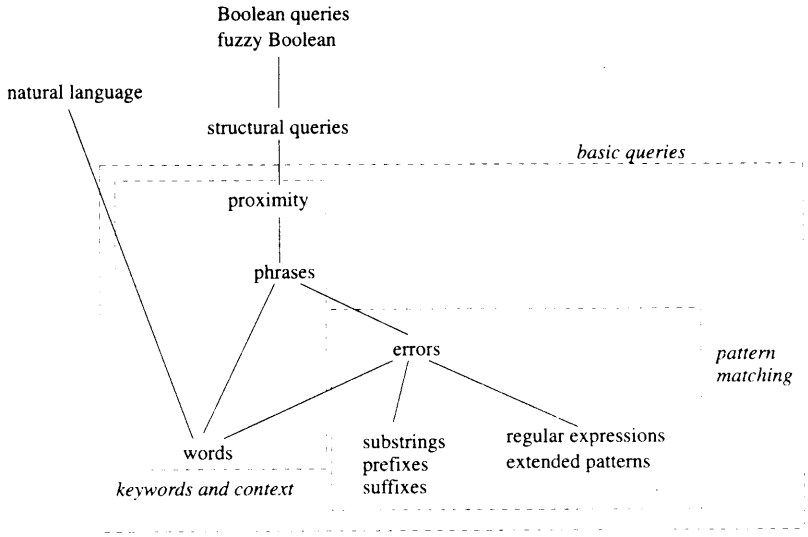

Figure 4.4 The types of queries covered and how they are structured.

The area of query languages for text databases is definitely moving towards higher flexibility. While text models are moving towards the goal of achieving a better understanding of the user needs (by providing relevance feedback, for instance), the query languages are allowing more and more power in the specification of the query. While extended patterns and searching allowing errors permit us to find patterns without complete knowledge of what is wanted, querying on the structure of the text (and not only on its content) provides greater expressiveness and increased functionality.

Another important research topic is visual query languages. Visual metaphors can help non-experienced users to pose complex Boolean queries. Also, a visual query language can include the structure of the document. This topic is related to user interfaces and visualization and is covered in Chapter 10.

4.7 Bibliographic Discussion

The material on classical query languages (most simple patterns, Boolean model, and fixed structure) is based on current commercial systems, such as Fulcrum, Verity, and others, as well as on non-commercial systems such as Glimpse [540] and Igrep [26].

The fuzzy Boolean model is described in [703]. The Levenshtein distance is described in [504] and [25]. Soundex is explained in [445]. A comparison of the effectiveness of different similarity models is given in [595]. A good source on regular expressions is [375]. A rich language on extended patterns is described in [837].

A classical reference on hypertext is [181]. The WebGlimpse system is presented in [539]. The discussion of hierarchical text is partially based on [41]. The original proposals are: PAT Expressions [693], Overlapped Lists [173] and the new improved proposal [206], Lists of References [534], Proximal Nodes [590], and Tree Matching [439]. PAT Expressions are the basic model of the PAT Text Searching System [309]. A simple structured text model is presented in [36] and a visual query language that includes structure is discussed in [44].

More information on Z39.50 can be obtained from [23]. More information on WAIS is given in [425]. For details on SFQL see [392].

Chapter 5
Query Operations

5.1 Introduction

Without detailed knowledge of the collection make-up and of the retrieval environment, most users find it difficult to formulate queries which are well designed for retrieval purposes. In fact, as observed with Web search engines, the users might need to spend large amounts of time reformulating their queries to accomplish effective retrieval. This difficulty suggests that the first query formulation should be treated as an initial (naive) attempt to retrieve relevant information. Following that, the documents initially retrieved could be examined for relevance and new improved query formulations could then be constructed in the hope of retrieving additional useful documents. Such query reformulation involves two basic steps: expanding the original query with new terms and reweighting the terms in the expanded query.

In this chapter, we examine a variety of approaches for improving the initial query formulation through query expansion and term reweighting. These approaches are grouped in three categories: (a) approaches based on feedback information from the user; (b) approaches based on information derived from the set of documents initially retrieved (called the *local* set of documents); and (c) approaches based on global information derived from the document collection. In the first category, user relevance feedback methods for the vector and probabilistic models are discussed. In the second category, two approaches for local analysis (i.e., analysis based on the set of documents initially retrieved) are presented. In the third category, two approaches for global analysis are covered.

Our discussion is not aimed at completely covering the area, neither does it intend to present an exhaustive survey of query operations. Instead, our discussion is based on a selected bibliography which, we believe, is broad enough to allow an overview of the main issues and tradeoffs involved in query operations. Local and global analysis are highly dependent on clustering algorithms. Thus, clustering is covered throughout our discussion. However, there is no intention of providing a complete survey of clustering algorithms for information retrieval.

5.2 User Relevance Feedback

Relevance feedback is the most popular query reformulation strategy. In a relevance feedback cycle, the user is presented with a list of the retrieved documents and, after examining them, marks those which are relevant. In practice, only the top 10 (or 20) ranked documents need to be examined. The main idea consists of selecting important terms, or expressions, attached to the documents that have been identified as relevant by the user, and of enhancing the importance of these terms in a new query formulation. The expected effect is that the new query will be moved towards the relevant documents and away from the non-relevant ones.

Early experiments using the Smart system [695] and later experiments using the probabilistic weighting model [677] have shown good improvements in precision for small test collections when relevance feedback is used. Such improvements come from the use of two basic techniques: query expansion (addition of new terms from relevant documents) and term reweighting (modification of term weights based on the user relevance judgement).

Relevance feedback presents the following main advantages over other query reformulation strategies: (a) it shields the user from the details of the query reformulation process because all the user has to provide is a relevance judgement on documents; (b) it breaks down the whole searching task into a sequence of small steps which are easier to grasp; and (c) it provides a controlled process designed to emphasize some terms (relevant ones) and de-emphasize others (non-relevant ones).

In the following three subsections, we discuss the usage of user relevance feedback to (a) expand queries with the vector model, (b) reweight query terms with the probabilistic model, and (c) reweight query terms with a variant of the probabilistic model.

5.2.1 Query Expansion and Term Reweighting for the Vector Model

The application of *relevance feedback* to the vector model considers that the term-weight vectors of the documents identified as relevant (to a given query) have similarities among themselves (i.e., relevant documents resemble each other). Further, it is assumed that non-relevant documents have term-weight vectors which are dissimilar from the ones for the relevant documents. The basic idea is to reformulate the query such that it gets closer to the term-weight vector space of the relevant documents.

Let us define some additional terminology regarding the processing of a given query q as follows,

D_r: set of relevant documents, as identified by the user, among the *retrieved* documents;
D_n: set of non-relevant documents among the *retrieved* documents;
C_r: set of relevant documents among all documents in the collection;

$|D_r|, |D_n|, |C_r|$: number of documents in the sets D_r, D_n, and C_r, respectively;
α, β, γ: tuning constants.

Consider first the unrealistic situation in which the complete set C_r of relevant documents to a given query q is known in advance. In such a situation, it can be demonstrated that the best query vector for distinguishing the relevant documents from the non-relevant documents is given by,

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j \tag{5.1}$$

The problem with this formulation is that the *relevant* documents which compose the set C_r are not known a priori. In fact, we are looking for them. The natural way to avoid this problem is to formulate an initial query and to incrementally change the initial query vector. This incremental change is accomplished by restricting the computation to the documents *known* to be relevant (according to the user judgement) at that point. There are three classic and similar ways to calculate the modified query $\vec{q}_m$ as follows,

$$\begin{aligned}
\textit{Standard_Rochio}: \quad \vec{q}_m &= \alpha\, \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
\textit{Ide_Regular}: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
\textit{Ide_Dec_Hi}: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma\ max_{non\text{-}relevant}(\vec{d}_j)
\end{aligned}$$

where $max_{non-relevant}(\vec{d}_j)$ is a reference to the highest ranked non-relevant document. Notice that now D_r and D_n stand for the sets of relevant and non-relevant documents (among the retrieved ones) according to the user judgement, respectively. In the original formulations, Rochio [678] fixed $\alpha = 1$ and Ide [391] fixed $\alpha = \beta = \gamma = 1$. The expressions above are modern variants. The current understanding is that the three techniques yield similar results (in the past, Ide Dec-Hi was considered slightly better).

The Rochio formulation is basically a direct adaptation of equation 5.1 in which the terms of the original query are added in. The motivation is that in practice the original query q may contain important information. Usually, the information contained in the relevant documents is more important than the information provided by the non-relevant documents [698]. This suggests making the constant γ smaller than the constant β. An alternative approach is to set γ to 0 which yields a *positive* feedback strategy.

The main advantages of the above relevance feedback techniques are simplicity and good results. The simplicity is due to the fact that the modified term weights are computed directly from the set of retrieved documents. The good

results are observed experimentally and are due to the fact that the modified query vector does reflect a portion of the intended query semantics. The main disadvantage is that *no* optimality criterion is adopted.

5.2.2 Term Reweighting for the Probabilistic Model

The probabilistic model dynamically ranks documents similar to a query q according to the probabilistic ranking principle. From Chapter 2, we already know that the similarity of a document d_j to a query q can be expressed as

$$sim(d_j, q) \; \alpha \; \sum_{i=1}^{t} w_{i,q} \; w_{i,j} \; \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \qquad \textbf{(5.2)}$$

where $P(k_i|R)$ stands for the probability of observing the term k_i in the set R of relevant documents and $P(k_i|\overline{R})$ stands for the probability of observing the term k_i in the set $\overline{R}$ of non-relevant documents.

Initially, equation 5.2 cannot be used because the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ are unknown. A number of different methods for estimating these probabilities automatically (i.e., without feedback from the user) were discussed in Chapter 2. With user feedback information, these probabilities are estimated in a slightly different way as follows.

For the initial search (when there are no retrieved documents yet), assumptions often made include: (a) $P(k_i|R)$ is constant for all terms k_i (typically 0.5) and (b) the term probability distribution $P(k_i|\overline{R})$ can be approximated by the distribution in the whole collection. These two assumptions yield:

$$\begin{aligned} P(k_i|R) &= 0.5 \\ P(k_i|\overline{R}) &= \frac{n_i}{N} \end{aligned}$$

where, as before, n_i stands for the number of documents in the collection which contain the term k_i. Substituting into equation 5.2, we obtain

$$sim_{initial}(d_j, q) = \sum_{i}^{t} w_{i,q} \; w_{i,j} \; \log \frac{N - n_i}{n_i}$$

For the feedback searches, the accumulated statistics related to the relevance or non-relevance of previously retrieved documents are used to evaluate the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$. As before, let D_r be the set of relevant retrieved documents (according to the user judgement) and $D_{r,i}$ be the subset of D_r composed of the documents which contain the term k_i. Then, the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ can be approximated by

$$P(k_i|R) = \frac{|D_{r,i}|}{|D_r|}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}|}{N - |D_r|} \qquad \textbf{(5.3)}$$

Using these approximations, equation 5.2 can rewritten as

$$sim(d_j, q) = \sum_{i=1}^{t} w_{i,q}\, w_{i,j}\, \log \left[\frac{|D_{r,i}|}{|D_r| - |D_{r,i}|} \div \frac{n_i - |D_{r,i}|}{N - |D_r| - (n_i - |D_{r,i}|)} \right]$$

Notice that here, contrary to the procedure in the vector space model, no query expansion occurs. The same query terms are being reweighted using feedback information provided by the user.

Formula 5.3 poses problems for certain small values of $|D_r|$ and $|D_{r,i}|$ that frequently arise in practice ($|D_r| = 1, |D_{r,i}| = 0$). For this reason, a 0.5 adjustment factor is often added to the estimation of $P(k_i|R)$ and $P(k_i|\overline{R})$ yielding

$$P(k_i|R) = \frac{|D_{r,i}| + 0.5}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + 0.5}{N - |D_r| + 1} \tag{5.4}$$

This 0.5 adjustment factor may provide unsatisfactory estimates in some cases, and alternative adjustments have been proposed such as n_i/N or $(n_i - |D_{r,i}|)/(N - |D_r|)$ [843]. Taking n_i/N as the adjustment factor (instead of 0.5), equation 5.4 becomes

$$P(k_i|R) = \frac{|D_{r,i}| + \frac{n_i}{N}}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + \frac{n_i}{N}}{N - |D_r| + 1}$$

The main advantages of this relevance feedback procedure are that the feedback process is directly related to the derivation of new weights for *query* terms and that the term reweighting is optimal under the assumptions of term independence and binary document indexing ($w_{i,q} \in \{0,1\}$ and $w_{i,j} \in \{0,1\}$). The disadvantages include: (1) document term weights are *not* taken into account during the feedback loop; (2) weights of terms in the previous query formulations are also disregarded; and (3) no query expansion is used (the same set of index terms in the original query is reweighted over and over again). As a result of these disadvantages, the probabilistic relevance feedback methods do *not* in general operate as effectively as the conventional vector modification methods.

To extend the probabilistic model with query expansion capabilities, different approaches have been proposed in the literature ranging from term weighting for query expansion to term clustering techniques based on spanning trees. All of these approaches treat probabilistic query expansion separately from probabilistic term reweighting. While we do not discuss them here, a brief history of research on this issue and bibliographical references can be found in section 5.6.

5.2.3 A Variant of Probabilistic Term Reweighting

The discussion above on term reweighting is based on the classic probabilistic model introduced by Robertson and Sparck Jones in 1976. In 1983, Croft extended this weighting scheme by suggesting distinct initial search methods

and by adapting the probabilistic formula to include within-document frequency weights. This variant of probabilistic term reweighting is more flexible (and also more powerful) and is briefly reviewed in this section.

The formula 5.2 for probabilistic ranking can be rewritten as

$$sim(d_j.q) \; \alpha \; \sum_{i=1}^{t} w_{i,q} \; w_{i,j} \; F_{i,j,q}$$

where $F_{i,j,q}$ is interpreted as a factor which depends on the triple $[k_i, d_j, q]$. In the classic formulation, $F_{i,j,q}$ is computed as a function of $P(k_i|R)$ and $P(k_i|\overline{R})$ (see equation 5.2). In his variant, Croft proposed that the initial search and the feedback searches use distinct formulations.

For the initial search, he suggested

$$\begin{aligned} F_{i,j,q} &= (C + idf_i) \; \overline{f}_{i,j} \\ \overline{f}_{i,j} &= K + (1 + K) \; \frac{f_{i,j}}{max(f_{i,j})} \end{aligned}$$

where $\overline{f}_{i,j}$ is a normalized within-document frequency. The parameters C and K should be adjusted according to the collection. For automatically indexed collections, C should be initially set to 0.

For the feedback searches, Croft suggested the following formulation for $F_{i,j,q}$

$$F_{i,j,q} = \left(C + \log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \; \overline{f}_{i,j}$$

where $P(k_i|R)$ and $P(k_i|\overline{R})$ are computed as in equation 5.4.

This variant of probabilistic term reweighting has the following advantages: (1) it takes into account the within-document frequencies; (2) it adopts a normalized version of these frequencies; and (3) it introduces the constants C and K which provide for greater flexibility. However, it constitutes a more complex formulation and, as before, it operates solely on the terms originally in the query (without query expansion).

5.2.4 Evaluation of Relevance Feedback Strategies

Consider the modified query vector $\vec{q}_m$ generated by the Rochio formula and assume that we want to evaluate its retrieval performance. A simplistic approach is to retrieve a set of documents using $\vec{q}_m$, to rank them using the vector formula, and to measure recall-precision figures relative to the set of relevant documents (provided by the experts) for the original query vector $\vec{q}$. In general, the results show spectacular improvements. Unfortunately, a significant part of this improvement results from the higher ranks assigned to the set R of documents

already identified as relevant during the feedback process [275]. Since the user has seen these documents already (and pointed them as relevants), such evaluation is unrealistic. Further, it masks any real gains in retrieval performance due to documents not seen by the user yet.

A more realistic approach is to evaluate the retrieval performance of the modified query vector $\vec{q}_m$ considering only the *residual collection* i.e., the set of all documents minus the set of feedback documents provided by the user. Because highly ranked documents are removed from the collection, the recall-precision figures for $\vec{q}_m$ tend to be lower than the figures for the original query vector $\vec{q}$. This is not a limitation because our main purpose is to compare the performance of distinct relevance feedback strategies (and not to compare the performance before and after feedback). Thus, as a basic rule of thumb, any experimentation involving relevance feedback strategies should always evaluate recall-precision figures relative to the residual collection.

5.3 Automatic Local Analysis

In a user relevance feedback cycle, the user examines the top ranked documents and separates them into two classes: the relevant ones and the non-relevant ones. This information is then used to select new terms for query expansion. The reasoning is that the expanded query will retrieve more relevant documents. Thus, there is an underlying notion of *clustering* supporting the feedback strategy. According to this notion, known relevant documents contain terms which can be used to describe a larger cluster of relevant documents. In this case, the description of this larger cluster of relevant documents is built interactively with assistance from the user.

A distinct approach is to attempt to obtain a description for a larger cluster of relevant documents automatically. This usually involves identifying terms which are related to the query terms. Such terms might be synonyms, stemming variations, or terms which are close to the query terms in the text (i.e., terms with a distance of at most k words from a query term). Two basic types of strategies can be attempted: global ones and local ones.

In a global strategy, all documents in the collection are used to determine a global thesaurus-like structure which defines term relationships. This structure is then shown to the user who selects terms for query expansion. Global strategies are discussed in section 5.4.

In a local strategy, the documents retrieved for a given query q are examined at query time to determine terms for query expansion. This is similar to a relevance feedback cycle but might be done without assistance from the user (i.e., the approach might be fully automatic). Two local strategies are discussed below: local clustering and local context analysis. The first is based on the work done by Attar and Fraenkel in 1977 and is used here to establish many of the fundamental ideas and concepts regarding the usage of clustering for query expansion. The second is a recent work done by Xu and Croft in 1996 and illustrates the advantages of combining techniques from both local and global analysis.

5.3.1 Query Expansion Through Local Clustering

Adoption of clustering techniques for query expansion is a basic approach which has been attempted since the early years of information retrieval. The standard approach is to build global structures such as association matrices which quantify term correlations (for instance, number of documents in which two given terms co-occur) and to use correlated terms for query expansion. The main problem with this strategy is that there is not consistent evidence that global structures can be used effectively to improve retrieval performance with general collections. One main reason seems to be that global structures do not adapt well to the local context defined by the current query. One approach to deal with this effect is to devise strategies which aim at optimizing the current search. Such strategies are based on *local clustering* and are now discussed. Our discussion is based on the original work by Attar and Fraenkel which appeared in 1977.

We first define basic terminology as follows.

Definition *Let $V(s)$ be a non-empty subset of words which are grammatical variants of each other. A canonical form s of $V(s)$ is called a* stem. *For instance, if $V(s)$=\{polish,polishing,polished\} then s=polish.*

For a detailed discussion on stemming algorithms see Chapter 7. While stems are adopted in our discussion, the ideas below are also valid for non-stemmed keywords. We proceed with a characterization of the local nature of the strategies covered here.

Definition *For a given query q, the set D_l of documents retrieved is called the local document set. Further, the set V_l of all distinct words in the local document set is called the local vocabulary. The set of all distinct stems derived from the set V_l is referred to as S_l.*

We operate solely on the documents retrieved for the current query. Since it is frequently necessary to access the text of such documents, the application of local strategies to the Web is unlikely at this time. In fact, at a client machine, retrieving the text of 100 Web documents for local analysis would take too long, reducing drastically the interactive nature of Web interfaces and the satisfaction of the users. Further, at the search engine site, analyzing the text of 100 Web documents would represent an extra spending of CPU time which is not cost effective at this time (because search engines depend on processing a high number of queries per unit of time for economic survival). However, local strategies might be quite useful in the environment of intranets such as, for instance, the collection of documents issued by a large business company. Further, local strategies might also be of great assistance for searching information in specialized document collections (for instance, medical document collections).

Local feedback strategies are based on expanding the query with terms correlated to the query terms. Such correlated terms are those present in local clusters built from the local document set. Thus, before we discuss local

query expansion, we discuss strategies for building local clusters. Three types of clusters are covered: association clusters, metric clusters, and scalar clusters.

Association Clusters

An association cluster is based on the co-occurrence of stems (or terms) inside documents. The idea is that stems which co-occur frequently inside documents have a synonymity association. Association clusters are generated as follows.

Definition *The frequency of a stem s_i in a document d_j, $d_j \in D_l$, is referred to as $f_{s_i,j}$. Let $\vec{m}=(m_{ij})$ be an association matrix with $|S_l|$ rows and $|D_l|$ columns, where $m_{ij}=f_{s_i,j}$. Let $\vec{m}^t$ be the transpose of $\vec{m}$. The matrix $\vec{s}=\vec{m}\vec{m}^t$ is a local stem-stem association matrix. Each element $s_{u,v}$ in $\vec{s}$ expresses a correlation $c_{u,v}$ between the stems s_u and s_v namely,*

$$c_{u,v} = \sum_{d_j \in D_l} f_{s_u,j} \times f_{s_v,j} \tag{5.5}$$

The correlation factor $c_{u,v}$ quantifies the absolute frequencies of co-occurrence and is said to be unnormalized. Thus, if we adopt

$$s_{u,v} = c_{u,v} \tag{5.6}$$

then the association matrix $\vec{s}$ is said to be unnormalized. An alternative is to normalize the correlation factor. For instance, if we adopt

$$s_{u,v} = \frac{c_{u,v}}{c_{u,u} + c_{v,v} - c_{u,v}} \tag{5.7}$$

then the association matrix $\vec{s}$ is said to be normalized. The adoption of normalization yields quite distinct associations as discussed below.

Given a local association matrix $\vec{s}$, we can use it to build local association clusters as follows.

Definition *Consider the u-th row in the association matrix $\vec{s}$ (i.e., the row with all the associations for the stem s_u). Let $S_u(n)$ be a function which takes the u-th row and returns the set of n largest values $s_{u,v}$, where v varies over the set of local stems and $v \neq u$. Then $S_u(n)$ defines a local association cluster around the stem s_u. If $s_{u,v}$ is given by equation 5.6, the association cluster is said to be unnormalized. If $s_{u,v}$ is given by equation 5.7, the association cluster is said to be normalized.*

Given a query q, we are normally interested in finding clusters only for the $|q|$ query terms. Further, it is desirable to keep the size of such clusters small. This means that such clusters can be computed efficiently at query time.

Despite the fact that the above clustering procedure adopts stems, it can equally be applied to non-stemmed keywords. The procedure remains unchanged except for the usage of keywords instead of stems. Keyword-based local clustering is equally worthwhile trying because there is controversy over the advantages of using a stemmed vocabulary, as discussed in Chapter 7.

Metric Clusters

Association clusters are based on the frequency of co-occurrence of pairs of terms in documents and do not take into account *where* the terms occur in a document. Since two terms which occur in the same sentence seem more correlated than two terms which occur far apart in a document, it might be worthwhile to factor in the distance between two terms in the computation of their correlation factor. Metric clusters are based on this idea.

Definition *Let the distance $r(k_i, k_j)$ between two keywords k_i and k_j be given by the number of words between them in a same document. If k_i and k_j are in distinct documents we take $r(k_i, k_j) = \infty$. A local stem-stem metric correlation matrix $\vec{s}$ is defined as follows. Each element $s_{u,v}$ of $\vec{s}$ expresses a metric correlation $c_{u,v}$ between the stems s_u and s_v namely,*

$$c_{u,v} = \sum_{k_i \in V(s_u)} \sum_{k_j \in V(s_v)} \frac{1}{r(k_i, k_j)}$$

In this expression, as already defined, $V(s_u)$ and $V(s_v)$ indicate the sets of keywords which have s_u and s_v as their respective stems. Variations of the above expression for $c_{u,v}$ have been reported in the literature (such as $1/r^2(k_i, k_j)$) but the differences in experimental results are not remarkable.

The correlation factor $c_{u,v}$ quantifies absolute inverse distances and is said to be unnormalized. Thus, if we adopt

$$s_{u,v} = c_{u,v} \tag{5.8}$$

then the association matrix $\vec{s}$ is said to be unnormalized. An alternative is to normalize the correlation factor. For instance, if we adopt

$$s_{u,v} = \frac{c_{u,v}}{|V(s_u)| \times |V(s_v)|} \tag{5.9}$$

then the association matrix $\vec{s}$ is said to be normalized.

Given a local metric matrix $\vec{s}$, we can use it to build local metric clusters as follows.

Definition *Consider the u-th row in the metric correlation matrix $\vec{s}$ (i.e., the row with all the associations for the stem s_u). Let $S_u(n)$ be a function which*

takes the u-th row and returns the set of n largest values $s_{u,v}$, where v varies over the set of local stems and $v \neq u$. Then $S_u(n)$ defines a local metric cluster around the stem s_u. If $s_{u,v}$ is given by equation 5.8, the metric cluster is said to be unnormalized. If $s_{u,v}$ is given by equation 5.9, the metric cluster is said to be normalized.

Scalar Clusters

One additional form of deriving a synonymity relationship between two local stems (or terms) s_u and s_v is by comparing the sets $S_u(n)$ and $S_v(n)$. The idea is that two stems with similar *neighborhoods* have some synonymity relationship. In this case we say that the relationship is indirect or induced by the neighborhood. One way of quantifying such neighborhood relationships is to arrange all correlation values $s_{u,i}$ in a vector $\vec{s}_u$, to arrange all correlation values $s_{v,i}$ in another vector $\vec{s}_v$, and to compare these vectors through a scalar measure. For instance, the cosine of the angle between the two vectors is a popular scalar similarity measure.

Definition *Let $\vec{s}_u = (s_{u,1}, s_{u,2}, \ldots, s_{u,n})$ and $\vec{s}_v = (s_{v,1}, s_{v,2}, \ldots, s_{v,n})$ be two vectors of correlation values for the stems s_u and s_v. Further, let $\vec{s} = (s_{u,v})$ be a scalar association matrix. Then, each $s_{u,v}$ can be defined as*

$$s_{u,v} = \frac{\vec{s}_u \cdot \vec{s}_v}{|\vec{s}_u| \times |\vec{s}_v|} \tag{5.10}$$

The correlation matrix $\vec{s}$ is said to be induced by the neighborhood. Using it, a scalar cluster is then defined as follows.

Definition *Let $S_u(n)$ be a function which returns the set of n largest values $s_{u,v}$, $v \neq u$, defined according to equation 5.10. Then, $S_u(n)$ defines a scalar cluster around the stem s_u.*

Interactive Search Formulation

Stems (or terms) that belong to clusters associated to the query stems (or terms) can be used to expand the original query. Such stems are called neighbors (of the query stems) and are characterized as follows.

A stem s_u which belongs to a cluster (of size n) associated to another stem s_v (i.e., $s_u \in S_v(n)$) is said to be a *neighbor* of s_v. Sometimes, s_u is also called a *searchonym* of s_v but here we opt for using the terminology *neighbor*. While neighbor stems are said to have a synonymity relationship, they are not necessarily synonyms in the grammatical sense. Often, neighbor stems represent distinct keywords which are though correlated by the current query context. The local aspect of this correlation is reflected in the fact that the documents and stems considered in the correlation matrix are all local (i.e., $d_j \in D_l$, $s_u \in V_l$).

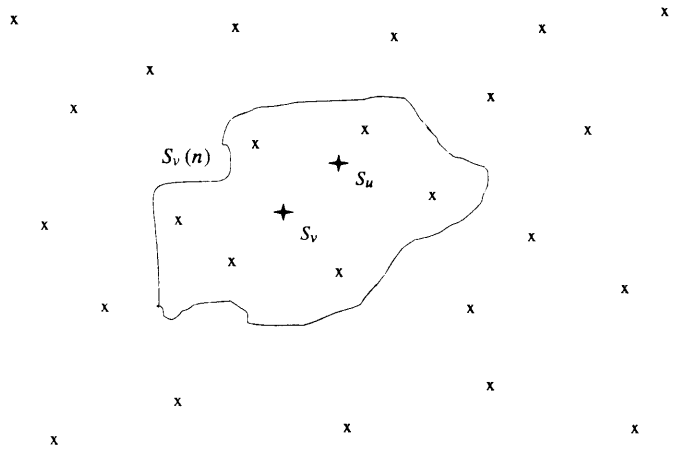

Figure 5.1 Stem s_u as a neighbor of the stem s_v.

Figure 5.1 illustrates a stem (or term) s_u which is located within a neighborhood $S_v(n)$ associated with the stem (or term) s_v. In its broad meaning, neighbor stems are an important product of the local clustering process since they can be used for extending a search formulation in a promising unexpected direction, rather than merely complementing it with missing synonyms.

Consider the problem of expanding a given user query q with neighbor stems (or terms). One possibility is to expand the query as follows. For each stem $s_v \in q$, select m neighbor stems from the cluster $S_v(n)$ (which might be of type association, metric, or scalar) and add them to the query. Hopefully, the additional neighbor stems will retrieve new relevant documents. To cover a broader neighborhood, the set $S_v(n)$ might be composed of stems obtained using correlation factors (i.e., $c_{u,v}$) normalized and unnormalized. The qualitative interpretation is that an unnormalized cluster tends to group stems whose ties are due to their large frequencies, while a normalized cluster tends to group stems which are more rare. Thus, the union of the two clusters provides a better representation of the possible correlations.

Besides the merging of normalized and unnormalized clusters, one can also use information about correlated stems to improve the search. For instance, as before, let two stems s_u and s_v be correlated with a correlation factor $c_{u,v}$. If $c_{u,v}$ is larger than a predefined threshold then a neighbor stem of s_u can also be interpreted as a neighbor stem of s_v and vice versa. This provides greater flexibility, particularly with Boolean queries. To illustrate, consider the expression $(s_u + s_v)$ where the + symbol stands for disjunction. Let $s_{u'}$ be a neighbor stem of s_u. Then, one can try both $(s_{u'} + s_v)$ and $(s_u + s_{u'})$ as synonym search expressions, because of the correlation given by $c_{u,v}$.

Experimental results reported in the literature usually support the hypothesis of the usefulness of local clustering methods. Furthermore, metric clusters seem to perform better than purely association clusters. This strengthens the hypothesis that there is a correlation between the association of two terms and the distance between them.

We emphasize that all the qualitative arguments in this section are explicitly based on the fact that all the clusters are local (i.e., derived solely from the documents retrieved for the current query). In a global context, clusters are derived from all the documents in the collection which implies that our qualitative argumentation might not stand. The main reason is that correlations valid in the whole corpora might not be valid for the current query.

5.3.2 Query Expansion Through Local Context Analysis

The local clustering techniques discussed above are based on the set of documents retrieved for the original query and use the top ranked documents for clustering neighbor terms (or stems). Such a clustering is based on term (stems were considered above) co-occurrence inside documents. Terms which are the best neighbors of each query term are then used to expand the original query q. A distinct approach is to search for term correlations in the whole collection — an approach called global analysis. Global techniques usually involve the building of a thesaurus which identifies term relationships in the whole collection. The terms are treated as concepts and the thesaurus is viewed as a concept relationship structure. Thesauri are expensive to build but, besides providing support for query expansion, are useful as a browsing tool as demonstrated by some search engines in the Web. The building of a thesaurus usually considers the use of small contexts and phrase structures instead of simply adopting the context provided by a whole document. Furthermore, with modern variants of global analysis, terms which are closest to the whole query (and not to individual query terms) are selected for query expansion. The application of ideas from global analysis (such as small contexts and phrase structures) to the local set of documents retrieved is a recent idea which we now discuss.

Local context analysis [838] combines global and local analysis and works as follows. First, the approach is based on the use of noun groups (i.e., a single noun, two adjacent nouns, or three adjacent nouns in the text), instead of simple keywords, as document concepts. For query expansion, concepts are selected from the top ranked documents (as in local analysis) based on their co-occurrence with query terms (no stemming). However, instead of documents, passages (i.e., a text window of fixed size) are used for determining co-occurrence (as in global analysis).

More specifically, the local context analysis procedure operates in three steps.

- First, retrieve the top n ranked passages using the original query. This is accomplished by breaking up the documents initially retrieved by the

query in fixed length passages (for instance, of size 300 words) and ranking these passages as if they were documents.

- Second, for each concept c in the top ranked passages, the similarity $sim(q, c)$ between the whole query q (not individual query terms) and the concept c is computed using a variant of tf-idf ranking.
- Third, the top m ranked concepts (according to $sim(q, c)$) are added to the original query q. To each added concept is assigned a weight given by $1 - 0.9 \times i/m$ where i is the position of the concept in the final concept ranking. The terms in the original query q might be stressed by assigning a weight equal to 2 to each of them.

Of these three steps, the second one is the most complex and the one which we now discuss.

The similarity $sim(q, c)$ between each related concept c and the original query q is computed as follows.

$$sim(q, c) = \prod_{k_i \in q} \left(\delta + \frac{\log(f(c, k_i) \times idf_c)}{\log n} \right)^{idf_i}$$

where n is the number of top ranked passages considered. The function $f(c, k_i)$ quantifies the correlation between the concept c and the query term k_i and is given by

$$f(c, k_i) = \sum_{j=1}^{n} pf_{i,j} \times pf_{c,j}$$

where $pf_{i,j}$ is the frequency of term k_i in the j-th passage and $pf_{c,j}$ is the frequency of the concept c in the j-th passage. Notice that this is the standard correlation measure defined for association clusters (by Equation 5.5) but adapted for passages. The inverse document frequency factors are computed as

$$idf_i = max(1, \frac{\log_{10} N/np_i}{5})$$
$$idf_c = max(1, \frac{\log_{10} N/np_c}{5})$$

where N is the number of passages in the collection, np_i is the number of passages containing the term k_i, and np_c is the number of passages containing the concept c. The factor δ is a constant parameter which avoids a value equal to zero for $sim(q, c)$ (which is useful, for instance, if the approach is to be used with probabilistic frameworks such as that provided by belief networks). Usually, δ is a small factor with values close to 0.1 (10% of the maximum of 1). Finally, the idf_i factor in the exponent is introduced to emphasize infrequent query terms.

The procedure above for computing $sim(q, c)$ is a non-trivial variant of tf-idf ranking. Furthermore, it has been adjusted for operation with TREC data

and did not work so well with a different collection. Thus, it is important to have in mind that tuning might be required for operation with a different collection.

We also notice that the correlation measure adopted with local context analysis is of type association. However, we already know that a correlation of type metric is expected to be more effective. Thus, it remains to be tested whether the adoption of a metric correlation factor (for the function $f(c, k_i)$) makes any difference with local context analysis.

5.4 Automatic Global Analysis

The methods of local analysis discussed above extract information from the local set of documents retrieved to expand the query. It is well accepted that such a procedure yields improved retrieval performance with various collections. An alternative approach is to expand the query using information from the whole set of documents in the collection. Strategies based on this idea are called global analysis procedures. Until the beginning of the 1990s, global analysis was considered to be a technique which failed to yield consistent improvements in retrieval performance with general collections. This perception has changed with the appearance of modern procedures for global analysis. In the following, we discuss two of these modern variants. Both of them are based on a thesaurus-like structure built using all the documents in the collection. However, the approach taken for building the thesaurus and the procedure for selecting terms for query expansion are quite distinct in the two cases.

5.4.1 Query Expansion based on a Similarity Thesaurus

In this section we discuss a query expansion model based on a global similarity thesaurus which is constructed automatically [655]. The similarity thesaurus is based on term to term relationships rather than on a matrix of co-occurrence (as discussed in section 5.3). The distinction is made clear in the discussion below. Furthermore, special attention is paid to the selection of terms for expansion and to the reweighting of these terms. In contrast to previous global analysis approaches, terms for expansion are selected based on their similarity to the whole query rather than on their similarities to individual query terms.

A *similarity thesaurus* is built considering term to term relationships. However, such relationships are not derived directly from co-occurrence of terms inside documents. Rather, they are obtained by considering that the terms are concepts in a concept space. In this concept space, each term is indexed by the documents in which it appears. Thus, terms assume the original role of documents while documents are interpreted as indexing elements. The following definitions establish the proper framework.

Definition *As before (see Chapter 2), let t be the number of terms in the collection, N be the number of documents in the collection, and $f_{i,j}$ be the frequency*

of occurrence of the term k_i in the document d_j. Further, let t_j be the number of distinct index terms in the document d_j and itf_j be the inverse term frequency for document d_j. Then,

$$itf_j = \log \frac{t}{t_j}$$

analogously to the definition of inverse document frequency.

Within this framework, to each term k_i is associated a vector $\vec{k}_i$ given by

$$\vec{k}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,N})$$

where, as in Chapter 2, $w_{i,j}$ is a weight associated to the index-document pair $[k_i, d_j]$. Here, however, these weights are computed in a rather distinct form as follows.

$$w_{i,j} = \frac{(0.5 + 0.5\frac{f_{i,j}}{max_j(f_{i,j})})\ itf_j}{\sqrt{\sum_{l=1}^{N}(0.5 + 0.5\frac{f_{i,l}}{max_l(f_{i,l})})^2\ itf_j^2}} \tag{5.11}$$

where $max_j(f_{i,j})$ computes the maximum of all factors $f_{i,j}$ for the i-th term (i.e., over all documents in the collection). We notice that the expression above is a variant of tf-idf weights but one which considers inverse term frequencies instead.

The relationship between two terms k_u and k_v is computed as a correlation factor $c_{u,v}$ given by

$$c_{u,v} = \vec{k}_u \cdot \vec{k}_v = \sum_{\forall\ d_j} w_{u,j} \times w_{v,j} \tag{5.12}$$

We notice that this is a variation of the correlation measure used for computing scalar association matrices (defined by Equation 5.5). The main difference is that the weights are based on interpreting documents as indexing elements instead of repositories for term co-occurrence. The global similarity thesaurus is built through the computation of the correlation factor $c_{u,v}$ for each pair of indexing terms $[k_u, k_v]$ in the collection (analogously to the procedure in section 5.3). Of course, this is computationally expensive. However, this global similarity thesaurus has to be computed only once and can be updated incrementally.

Given the global similarity thesaurus, query expansion is done in three steps as follows.

- First, represent the query in the concept space used for representation of the index terms.

- Second, based on the global similarity thesaurus, compute a similarity $sim(q, k_v)$ between each term k_v correlated to the query terms and the whole query q.
- Third, expand the query with the top r ranked terms according to $sim(q, k_v)$.

For the first step, the query is represented in the concept space of index term vectors as follows.

Definition *To the query q is associated a vector $\vec{q}$ in the term-concept space given by*

$$\vec{q} = \sum_{k_i \in q} w_{i,q} \vec{k}_i$$

where $w_{i,q}$ is a weight associated to the index-query pair $[k_i, q]$. This weight is computed analogously to the index-document weight formula in equation 5.11.

For the second step, a similarity $sim(q, k_v)$ between each term k_v (correlated to the query terms) and the user query q is computed as

$$sim(q, k_v) = \vec{q} \cdot \vec{k}_v = \sum_{k_u \in Q} w_{u,q} \times c_{u,v}$$

where $c_{u,v}$ is the correlation factor given in equation 5.12. As illustrated in Figure 5.2, a term might be quite close to the whole query while its distances to individual query terms are larger. This implies that the terms selected here for query expansion might be distinct from those selected by previous global analysis methods (which adopted a similarity to individual query terms for deciding terms for query expansion).

For the third step, the top r ranked terms according to $sim(q, k_v)$ are added to the original query q to form the expanded query q'. To each expansion term k_v in the query q' is assigned a weight $w_{v,q'}$ given by

$$w_{v,q'} = \frac{sim(q, k_v)}{\sum_{k_u \in q} w_{u,q}}$$

The expanded query q' is then used to retrieve new documents to the user. This completes the technique for query expansion based on a similarity thesaurus. Contrary to previous global analysis approaches, this technique has yielded improved retrieval performance (in the range of 20%) with three different collections.

It is worthwhile making one final observation. Consider a document d_j which is represented in the term-concept space by $\vec{d}_j = \sum_{k_i \in d_j} w_{i,j} \vec{k}_i$. Further, assume that the original query q is expanded to include all the t index terms

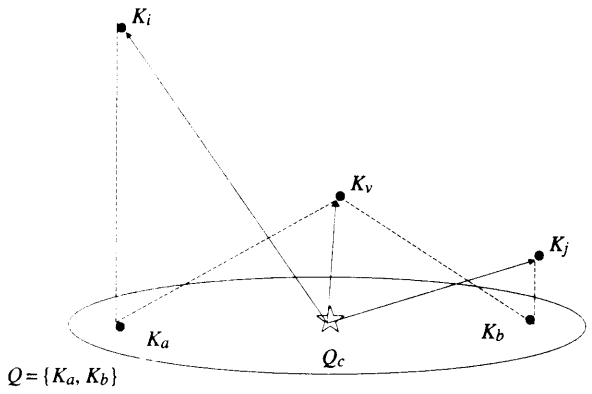

Figure 5.2 The distance of a given term k_v to the query centroid Q_c might be quite distinct from the distances of k_v to the individual query terms.

(properly weighted) in the collection. Then, the similarity $sim(q, d_j)$ between the document d_j and the query q can be computed in the term-concept space by

$$sim(q, d_j) \propto \sum_{k_v \in d_j} \sum_{k_u \in q} w_{v,j} \times w_{u,q} \times c_{u,v}$$

Such an expression is analogous to the formula for query-document similarity in the generalized vector space model (see Chapter 2). Thus, the generalized vector space model can be interpreted as a query expansion technique. The main differences with the term-concept idea are the weight computation and the fact that only the top r ranked terms are used for query expansion with the term-concept technique.

5.4.2 Query Expansion based on a Statistical Thesaurus

In this section, we discuss a query expansion technique based on a global statistical thesaurus [200]. Despite also being a global analysis technique, the approach is quite distinct from the one described above which is based on a similarity thesaurus.

The global thesaurus is composed of classes which group correlated terms in the context of the whole collection. Such correlated terms can then be used to expand the original user query. To be effective, the terms selected for expansion must have high term discrimination values [699] which implies that they must be low frequency terms. However, it is difficult to cluster low frequency terms effectively due to the small amount of information about them (they occur in few documents). To circumvent this problem, we cluster documents into

classes instead and use the low frequency terms in these documents to define our thesaurus classes. In this situation, the document clustering algorithm must produce small and tight clusters.

A document clustering algorithm which produces small and tight clusters is the *complete link algorithm* which works as follows (naive formulation).

(1) Initially, place each document in a distinct cluster.
(2) Compute the similarity between all pairs of clusters.
(3) Determine the pair of clusters $[C_u, C_v]$ with the highest inter-cluster similarity.
(4) Merge the clusters C_u and C_v.
(5) Verify a stop criterion. If this criterion is not met then go back to step 2.
(6) Return a hierarchy of clusters.

The similarity between two clusters is defined as the minimum of the similarities between all pairs of inter-cluster documents (i.e., two documents not in the same cluster). To compute the similarity between documents in a pair, the cosine formula of the vector model is used. As a result of this minimality criterion, the resultant clusters tend to be small and tight.

Consider that the whole document collection has been clustered using the complete link algorithm. Figure 5.3 illustrates a small portion of the whole cluster hierarchy in which $sim(C_u, C_v) = 0.15$ and $sim(C_{u+v}, C_z) = 0.11$ where C_{u+v} is a reference to the cluster which results from merging C_u and C_v. Notice that the similarities decrease as we move up in the hierarchy because high level clusters include more documents and thus represent a looser grouping. Thus, the tightest clusters lie at the bottom of the clustering hierarchy.

Given the document cluster hierarchy for the whole collection, the terms that compose each class of the global thesaurus are selected as follows.

- Obtain from the user three parameters: threshold class (TC), number of

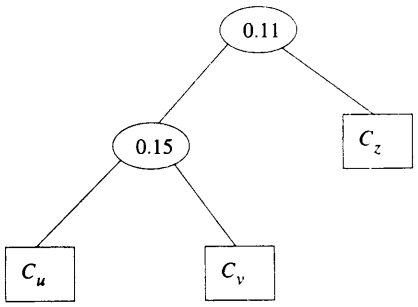

Figure 5.3 Hierarchy of three clusters (inter-cluster similarities indicated in the ovals) generated by the complete link algorithm.

documents in a class (NDC), and minimum inverse document frequency (MIDF).

- Use the parameter TC as a threshold value for determining the document clusters that will be used to generate thesaurus classes. This threshold has to be surpassed by $sim(C_u, C_v)$ if the documents in the clusters C_u and C_v are to be selected as sources of terms for a thesaurus class. For instance, in Figure 5.3, a value of 0.14 for TC returns the thesaurus class C_{u+v} while a value of 0.10 for TC returns the classes C_{u+v} and C_{u+v+z}.
- Use the parameter NDC as a limit on the size of clusters (number of documents) to be considered. For instance, if both C_{u+v} and C_{u+v+z} are preselected (through the parameter TC) then the parameter NDC might be used to decide between the two. A low value of NDC might restrict the selection to the smaller cluster C_{u+v}.
- Consider the set of documents in each document cluster preselected above (through the parameters TC and NDC). Only the lower frequency documents are used as sources of terms for the thesaurus classes. The parameter MIDF defines the minimum value of inverse document frequency for any term which is selected to participate in a thesaurus class. By doing so, it is possible to ensure that only *low frequency* terms participate in the thesaurus classes generated (terms too generic are not good synonyms).

Given that the thesaurus classes have been built, they can be used for query expansion. For this, an average term weight wt_C for each thesaurus class C is computed as follows.

$$wt_C = \frac{\sum_{i=1}^{|C|} w_{i,C}}{|C|}$$

where $|C|$ is the number of terms in the thesaurus class C and $w_{i,C}$ is a precomputed weight associated with the term-class pair $[k_i, C]$. This average term weight can then be used to compute a thesaurus class weight w_C as

$$w_C = \frac{wt_C}{|C|} \times 0.5$$

The above weight formulations have been verified through experimentation and have yielded good results.

Experiments with four test collections (ADI, Medlars, CACM, and ISI; see Chapter 3 for details on these collections) indicate that global analysis using a thesaurus built by the complete link algorithm might yield consistent improvements in retrieval performance.

The main problem with this approach is the initialization of the parameters TC, NDC, and MIDF. The threshold value TC depends on the collection and can be difficult to set properly. Inspection of the cluster hierarchy is almost always necessary for assisting with the setting of TC. Care must be exercised because a

high value of TC might yield classes with too few terms while a low value of TC might yield too few classes. The selection of the parameter NDC can be decided more easily once TC has been set. However, the setting of the parameter MIDF might be difficult and also requires careful consideration.

5.5 Trends and Research Issues

The relevance feedback strategies discussed here can be directly applied to the graphical interfaces of modern information systems. However, since interactivity is now of greater importance, new techniques for capturing feedback information from the user are desirable. For instance, there is great interest in graphical interfaces which display the documents in the answer set as points in a 2D or 3D space. The motivation is to allow the user to quickly identify (by visual inspection) relationships among the documents in the answer. In this scenario, a rather distinct strategy for quantifying feedback information might be required. Thus, relevance strategies for dealing with visual displays are an important research problem.

In the past, global analysis was viewed as an approach which did not yield good improvements in retrieval performance. However, new results obtained at the beginning of the 1990s changed this perception. Further, the Web has provided evidence that techniques based on global analysis might be of interest to the users. For instance, this is the case with the highly popular 'Yahoo!' software which uses a manually built hierarchy of concepts to assist the user with forming the query. This suggests that investigating the utilization of global analysis techniques in the Web is a promising research problem.

Local analysis techniques are interesting because they take advantage of the local context provided with the query. In this regard, they seem more appropriate than global analysis techniques. Furthermore, many positive results have been reported in the literature. The application of local analysis techniques to the Web, however, has not been explored and is a promising research direction. The main challenge is the computational burden imposed on the search engine site due to the need to process document texts at query time. Thus, a related research problem of relevance is the development of techniques for speeding up query processing at the search engine site. In truth, this problem is of interest even if one considers only the normal processing of the queries because the search engines depend on processing as many queries as possible for economic survival.

The combination of local analysis, global analysis, visual displays, and interactive interfaces is also a current and important research problem. Allowing the user to visually explore the document space and providing him with clues which assist with the query formulation process are highly relevant issues. Positive results in this area might become a turning point regarding the design of user interfaces and are likely to attract wide attention.

5.6 Bibliographic Discussion

Query expansion methods have been studied for a long time. While the success of expansion methods throughout the years has been debatable, more recently researchers have reached the consensus that query expansion is a useful and little explored (commercially) technique. Useful because its modern variants can be used to consistently improve the retrieval performance with general collections. Little explored because few commercial systems (and Web search engines) take advantage of it.

Early work suggesting the expansion of a user query with closely related terms was done by Maron and Kuhns in 1960 [547]. The classic technique for combining query expansion with term reweighting in the vector model was studied by Rocchio in 1965 (using the Smart system [695] as a testbed) and published later on [678]. Ide continued the studies of Rocchio and proposed variations to the term reweighting formula [391].

The probabilistic model was introduced by Robertson and Sparck Jones [677] in 1976. A thorough and entertaining discussion of this model can be found in the book by van Rijsbergen [785]. Croft and Harper [199] suggested that the initial search should use a distinct computation. In 1983, Croft [198] proposed to extend the probabilistic formula to include within-document frequencies and introduced the C and K parameters.

Since the probabilistic model does not provide means of expanding the query, query expansion has to be done separately. In 1978, Harper and van Rijsbergen [345] used a term-term clustering technique based on a maximum spanning tree to select terms for probabilistic query expansion. Two years later, they also introduced a new relevance weighting scheme, called EMIM [344], to be used with their query expansion technique. In 1981, Wu and Salton [835] used relevance feedback to reweight terms (using a probabilistic formula) extracted from relevant documents and used these terms to expand the query. Empirical results showed improvements in retrieval performance.

Our discussion on user relevance feedback for the vector and probabilistic models in section 5.2 is based on four sources: a nice paper by Salton and Buckley [696], the book by van Rijsbergen [785], the book by Salton and McGill [698], and two book chapters by Harman [340, 339].

Regarding automatic query expansion, Lesk [500] tried variations of term-term clustering in the Smart system without positive results. Following that, Sparck Jones and Barber [413] and Minker, Wilson and Zimmerman [562] also observed no improvements with query expansion based on term-term global clustering. These early research results left the impression that query expansion based on global analysis is not an effective technique. However, more recent results show that this is not the case. In fact, the research results obtained by Vorhees [793], by Crouch and Yang [200], and by Qiu and Frei [655] indicate that query expansion based on global analysis can consistently yield improved retrieval performance.

Our discussion on query expansion through local clustering is based on early work by Attar and Fraenkel [35] from 1977. The idea of local context

analysis is much more recent and was introduced by Xu and Croft [838] in 1996. The discussion on query expansion using a global similarity thesaurus is based on the work by Qiu and Frei [655]. Finally, the discussion on query expansion using a global statistical thesaurus is based on the work of Crouch and Yang [200] which is influenced by the term discrimination value theory introduced by Salton, Yang, and Yu [699] early in 1975.

Since query expansion frequently is based on some form of clustering, our discussion covered a few clustering algorithms. However, our aim was not to provide a thorough review of clustering algorithms for information retrieval. Such a review can be found in the work of Rasmussen [668].

Chapter 6
Text and Multimedia Languages and Properties

6.1 Introduction

Text is the main form of communicating knowledge. Starting with hieroglyphs, the first written surfaces (stone, wood, animal skin, papyrus, and rice paper), and paper, text has been created everywhere, in many forms and languages. We use the term *document* to denote a single unit of information, typically text in a digital form, but it can also include other media. In practice, a document is loosely defined. It can be a complete logical unit, like a research article, a book or a manual. It can also be part of a larger text, such as a paragraph or a sequence of paragraphs (also called a *passage* of text), an entry in a dictionary, a judge's opinion on a case, the description of an automobile part, etc. Furthermore, with respect to its physical representation, a document can be any physical unit, for example a file, an email, or a World Wide Web (or just Web) page.

A document has a given syntax and structure which is usually dictated by the application or by the person who created it. It also has a semantics, specified by the author of the document (who is not necessarily the same as the creator). Additionally, a document may have a presentation style associated with it, which specifies how it should be displayed or printed. Such a style is usually given by the document syntax and structure and is related to a specific application (for example, a Web browser). Figure 6.1 depicts all these relations. A document can also have information about itself, called *metadata*. The next section explains different types of metadata and their relevance.

The *syntax* of a document can express structure, presentation style, semantics, or even external actions. In many cases one or more of these elements are implicit or are given together. For example, a structural element (e.g., a section) can have a fixed formatting style. The semantics of a document is also associated with its use. For example, Postscript directives are designed for drawing.

The syntax of a document can be implicit in its content, or expressed in a simple declarative language or even in a programming language. For example, many editor formats are declarative while a TeX document uses a powerful typesetting language. Although a powerful language could be easier to parse than the data itself, it might be difficult to convert documents in that language to other formats. Many syntax languages are proprietary and specific, but open

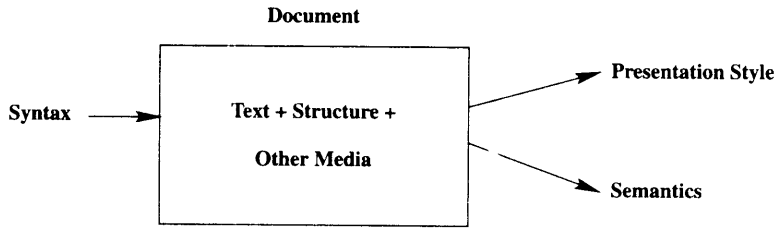

Figure 6.1 Characteristics of a document.

and generic languages are better because documents can be interchanged between applications and are more flexible. Text can also be written in natural language. However, at present the semantics of natural language is still not easy for a computer to understand. The current trend is to use languages which provide information on the document structure, format, and semantics while being readable by humans as well as computers. The Standard Generalized Markup Language (SGML), which is covered later on in this chapter, tries to balance all the issues above. Metadata, markup, and semantic encoding represent different levels of formalization of the document contents.

Most documents have a particular formatting style. However, new applications are pushing for external formatting such that information can be represented independently of style, and vice versa. The presentation style can be embedded in the document, as in TeX or Rich Text Format (RTF). Style can be complemented by macros (for example, LaTeX in the case of TeX). In most cases, style is defined by the document author. However, the reader may decide part of the style (for example, by setting options in a Web browser). The style of a document defines how the document is visualized in a computer window or a printed page, but can also include treatment of other media such as audio or video.

In this chapter we first cover metadata. Following that we discuss text characteristics such as formats and natural language statistics. Next we cover languages to describe text structure, presentation style, or semantics. The last part is devoted to multimedia formats and languages.

6.2 Metadata

Most documents and text collections have associated with them what is known as metadata. Metadata is information on the organization of the data, the various data domains, and the relationship between them. In short, metadata is 'data about the data.' For instance, in a database management system, the schema