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

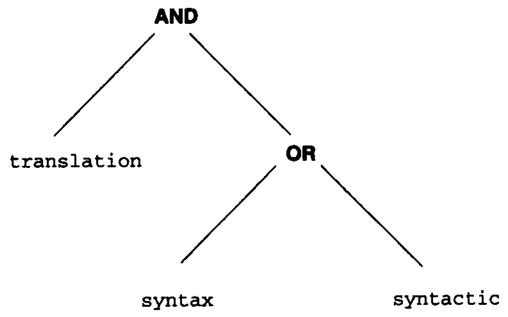

Figure 4.1 An example of a query syntax tree. It will retrieve all the documents which contain the word '`translation`' as well as either the word '`syntax`' or the word '`syntactic`'.

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

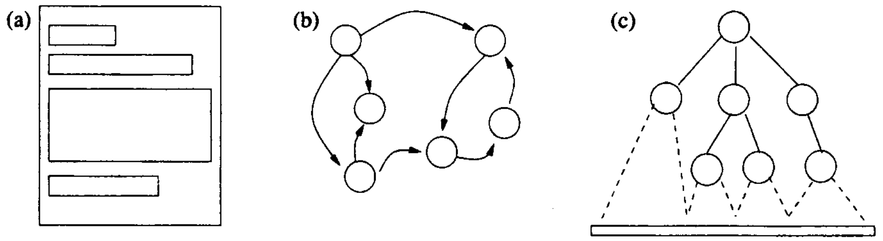

Figure 4.2 The three main structures: (*a*) form-like fixed structure, (*b*) hypertext structure, and (*c*) hierarchical structure.

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.

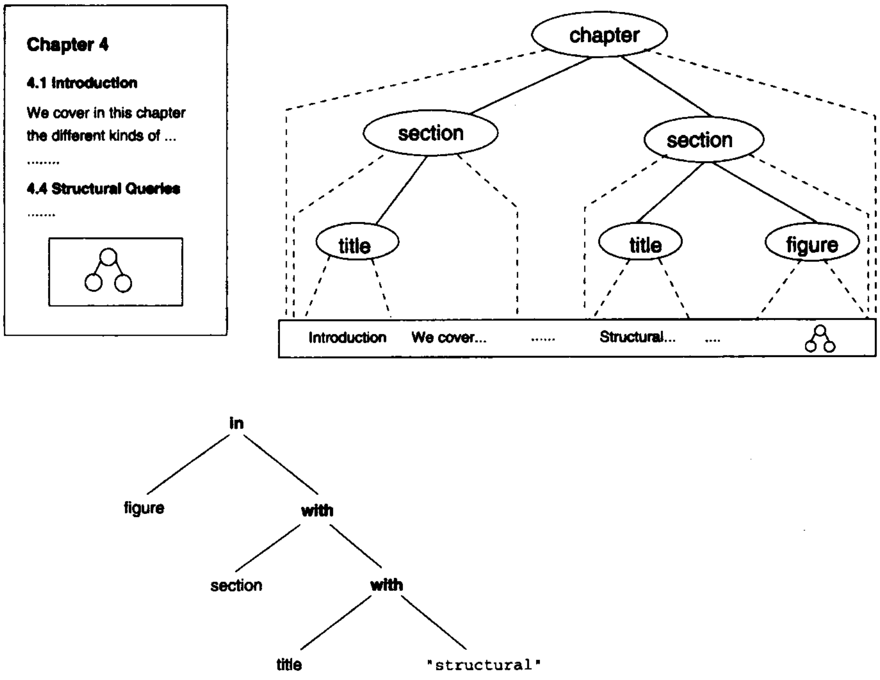

Figure 4.3 An example of a hierarchical structure: the page of a book, its schematic view, and a parsed query to retrieve the figure.

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

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,

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

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,

$$\begin{aligned}
Standard\_Rochio: \quad \vec{q}_m &= \alpha\, \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
Ide\_Regular: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
Ide\_Dec\_Hi: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma\ max_{non\text{-}relevant}(\vec{d}_j)
\end{aligned}$$

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

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

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

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

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 $\vec{s}_u = (s_{u,1}, s_{u,2}, \dots, s_{u,n})$ and $\vec{s}_v = (s_{v,1}, s_{v,2}, \dots, s_{v,n})$ 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*

$$s_{u,v} = \frac{\vec{s}_u \cdot \vec{s}_v}{|\vec{s}_u| \times |\vec{s}_v|} \qquad \mathbf{(5.10)}$$

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., $d_j \in D_l$, $s_u \in V_l$).

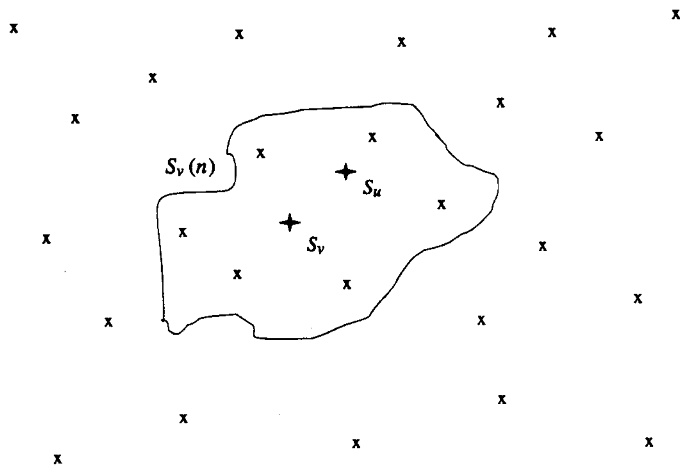

Figure 5.1 Stem s_u as a neighbor of the stem s_v.

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

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

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

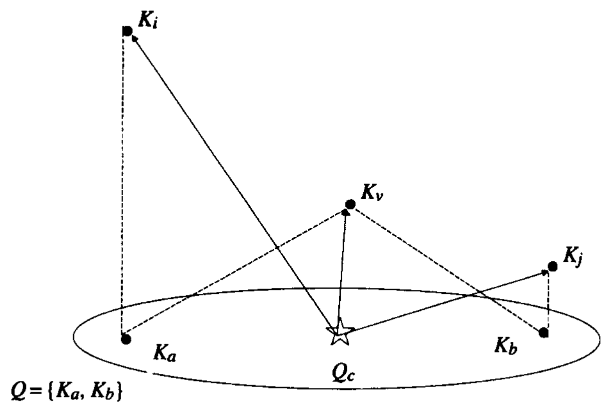

Figure 5.2 The distance of a given term k_v to the query centroid Q_c might be quite distinct from the distances of k_v to the individual query terms.

(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

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

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

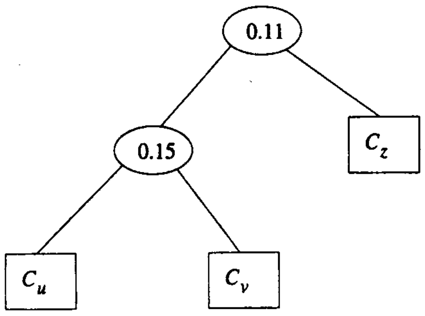

Figure 5.3 Hierarchy of three clusters (inter-cluster similarities indicated in the ovals) generated by the complete link algorithm.

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.