Chapter 5

Query Languages

We cover in this chapter the different kind 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 kind of queries as those answered by a system based on keyword ranking (as done by Web search engines) or on a hypertext model. In Chapter 7 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/she 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 (that is, the text syntax) to find relevant documents. In that sense, the system may fail to find the relevant answers (see Chapter [**REF**]). For this reason, a number of techniques meant to enhance the usefulness of the queries exist. Some examples are the expansion of a word to the set of its synonyms or the use of a thesaurus, stemming to put together all the derivatives of the same word, and others. Moreover, some words which are very frequent and do not carry meaning (such as `"the"`), called *stop-words*, may be removed. This subject is covered in Chapter [**REF**]. 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 "keywords" the first ones.

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 call simply "documents" those retrieval units, 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 a 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 in Internet and by CD-ROM publishers.

5.1 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 **??**, are also considered as basic queries.

5.1.1 Single-Word Queries

The most elementary query that can be formulated in a text retrieval system is the word. Text documents are assumed to be essentially long sequences of words. Although some models present a more general view, virtually all models allow seeing the text in this perspective and to search words. Some models (such as the full text model) are also able to see the internal division of words into letters. Those last models permit searching other types of patterns, which are covered in Section 5.2. 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 specifying 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 administrator 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. The first is called "term frequency" and counts the number of times a word appears inside a document. The second is called "inverse document frequency" and is based on a counting of the number of documents in which a word appears. See Chapter 2 for more details.

Additionally, the exact positions where the word appears in the text may be required. This might be useful for the user interface to highlight each occurrence.

5.1.2 Context Queries

Many systems complement single-word queries with the ability to search words in a given *context*, that is, near other words. As detailed in Chapter [**REF**], words which appear near each other may signal 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.

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 then the word `"retrieval"`. In phrase queries it is normally understood that the separators which are in the text need not be the same as in the query (e.g. two spaces versus one space), and even uninteresting words are not considered at all. For instance, the previous example could match in a text such as `"...enhance the retrieval..."`. Although this feature is very useful 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 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 Chapter [**REF**] 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.

5.1.3 Boolean Queries

The oldest (and still heavily used) mean to combine 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 5.1 shows an example.

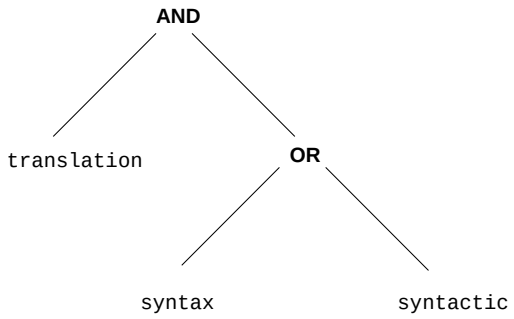

Figure 5.1: An example of a query syntax tree. It will retrieve all the documents which contain the word "translation" as well as either the word "syntax" or the word "syntactic".

The operators most commonly used, given two basic queries or boolean sub-expressions 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 it is probably not what the user wants. The BUT operator, instead, sets the universe of retrievable elements to the result of e_1 [1].

Apart from selecting the appropriate documents, it is up to the system to present those documents sorted by some criterion, to highlight the occurrences of the words mentioned in the query, and to allow feedback by taking the answer set as a basis to reformulate the query. The information needed to achieve this must be also provided as part of the result set.

With classic boolean systems, no ranking of the retrieved documents is 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, so that they take many operands and instead of forcing an element to appear in *all* operands (AND) or *one* operand (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 [**REF**]).

[1] 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, such as NOT does.

5.1.4 Natural Language

Pushing the fuzzy boolean model even further, the distinction between AND and OR could be completely blurred, so that a query becomes simply an enumeration of words and context queries which are of interest to the user, and all the documents matching some query are retrieved, giving more weight to those matching more parts of the query. The negation can be handled by letting the user express 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 reexpressed 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 can be searched but only a small set of hopefully representative keywords is extracted from each document). Natural language querying is explained in more detail in Chapter [**REF**].

5.2 Pattern Matching

In this section we discuss more specific query formulations (based on the concept of a *pattern*) which allow retrieving 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 be found 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 specifying some types of patterns, which range from very simple (for example words) to rather complex (such as regular expressions). The more powerful the set of patterns allowed, the more involved queries can the user formulate and the more complex is the implementation of the search, in general. The most used types of patterns are:

Words: a string (sequence of characters) which must be a word in the text (see section 5.1). 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 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 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 which lexicographically lies between them. 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 an 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. For instance, the edit distance between "color" and "colour" is one, while the edit distance between "survey" and "surgery" is two. This distance has been found superior to model errors than other more complex methods such as the Soundex system. 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 to the search pattern. Under this 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" or "wer" are at edit distance 1 to "flower"). Variations on this distance model are of use in computational biology to search on DNA or protein sequences.

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. some patterns positions match with a set of characters. 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. some pattern position 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 allowing that some parts of the pattern match exactly and others with errors.

5.3 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 text structure. The text collections tend to have some structure built into them, and the choice of being able 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.

Mixing contents and structure in queries allows posing 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 structural queries. In the boolean syntax tree (recall the example of Figure 5.1) 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 5.2 illustrates them. Although structured query languages should be amenable for ranking, this is still an open problem.

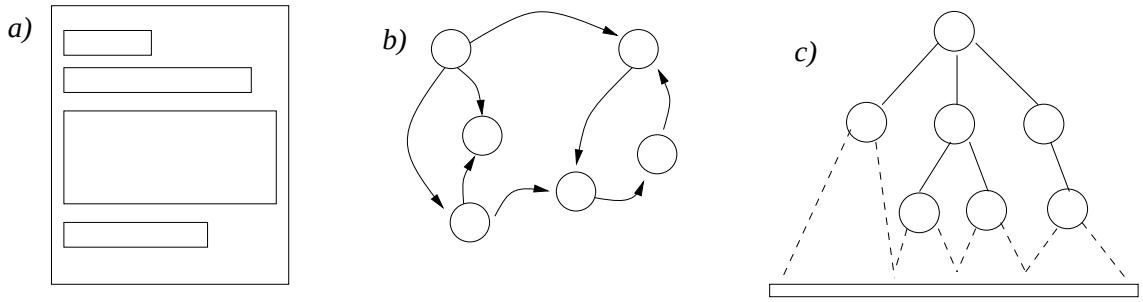

Figure 5.2: The three main structures: *a*) form-like fixed structure, *b*) hypertext structure and *c*) hierarchical structure.

In which 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 querying 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.

5.3.1 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, but only rarely could they appear in any order or appear repeatedly across the document or could the document have text not classified under any field. They 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 he sent to a given person with `"footbal"` in the subject field. However, the model is inadequate to represent the hierarchical structure present in a 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. therefore 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 the database as a text allows querying the textual fields with much more power than what is common in relational database systems. On the other hand, relational databases may use better their knowledge on the data types involved to build specialized and more efficient indices. A number of approaches to mix both trends have been proposed in the last 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 SFQL, which is covered in section 5.4.

5.3.2 Hypertext

Hypertexts probably represent the opposite trend with respect to structuring power. A hypertext is a directed graph where the nodes hold some text and the links represent connections among nodes or among positions inside the nodes. Hypertexts 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 hypertext began as a merely navigational activity. That is, the user had to manually traverse the hypertext nodes following links to search what he/she 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 connection structure.

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 hypertext based on their content and their structure. This problem is covered in detail in Chapter [**REF**].

5.3.3 Hierarchical Structures

An intermediate structuring model which lies between fixed structure and hypertext is the hierarchical structure. This represents a recursive decomposition of the text and it is a natural model for many text collections (e.g. books, articles, legal documents, structured programs, etc.). Figure 5.3 shows an example of such structure.

On the other hand, the simplification from hypertext to a hierarchy allows the use 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.

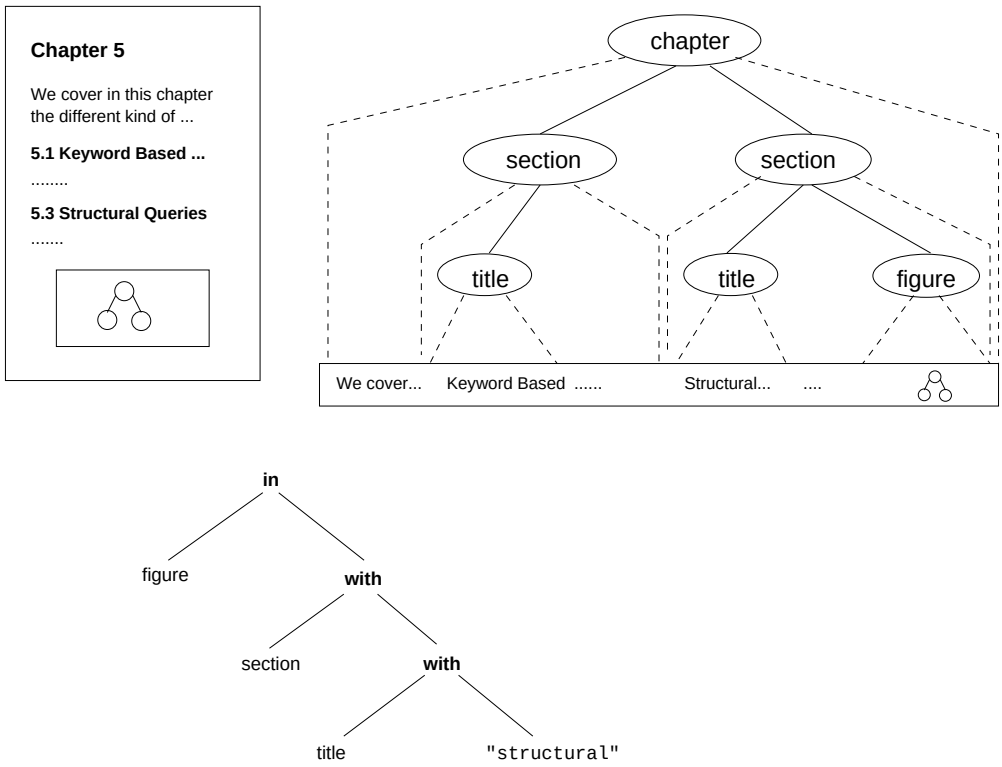

Figure 5.3: An example of a hierarchical structure: the page of a book, it schematic view, and a parsed query to retrieve the figure.

A Sample of Hierarchical Models

PAT Expressions are built on the same index of the text, i.e. there is no special index on structure. The structure is assumed to be marked in the text by *tags* (as in HTML), and therefore the structure 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 (since the tags need not be especially designed to be tags, e.g. one can define that the end-of-lines are the marks in order to define a structure of lines). This also allows a very efficient implementation and no additional space overhead for the structure.

Each expression 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 nor 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 (which can not be determined beforehand) then the result is converted to positions. Also, the dynamic definition of regions is flexible but requires that

the structure can be expressed using tags (also called "markup"), which for instance does not occur in some structured programming languages.

Overlapped Lists can be seen as an evolution from PAT Expressions. The model allows that the areas of a region overlap, but not nest. This elegantly solves the problems of mixing regions and sets of positions. The model considers the use of an inverted list (see Chapter 7) where not only the words but also the regions are indexed.

Apart from the operations of PAT Expressions, the model allows performing set union, and to combine regions. Combination means selecting the minimal text areas including 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.

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

Lists of References is an attempt to uniformize definition and querying of structured text, using a common language. The language goes beyond querying structured text, so we restrict our attention to the subset of our interest.

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 has also navigational features).

A static hierarchical structure makes it possible to speak in terms of direct ancestorship of nodes, a concept difficult to express when the structure is dynamic. The language allows 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 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.

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 of 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 is 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 short 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 relies on a single primitive: tree inclusion. The idea of tree inclusion is, seeing both the structure of the database and the query (a pattern on structure) as trees, find 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, and 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 problem intractable (NP-hard in many cases). Even the good cases have an inefficient solution in practice.

Discussion

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

Static or Dynamic Structure: as seen, in a static structure there are one or more explicit hierarchies (which can be queried, e.g., by ancestorship), 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 on the text markup, while a dynamic one is more flexible to build arbitrary structures.

Restrictions on the structure: the text or the answers may have restrictions about nesting and/or overlapping. In some cases those 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 models the text content is merely seen as a secondary source of information, used only to restrict the matches of structural elements. This is the reverse side of what happens to structure in classical models, where it is used only to restrict text matches. It is important that the integration of content and structure gives both aspects all their power and that integrates elegantly both types of queries for further manipulation.

Query language: typical queries on structure allow selecting areas that contain or not others, or are contained or not in others, that follow or are followed or are close to others, and set manipulation. Many of them are implemented in most models, although each models has unique features. Some kind of standardization, expressiveness taxonomy or formal categorization would be highly desirable and does not yet exist.

5.4 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 the Chapters [**REF**,**REF**] on Library Systems and Digital Libraries. The most important are:

Z39.50: is a protocol, approved as 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, despite the appearance of the client user interface and the query language of the host database. This database is assumed to be a text collection with some fixed fields (although it is more flexible than usual). The use of this protocol is very extended, for instance it is used internally by WAIS. 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 based on bibliographical information (using the MARC format [**REF**]), it has been extended to query other types of information as well.

WAIS: (Wide Area Information Service) is a suite of protocols which was popular at the beginning of the 90s before the boom of the WWW. The goal of WAIS was to be a network publishing protocol and be able to query databases through Internet.

In the CD-ROM publishing arena, there are several proposals for query protocols. The main goal of these protocols is to provide "disc interchangeability". This means more flexibility in data communication between primary information providers and end users. 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 them:

CCL: (Common Command Language) is a NISO proposal (Z39.58 or ISO 8777) based in 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 in the classical boolean model.

CD-RDx: (Compact Disk Read only Data exchange) uses a client-server architecture and has been implemented in most used platforms. The client is generic while the server is designed and provided by the CD-ROM publisher which includes it with the database in the CD-ROM. It allows fixed-length fields, images and audio, and is being supported by some US national agencies as 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 GSML. 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 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 in a relational model, which is not always the best choice for a document database.

5.5 Conclusions and Trends

We reviewed in this chapter the main aspects of the query languages that retrieve information from textual databases. We ranged from the most classical tools to the most novel capabilities that are emerging, from searching words to extended patterns, from the boolean model to querying structures. Table 5.1 shows the different queries allowed in the different models. Although the probabilistic and the Bayesian belief network (BBN) model are based on word queries, they can incorporate set operations.

We present in Figure 5.4 the types of operations we covered and how can they be structured (not all them exist in all models and not all 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 can be formed by words and by regular expressions (skipping the ability to allow errors).

The subject of query languages for text databases is definitely moving to a higher flexibility. While the text models are moving to the goal of achieving a better understanding of the user needs

Model	Queries allowed
Boolean	word, set operations
Vector	words
Probabilistic	words
BBN	words
Full-text	words, set operations, pattern matching

Table 5.1: Relationship between type of queries and models.

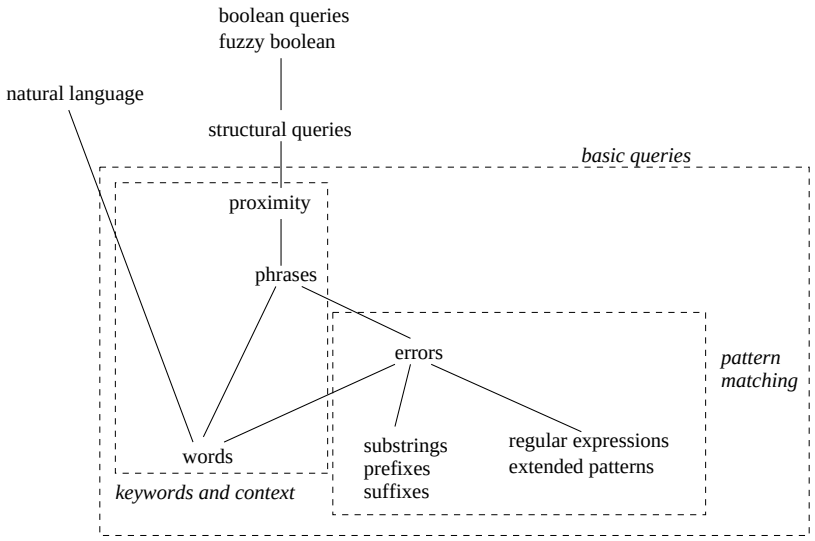

Figure 5.4: The types of queries covered and how are they structured.

(by providing relevance feedback, for instance) the query languages are allowing more and more power in the specification of the query. Not only extended patterns and searching allowing errors permit to find patterns without complete knowledge of what is wanted, but it is also becoming more common to query on the structure of the text and not only on its content.

5.6 Bibliographic Discussion

The material on classical query languages (most simple patterns, boolean model, and fixed structure) is based on current commercial systems, such as Fulcrum, Verity and others, as well as on non-commercial systems such as Glimpse (Manber and Wu 1993) and Igrep (Araújo, Navarro and Ziviani 1997).

The fuzzy-boolean model is described in (Salton, Fox and Wu 1982). The Levenshtein distance is described in (Levenshtein 1966) and (Apostolico and Galil 1985). Soundex is explained in (Knuth 1973). A comparison of the efectiveness of different similarity models is given in (Nesbit 1986). A good source on regular expressions is (Hopcroft and Ullman 1979). A rich language on extended

patterns is described in (Wu and Manber 1992).

A classical reference on hypertext is (Conklin 1987). The WebGlimpse system is presented in (Manber, Smith and Gopal 1997). The discussion of hierarchical text is partially based on (Baeza-Yates and Navarro 1996). The original proposals are: PAT Expressions (Salminen and Tompa 1992), Overlapped Lists (Clarke, Cormack and Burkowski 1995) and the new improved proposal (Dao, Sacks-Davis and Tohm 1996), Lists of References (MacLeod 1991), Proximal Nodes (Navarro and Baeza-Yates 1997) and Tree Matching (Kilpeläinen and Mannila 1993). PAT Expressions are the basic model of the PAT Text Searching System (Gonnet 1987).

More information on *Z39.50* can be obtained from (Z39.50 - ANSI/NISO Standards 1995). More information on *WAIS* is given in (Kahle and Medlar, 1991). For details on *SFQL* see (IEEE SFQL, 1992).

Bibliography

[AG85] A. Apostolico and Z. Galil. *Combinatorial Algorithms on Words*. Springer-Verlag, New York, 1985.

[ANZ97] M. Araújo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In *Proc. WSP'97*, pages 2–20. Carleton University Press, 1997.

[BYN96] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.

[CCB95] C. Clarke, G. Cormack, and F. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 1995.

[Con87] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.

[DSDT96] T. Dao, R. Sacks-Davis, and J. Tohm. Indexing structured text for queries on containment relationships. In *Proc. 7th Australasian Conference*, 1996.

[Gon87] G. Gonnet. Examples of PAT applied to the Oxford English Dictionary. Technical Report OED-87-02, UW Centre for the New OED and Text Research, Univ. of Waterloo, 1987.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.

[KM91] B. Kahle and A. Medlar. An information server for corporate users: Wide Area Information Servers. *ConneXions - The Interoperability Report*, 5(11):2–9, 1991. Available from ftp://think.com/wais/wais-corporate-paper.text.

[KM93] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. ACM SIGIR'93*, pages 214–222, 1993.

[Knu73] D. Knuth. *The Art of Computer Programming*, volume 3: Searching and Sorting. Addison-Wesley, 1973.

[Mac91] I. MacLeod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[MSG97] U. Manber, M. Smith, and B. Gopal. Webglimpse: combining browsing and searching. In *Proc. of USENIX Technical Conference*, 1997.

[MW93] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. Technical Report 93-34, Dept. of CS, Univ. of Arizona, Oct 1993.

[NBY97] G. Navarro and R. Baeza-Yates. Proximal Nodes: a language to query document databases by content and structure. *ACM TOIS*, 15(4):401–435, October 1997.

[Nes86] J. Nesbit. The accuracy of approximate string matching algorithms. *J. of Computer-Based Instruction*, 13(3):80–83, 1986.

[oODS92] IEEE Standards Committee on Optical Disk and Multimedia Platforms (SCODMP). Ieee sfql. Technical report, IEEE, Washington, USA, 1992.

[SFW82] G. Salton, E. Fox, and H. Wu. Extended boolean information retrieval. Technical Report TR 82-511, Dept. of Computer Science, Cornell Univ., August 1982.

[ST92] A. Salminen and F. Tompa. PAT expressions: an algebra for text search. Technical Report OED-92-02, UW Centre for the New Oxford English Dictionary, July 1992.

[Sta95] ANSI/NISO Standards. Z39.50-information retrieval: Application service definition and protocol specification. Technical report, Washington, USA, 1995. See http://lcweb.loc.gov/z3950/agency.

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, October 1992.