10.6.6 Using Hyperlinks to Organize Retrieval Results

Although the SuperBook authors describe it as a hypertext system, it is actually better thought of as a means of showing search results in the context of a structure that users can understand and view all at once. The hypertext component was not analyzed separately to assess its importance, but it usually is not mentioned by the authors when describing what is successful about their design. In fact, it seems to be responsible for one of the main problems seen with the revised version of the system — that users tend to wander off (often unintentionally) from the pages they are reading, thus causing the time spent on a given topic to be longer for SuperBook in some cases. (Using completion time to evaluate users on browsing tasks can be problematic, however, since by definition browsing is a casual, unhurried process [804].)

This wandering may occur in part because SuperBook uses a non-standard kind of hypertext, in which any word is automatically linked to occurrences of the same word in other parts of the document. This has not turned out to be how hypertext links are created in practice. Today, hyperlinked help systems and hyperlinks on the Web make much more discriminating use of hyperlink connections (in part since they are usually generated by an author rather than automatically). These links tend to be labeled in a somewhat meaningful manner by their surrounding context. Back-of-the-book indexes often do not contain listings of every occurrence of a word, but rather to the more important uses or the beginnings of series of uses. Automated hypertext linking should perhaps be based on similar principles. Additionally, at least one study showed that users formed better mental models of a small hypertext system that was organized hierarchically than one that allowed more flexible access [226]. Problems relating to navigation of hypertext structure have long been suspected and investigated in the hypertext literature [181, 551, 440, 334].

More recent work has made better use of hyperlink information for providing context for retrieval results. Some of this work is described below.

Cha-Cha: SuperBook on the Web

The Cha-Cha intranet search system [164] extends the SuperBook idea to a large heterogeneous Web site such as might be found in an organization's intranet. Figure 10.21 shows an example. This system differs from SuperBook in several ways. On most Web sites there is no existing real table of contents or category structure, and an intranet like those found at large universities or large corporations is usually not organized by one central unit. Cha-Cha uses link structure present within the site to create what is intended to be a meaningful organization on top of the underlying chaos. After the user issues a query, the shortest paths from the root page to each of the search hits are recorded and a subset of these are selected to be shown as a hierarchy, so that each hit is shown only once. (Users can begin with a query, rather than with a table of contents view.) If a user does not know to use the term 'health center' but instead queries on 'medical center,' if 'medical' appears as a term in a document within

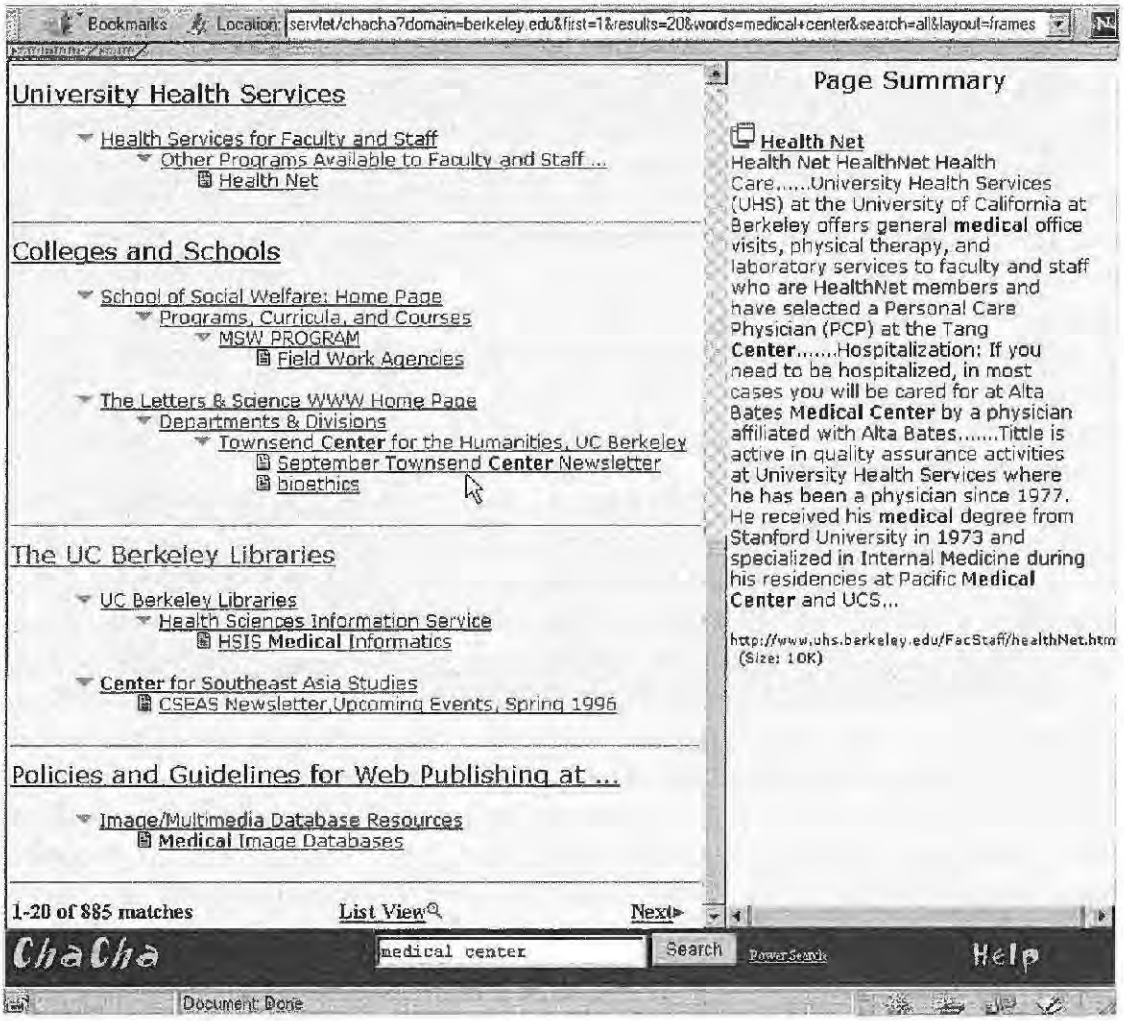


Figure 10.21 The Cha-Cha interface for showing Web intranet search results in context displaying results on the query 'medical centre'[164].

the health center part of the Web, the home page (or starting point) of this center will be presented as well as the more specific hits. Users can then either query or navigate within a subset of sites if they wish. The organization produced by this simple method is surprisingly comprehensible on the UC Berkeley site. It seems especially useful for providing the information about the sources (the Web server) associated with the search hits, whose titles are often cryptic.

The AMIT system [826] also applies the basic ideas behind SuperBook to the Web, but focuses on a single-topic Web site, which is likely to have a more reasonable topic structure than a complex intranet. The link structure of the Web site is used as contextualizing information but all of the paths to a given document are shown and focus-plus-context is used to emphasize subsets of the document space. The WebTOC system [585] is similar to AMIT but focuses on showing the structure and number of documents within each Web subhierarchy, and is not tightly coupled with search.

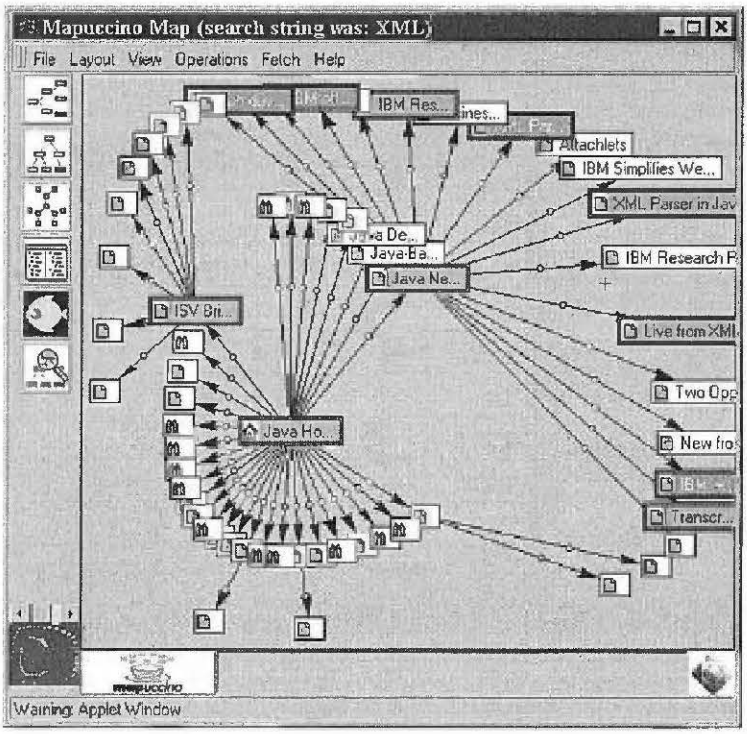


Figure 10.22 Example of a Web subset visualized by Mapuccino (courtesy of M. Jacovi, B. Shaul and Y. Maarek).

Mapuccino: Graphical Depiction of Link Structure

The Mapuccino system (formerly WebCutter) [527] allows the user to issue a query on a particular Web site. The system crawls the site in real-time, checking each encountered page for relevance to the query. When a relevant page is found, the weights on that page's outlinks are increased. Thus, the search is based partly on an assumption that relevant pages will occur near one another in the Web site. The subset of the Web site that has been crawled is depicted graphically in a nodes-and-links view (see Figure 10.22). This kind of display does not provide the user with information about what the *contents* of the pages are, but rather only shows their link structure. Other researchers have also investigated spreading activation among hypertext links as a way to guide an information retrieval system, e.g., [278, 555].

10.6.7 Tables

Tabular display is another approach for showing relationships among retrieval documents. The Envision system [273] allows the user to organize results according to metadata such as author or date along the X and Y-axes, and uses graphics to show values for attributes associated with retrieved documents within each cell (see Figure 10.23). Color, shape, and size of an iconic representation of a document are used to show the computed relevance, the type of document, or

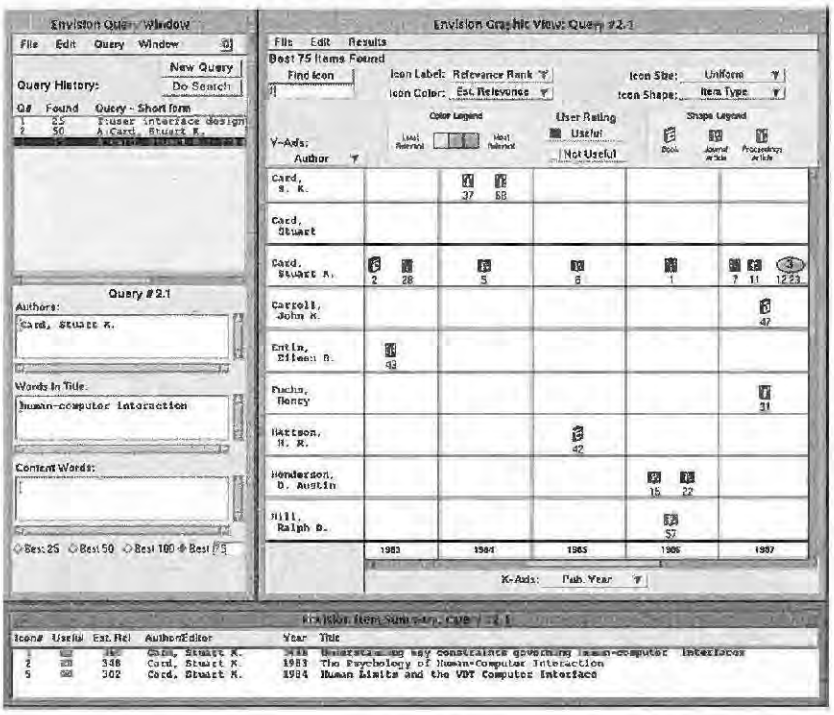


Figure 10.23 The Envision tabular display for graphically organizing retrieved documents [270].

other attributes. Clicking on an icon brings up more information about the document in another window. Like the WebCutter system, this view provides few cues about how the documents are related to one another in terms of their content or meaning. The SenseMaker system also allows users to group documents into different views via a table-like display [51], including a Scatter/Gather [203] style view. Although tables are appealing, they cannot show the intersections of many different attributes; rather they are better for pairwise comparisons. Another problem with tables for display of textual information is that very little information can be fitted on a screen at a time, making comparisons difficult.

The Table Lens [666] is an innovative interface for viewing and interactively reorganizing very large tables of information (see Figure 10.24). It uses focus-plus-context to fit hundreds of rows of information in a space occupied by at most two dozen rows in standard spreadsheets. And because it allows for rapid reorganization via sorting of columns, users can quickly switch from a view focused around one kind of metadata to another. For example, first sorting documents by rank and then by author name can show the relative ranks of different articles by the same author. A re-sort by date can show patterns in relevance scores with respect to date of publication. This rapid re-sorting capability helps circumvent the problems associated with the fact that tables cannot show many simultaneous intersections.

Another variation on the table theme is that seen in the Perspective Wall [530] in which a focus-plus-context display is used to center information currently

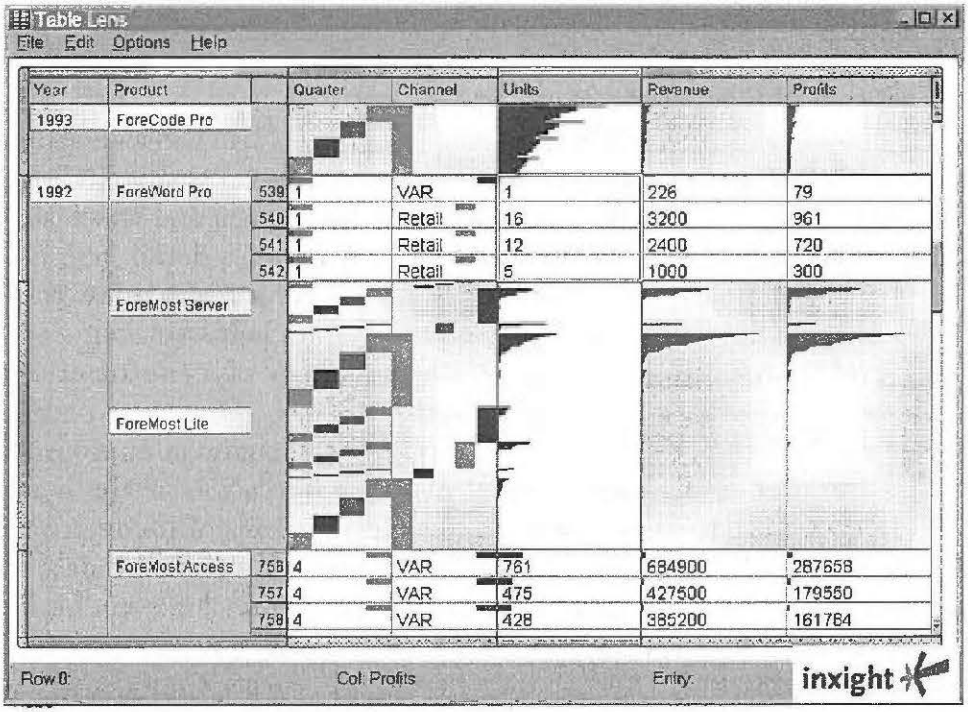


Figure 10.24 The TableLens visualization [666].

of interest in the middle of the display, compressing less important information into the periphery on the sides of the wall. The idea is to show in detail the currently most important information while at the same time retaining the context of the rest of the information. For example, if viewing documents in chronological order, the user can easily tell if they are currently looking at documents in the beginning, middle, or end of the time range.

These interfaces have not been applied to information access tasks. The problem with such displays when applied to text is that they require an attribute that can be shown according to an underlying order, such as date. Unfortunately, information useful for organizing text content, such as topic labels, does not have an inherent meaningful order. Alphabetical order is useful for looking up individual items, but not for seeing patterns across items according to adjacency, as in the case for ordered data types like dates and size.

10.7 Using Relevance Judgements

An important part of the information access process is query reformulation, and a proven effective technique for query reformulation is *relevance feedback*. In its original form, relevance feedback refers to an interaction cycle in which the user selects a small set of documents that appear to be relevant to the query, and the system then uses features derived from these selected relevant documents to revise the original query. This revised query is then executed and a new set of documents is returned. Documents from the original set can appear in the new results

list, although they are likely to appear in a different rank order. Relevance feedback in its original form has been shown to be an effective mechanism for improving retrieval results in a variety of studies and settings [702, 343, 127]. In recent years the scope of ideas that can be classified under this term has widened greatly.

Relevance feedback introduces important design choices, including which operations should be performed automatically by the system and which should be user initiated and controlled. Bates discusses this issue in detail [66], asserting that despite the emphasis in modern systems to try to automate the entire process, an intermediate approach in which the system helps automate search at a *strategic* level is preferable. Bates suggests an analogy of an automatic camera versus one with adjustable lenses and shutter speeds. On many occasions, a quick, easy method that requires little training or thought is appropriate. At other times the user needs more control over the operation of the machinery, while still not wanting to know about the low level details of its operation.

A related idea is that, for any interface, control should be described in terms of the task being done, not in terms of how the machine can be made to accomplish the task [607]. Continuing the camera analogy, the user should be able to control the mood created by the photograph, rather than the adjustment of the lens. In information access systems, control should be over the kind of information returned, not over which terms are used to modify the query. Unfortunately it is often quite difficult to build interfaces to complex systems that behave in this manner.

10.7.1 Interfaces for Standard Relevance Feedback

A standard interface for relevance feedback consists of a list of titles with checkboxes beside the titles that allow the user to mark relevant documents. This can imply either that unmarked documents are not relevant or that no opinion has been made about unmarked documents, depending on the system. Another option is to provide a choice among several checkboxes indicating relevant or not relevant (with no selection implying no opinion). In some cases users are allowed to indicate a value on a relevance scale [73]. Standard relevance feedback algorithms usually do not perform better given negative relevance judgement evidence [225], but machine learning algorithms can take advantage of negative feedback [629, 460].

After the user has made a set of relevance judgements and issued a search command, the system can either automatically reweight the query and re-execute the search, or generate a list of terms for the user to select from in order to augment the original query. (See Figure 10.25, taken from [448].) Systems usually do not suggest terms to remove from the query.

After the query is re-executed, a new list of titles is shown. It can be helpful to retain an indicator such as a marked checkbox beside the documents that the user has already judged. A difficult design decision concerns whether or not to show documents that the user has already viewed towards the top of the ranked list [1]. Repeatedly showing the same set of documents at the top may inconvenience a user who is trying to create a large set of relevant documents,

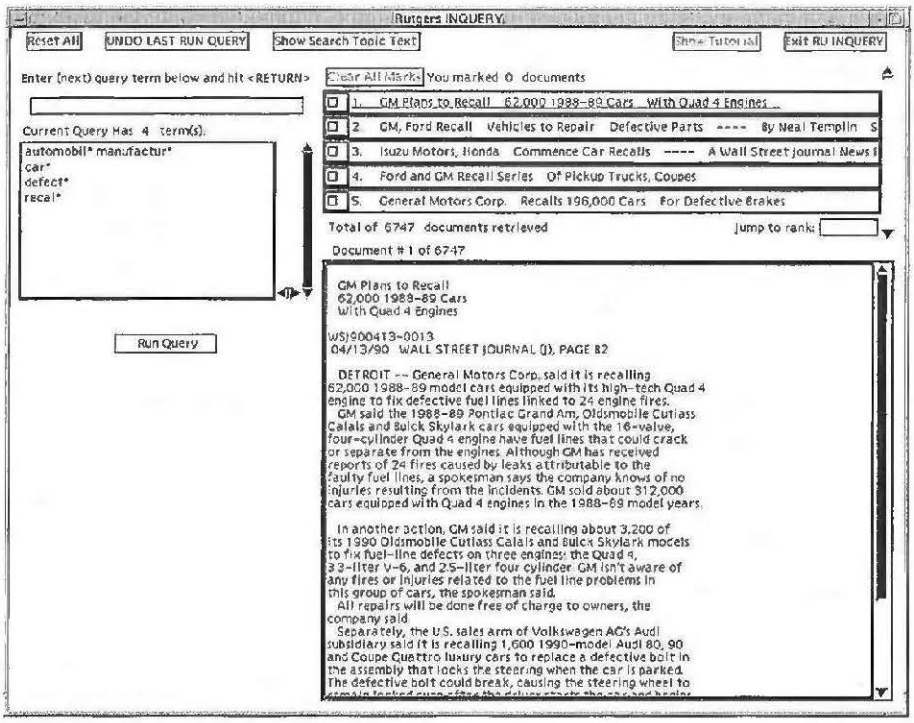


Figure 10.25 An example of an interface for relevance feedback [448].

but at the same time, this can serve as feedback indicating that the revised query does not downgrade the ranking of those documents that have been found especially important. One solution is to retain a separate window that shows the rankings of only the documents that have not been retrieved or ranked highly previously. Another solution is to use smaller fonts or gray-out color for the titles of documents already seen.

Creating multiple relevance judgements is an effortful task, and the notion of relevance feedback is unfamiliar to most users. To circumvent these problems, Web-based search engines have adopted the terminology of 'more like this' as a simpler way to indicate that the user is requesting documents similar to the selected one. This 'one-click' interaction method is simpler than standard relevance feedback dialog which requires users to rate a small number of documents and then request a reranking. Unfortunately, in most cases relevance feedback requires many relevance judgements in order to work well. To partly alleviate this problem, Aalbersberg [1] proposes incremental relevance feedback which works well given only one relevant document at a time and thus can be used to hide the two-step procedure from the user.

10.7.2 Studies of User Interaction with Relevance Feedback Systems

Standard relevance feedback assumes the user is involved in the interaction by specifying the relevant documents. In some interfaces users are also able to

select which terms to add to the query. However, most ranking and reweighting algorithms are difficult to understand or predict (even for the creators of the algorithms!) and so it might be the case that users have difficulties controlling a relevance feedback system explicitly.

A recent study was conducted to investigate directly to what degree user control of the feedback process is beneficial. Koenemann and Belkin [448] measured the benefits of letting users 'under the hood' during relevance feedback. They tested four cases using the Inquery system [772]:

- **Control** No relevance feedback; the subjects could only reformulate the query by hand.
- **Opaque** The subjects simply selected relevant documents and saw the revised rankings.
- **Transparent** The subjects could see how the system reformulated the queries (that is, see which terms were added — the system did not reweight the subjects' query terms) and the revised rankings.
- **Penetrable** The system is stopped midway through the reranking process. The subjects are shown the terms that the system would have used for opaque and transparent query reformulation. The subjects then select which, if any, of the new terms to add to the query. The system then presents the revised rankings.

The 64 subjects were much more effective (measuring precision at a cut-off of top 5, top 10, top 30, and top 100 documents) with relevance feedback than without it. The penetrable group performed significantly better than the control, with the opaque and transparent performances falling between the two in effectiveness. Search times did not differ significantly among the conditions, but there were significant differences in the number of feedback iterations. The subjects in the penetrable group required significantly fewer iterations to achieve better queries (an average of 5.8 cycles in the penetrable group, 8.2 cycles in the control group, 7.7 cycles in the opaque group, and surprisingly, the transparent group required more cycles, 8.8 on average). The average number of documents marked relevant ranged between 11 and 14 for the three conditions. All subjects preferred relevance feedback over the baseline system, and several remarked that they preferred the 'lazy' approach of selecting suggested terms over having to think up their own.

An observational study on a TTY-based version of an online catalog system [338] also found that users performed better using a relevance feedback mechanism that allowed manual selection of terms. However, a later observational study did not find overall success with this form of relevance feedback [337]. The authors attribute these results to a poor design of a new graphical interface. These results may also be due to the fact that users often selected only one relevant document before performing the feedback operation, although they were using a system optimized from multiple document selection.

10.7.3 Fetching Relevant Information in the Background

Standard relevance feedback is predicated on the goal of improving an ad hoc query or building a profile for a routing query. More recently researchers have begun developing systems that monitor users' progress and behavior over long interaction periods in an attempt to predict which documents or actions the user is likely to want in future. These systems are called semi-automated *assistants* or recommender 'agents,' and often make use of machine learning techniques [565]. Some of these systems require explicit user input in the form of a goal statement [406] or relevance judgements [629], while others quietly record users' actions and try to make inferences based on these actions.

A system developed by Kozierok and Maes [460, 536] makes predictions about how users will handle email messages (what order to read them in, where to file them) and how users will schedule meetings in a calendar manager application. The system 'looks over the shoulder' of the users, recording every relevant action into a database. After enough data has been accumulated, the system uses a nearest-neighbors method [743] to predict a user's action based on the similarity of the current situation to situations already encountered. For example, if the user almost always saves email messages from a particular person into a particular file, the system can offer to automate this action the next time a message from that person arrives [536]. This system integrates learning from both implicit and explicit user feedback. If a user ignores the system's suggestion, the system treats this as negative feedback, and accordingly adds the overriding action to the action database. After certain types of incorrect predictions, the system asks the user questions that allow it to adjust the weight of the feature that caused the error. Finally, the user can explicitly train the system by presenting it with hypothetical examples of input-action pairs.

Another system, Syskill and Webert [629], attempts to learn a user profile based on explicit relevance judgements of pages explored while browsing the Web. In a sense this is akin to standard relevance feedback, except the user judgements are retained across sessions and the interaction model differs: as the user browses a new Web page, the links on the page are automatically annotated as to whether or not they should be relevant to the user's interest.

A related system is Letizia [518], whose goal is to bring to the user's attention a percentage of the available next moves that are most likely to be of interest, given the user's earlier actions. Upon request, Letizia provides recommendations for further action on the user's part, usually in the form of suggestions of links to follow when the user is unsure what to do next. The system monitors the user's behavior while navigating and reading Web pages, and concurrently evaluates the links reachable from the current page. The system uses only implicit feedback. Thus, saving a page as a bookmark is taken as strong positive evidence for the terms in the corresponding Web page. Links skipped are taken as negative support for the information reachable from the link. Selected links can indicate positive or negative evidence, depending on how much time the user spends on the resulting page and whether or not the decision to leave a page quickly is later reversed. Additionally, the evidence for user interest remains persistent across

browsing sessions. Thus, a user who often reads kayaking pages is at another time reading the home page of a professional contact and may be alerted to the fact that the colleague's personal interests page contains a link to a shared hobby. The system uses a best-first search strategy and heuristics to determine which pages to recommend most strongly.

A more user-directed approach to prefetching potentially relevant information is seen in the Butterfly system [531]. This interface helps the user follow a series of citation links from a given reference, an important information seeking strategy [66]. The system automatically examines the document the user is currently reading and prefetches the bibliographic citations it refers to. It also retrieves lists of articles that cite the focus document. The underlying assumption is that the services from which the citations are requested do not respond immediately. Rather than making the user wait during the delay associated with each request, the system handles many requests in parallel and the interface uses graphics and animations to show the incrementally growing list of available citations. The system does not try to be clever about which cites to bring first; rather the user can watch the 'organically' growing visualization of the document and its citations, and based on what looks relevant, direct the system as to which parts of the citation space to spend more time on.

10.7.4 Group Relevance Judgements

Recently there has been much interest in using relevance judgements from a large number of different users to rate or rank information of general interest [672]. Some variations of this *social recommendation* approach use only similarity among relevance judgements by people with similar tastes, ignoring the representation of the information being judged altogether. This has been found highly effective for rating information in which taste plays a major role, such as movie and music recommendations [720]. More recent work has combined group relevance judgements with content information [64].

10.7.5 Pseudo-Relevance Feedback

At the far end of the system versus user feedback spectrum is what is informally known as pseudo-relevance feedback. In this method, rather than relying on the user to choose the top k relevant documents, the system simply assumes that its top-ranked documents are relevant, and uses these documents to augment the query with a relevance feedback ranking algorithm. This procedure has been found to be highly effective in some settings [760, 465, 12], most likely those in which the original query statement is long and precise. An intriguing extension to this idea is to use the output of clustering of retrieval results as the input to a relevance feedback mechanism, either by having the user or the system select the cluster to be used [359], but this idea has not yet been evaluated.

10.8 Interface Support for the Search Process

The user interface designer must make decisions about how to arrange various kinds of information on the computer screen and how to structure the possible sequences of interactions. This design problem is especially daunting for a complex activity like information access. In this section we discuss design choices surrounding the layout of information within complex information systems, and illustrate the ideas with examples of existing interfaces. We begin with a discussion of very simple search interfaces, those used for string search in 'find' operations, and then progress to multiwindow interfaces and sophisticated workspaces. This is followed by a discussion of the integration of scanning, selecting, and querying within information access interfaces and concludes with interface support for retaining the history of the search process.

10.8.1 Interfaces for String Matching

A common simple search need is that of the 'find' operation, typically run over the contents of a document that is currently being viewed. Usually this function does not produce ranked output, nor allow Boolean combinations of terms; the main operation is a simple string match (without regular expression capabilities). Typically a special purpose search window is created, containing a few simple controls (e.g., case-sensitivity, search forward or backward). The user types the query string into an entry form and string matches are highlighted in the target document (see Figure 10.26).

The next degree of complexity is the 'find' function for searching across small collections, such as the files on a personal computer's hard disk, or the history list of a Web browser. This type of function is also usually implemented as a simple string match. Again, the controls and parameter settings are shown at the top of a special purpose search window and the various options are set via checkboxes and entry forms. The difference from the previous example is that a results list is shown within the search interface itself (see Figure 10.27).

A common problem arises even in these very simple interfaces. An ambiguous state occurs in which the results for an earlier search are shown while the user is entering a new query or modifying the previous one. If the user types in

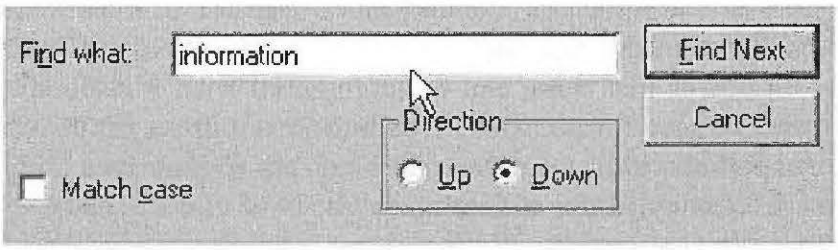


Figure 10.26 An example of a simple interface for string matching, from Netscape Communicator 4.05.

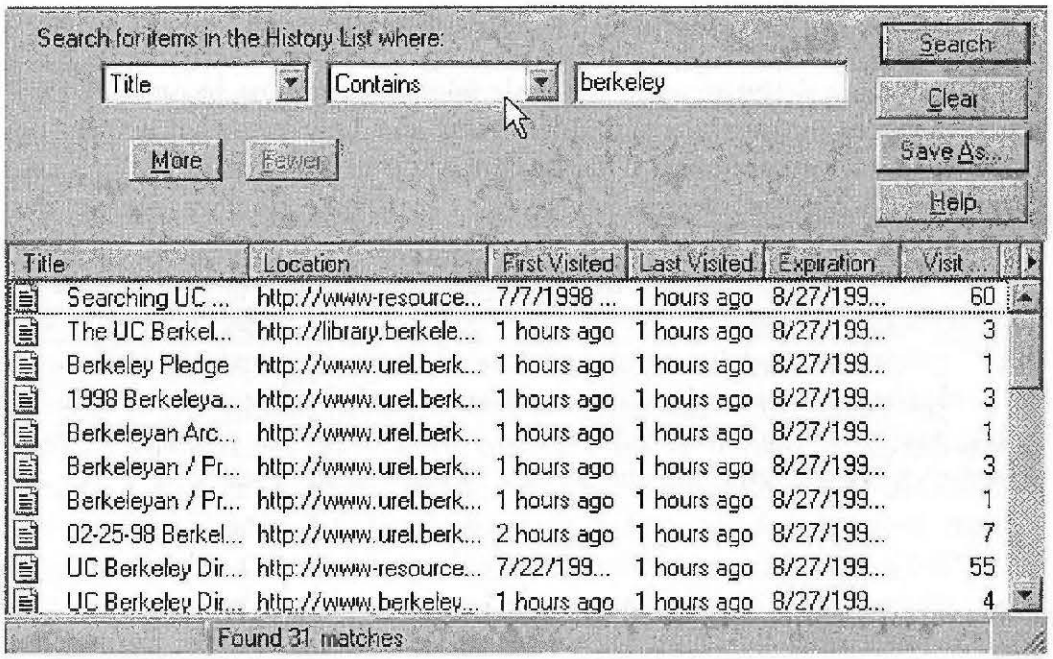


Figure 10.27 An example of an string matching over a list, in this case, a history of recently viewed Web pages, from Netscape Communicator 4.05.

new terms and but then does not activate the search, the interface takes on a potentially misleading state, since a user could erroneously assume that the old search hits shown correspond to the newly typed-in query. One solution for this problem is to clear the results list as soon as the user begins to type in a new query.

However, the user may want to refer to terms shown in the search results to help reformulate the query, or may decide not to issue the new query and instead continue with the previous results. These goals would be hampered by erasing the current result set as soon as the new query is typed. Another solution is to bring up a new window for every new query. However, this requires the user to execute an additional command and can lead to a proliferation of windows. A third, probably more workable solution, is to automatically 'stack' the queries and results lists in a compact format and allow the user to move back and forth among the stacked up prior searches.

Simple interfaces like these can be augmented with functionality that can greatly aid initial query formulation. Spelling errors are a major cause of void result sets. A spell-checking function that suggests alternatives for query terms that have low frequency in the collection might be useful at this stage. Another option is to suggest thesaurus terms associated with the query terms at the time the query terms are entered. Usually these kinds of information are shown after the query is entered and documents have been retrieved, but an alternative is to provide this information as the user enters the query, in a form of query preview.

10.8.2 Window Management

For search tasks more complex than the simple string matching find operations described above, the interface designer must decide how to lay out the various choices and information displays within the interface.

As discussed above, traditional bibliographic search systems use TTY-based command-line interfaces or menus. When the system responds to a command, the new results screen obliterates the contents of the one before it, requiring the user to remember the context. For example, the user can usually see only one level of a subject hierarchy at a time, and must leave the subject view in order to see query view or the document view. The main design choices in such a system are in the command or menu structure, and the order of presentation of the available options.

In modern graphical interfaces, the windowing system can be used to divide functionality into different, simultaneously displayed views [582]. In information access systems, it is often useful to link the information from one window to the information in another, for example, linking documents to their position in a table of contents, as seen in SuperBook. Users can also use the selection to cut and paste information from one window into another, for example, copy a word from a display of thesaurus terms and paste the word into the query specification form.

When arranging information within windows, the designer must choose between a *monolithic* display, in which all the windows are laid out in predefined positions and are all simultaneously viewable, *tiled windows*, and *overlapping windows*. User studies have been conducted comparing these options when applied to various tasks [725, 96]. Usually the results of these studies depend on the domain in which the interface is used, and no clear guidelines have yet emerged for information access interfaces.

The monolithic interface has several advantages. It allows the designer to control the organization of the various options, makes all the information simultaneously viewable, and places the features in familiar positions, making them easier to find. But monolithic interfaces have disadvantages as well. They often work best if occupying the full viewing screen, and the number of views is inherently limited by the amount of room available on the screen (as opposed to overlapping windows which allow display of more information than can fit on the screen at once). Many modern work-intensive applications adopt a monolithic design, but this can hamper the integration of information access with other work processes such as text editing and data analysis. Plaisant *et al.* [644] discuss issues relating to coordinating information across different windows to providing overview plus details.

A problem for any information access interface is an inherent limit in how many kinds of information can be shown at once. Information access systems must always reserve room for a text display area, and this must take up a significant proportion of screen space in order for the text to be legible. A tool within a paint program, for example, can be made quite small while nevertheless remaining recognizable and usable. For legibility reasons, it is difficult to compress many of the information displays needed for an information access system (such

as lists of thesaurus terms, query specifications, and lists of saved titles) in this manner. Good layout, graphics, and font design can improve the situation; for example, Web search results can look radically different depending on spacing, font, and other small touches [580].

Overlapping windows provide flexibility in arrangement, but can quickly lead to a crowded, disorganized display. Researchers have observed that much user activity is characterized by movement from one set of functionally related windows to another. Bannon *et al.* [54] define the notion of a *workspace* — the grouping together of sets of windows known to be functionally related to some activity or goal — arguing that this kind of organization more closely matches users' goal structure than individual windows [96]. Card *et al.* [140] also found that window usage could be categorized according to a 'working set' model. They looked at the relationship between the demands of the task and the number of windows in use, and found the largest number of individual windows were in use when users transitioned from one task to another.

Based on these and other observations, Henderson and Card [420] built a system intended to make it easier for users to move between 'multiple virtual workspaces' [96]. The system uses a 3D spatial metaphor, where each workspace is a 'room,' and users transition between workspaces by 'moving' through virtual doors. By 'traveling' from one room to the next, users can change from one work context to another. In each work context, the application programs and data files that are associated with that work context are visible and readily available for reopening and perusal. The workspace notion as developed by Card *et al.* also emphasizes the importance of having sessions persist across time. The user should be able to leave a room dedicated to some task, work on another task, and three days later return to the first room and see all of the applications still in the same state as before. This notion of bundling applications and data together for each task has since been widely adopted by window manager software in workstation operating system interfaces.

Elastic windows [428] is an extension to the workspace or rooms notion to the organization of 2D tiled windows. The main idea is to make the transition easier from one role or task to another, by adjusting how much of the screen real estate is consumed by the current role. The user can enlarge an entire group of windows with a simple gesture, and this resizing automatically causes the rest of the workspaces to reduce in size so they all still fit on the screen without overlap.

10.8.3 Example Systems

The following sections describe the information layout and management approaches taken by several modern information access interfaces.

The InfoGrid Layout

The InfoGrid system [667] is a typical example of a monolithic layout for an information access interface. The layout assumes a large display is available

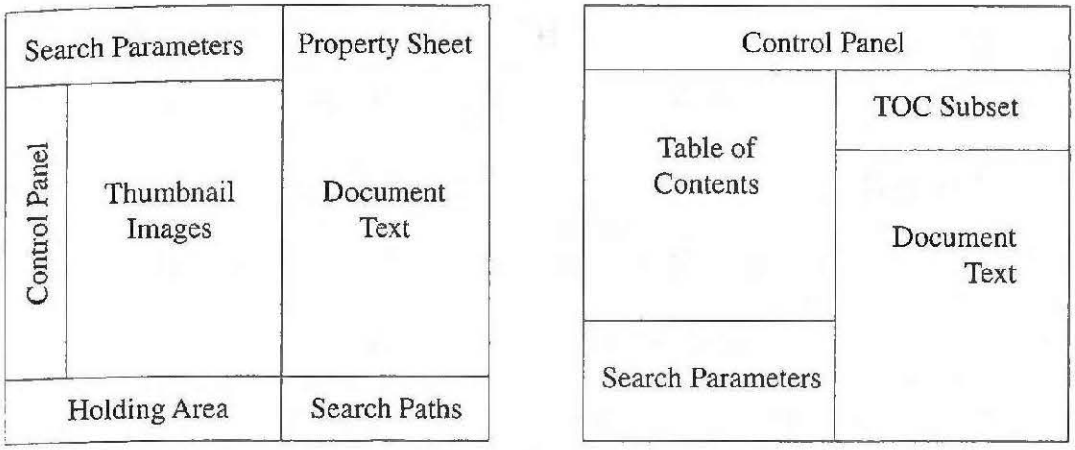


Figure 10.28 Diagrams of monolithic layouts for information access interfaces.

and is divided into a left-hand and right-hand side (see Figure 10.28). The left-hand side is further subdivided into an area at the top that contains structured entry forms for specifying the properties of a query, a column of iconic controls lining the left side, and an area for retaining documents of interest along the bottom. The main central area is used for the viewing of retrieval results, either as thumbnail representations of the original documents, or derived organizations of the documents, such as Scatter/Gather-style cluster results. Users can select documents from this area and store them in the holding area below or view them in the right-hand side. Most of the right-hand side of the display is used for viewing selected documents, with the upper portion showing metadata associated with the selected document. The area below the document display is intended to show a graphical history of earlier interactions.

Designers must make decisions about which kinds of information to show in the primary view(s). If InfoGrid were used on a smaller display, either the document viewing area or the retrieval results viewing area would probably have to be shown via a pop-up overlapping window; otherwise the user would have to toggle between the two views. If the system were to suggest terms for relevance feedback, one of the existing views would have to be supplanted with this information or a pop-up window would have to be used to display the candidate terms. The system does not provide detailed information for source selection, although this could be achieved in a very simple way with a pop-up menu of choices from the control panel.

The SuperBook Layout

The layout of the InfoGrid is quite similar to that of SuperBook (see section 10.6). The main difference is that SuperBook retains the table of contents-like display in the main left-hand pane, along with indicators of how many documents containing search hits occur in each level of the outline. Like InfoGrid, the main pane of the right-hand side is used to display selected documents. Query

formulation is done just below the table of contents view (although in earlier versions this appeared in a separate window). Terms related to the user's query are shown in this window as well. Large images appear in pop-up overlapping windows.

The SuperBook layout is the result of several cycles of iterative design [481]. Earlier versions used overlapping windows instead of a monolithic layout, allowing users to sweep out a rectangular area on the screen in order to create a new text box. This new text box had its own set of buttons that allowed users to jump to occurrences of highlighted words in other documents or to the table of contents. SuperBook was redesigned after noting results of experimental studies [350, 532] showing that users can be more efficient if given fewer, well chosen interaction paths, rather than allowing wide latitude (A recent study of auditory interfaces found that although users were more efficient with a more flexible interface, they nevertheless preferred the more rigid, predictable interface [801]). The designers also took careful note of log files of user interactions. Before the redesign, users had to choose to view the overall frequency of a hit, move the mouse to the table of contents window, click the button and wait for the results to be updated. Since this pattern was observed to occur quite frequently, in the next version of the interface, the system was redesigned to automatically perform this sequence of actions immediately after a search was run.

The SuperBook designers also attempted a redesign to allow the interface to fit into smaller displays. The redesign made use of small, overlapping windows. Some of the interaction sequences that were found useful in this more constrained environment were integrated into later designs for large monolithic displays.

The DLITE Interface

The DLITE system [193, 192] makes a number of interesting design choices. It splits functionality into two parts: control of the search process and display of results . The control portion is a graphical direct manipulation display with animation (see Figure10.29). Queries, sources, documents, and groups of retrieved documents are represented as graphical objects. The user creates a query by filling out the editable fields within a query constructor object. The system manufactures a query object, which is represented by a small icon which can be dragged and dropped onto iconic representations of collections or search services. If a service is active, it responds by creating an empty results set object and attaching the query to this. A set of retrieval results is represented as a circular pool, and documents within the result set are represented as icons distributed along the perimeter of the pool. Documents can be dragged out of the results set pool and dropped into other services, such as a document summarizer or a language translator. Meanwhile, the user can make a copy of the query icon and drop it onto another search service. Placing the mouse over the iconic representation of the query causes a 'tool-tips' window to pop up to show the contents of the underlying query. Queries can be stored and reused at a later time, thus facilitating retention of previously successful search strategies.

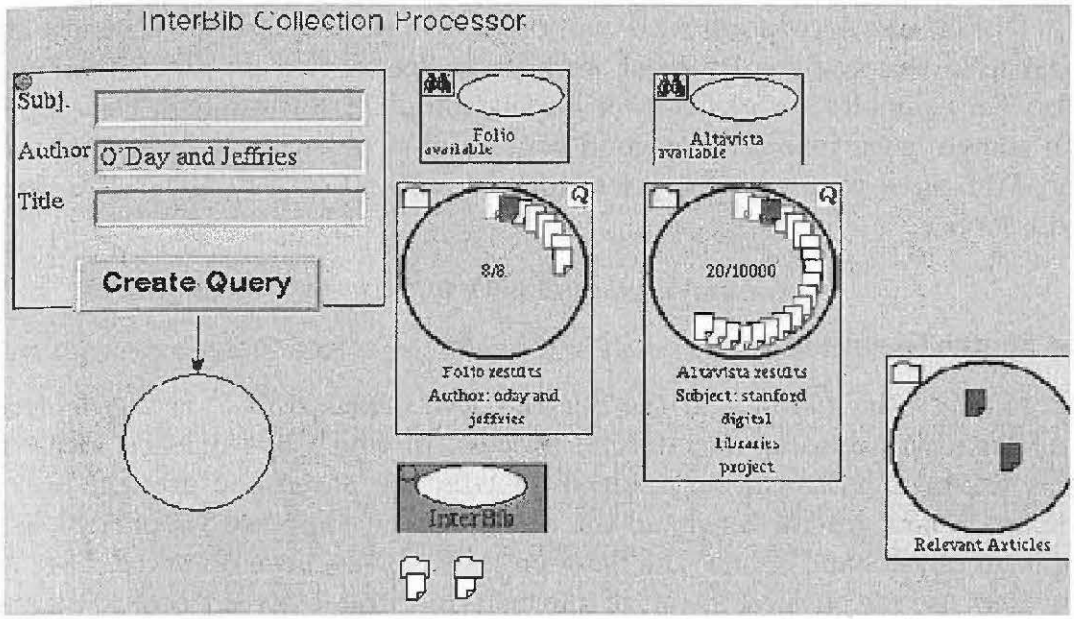


Figure 10.29 The DLITE interface [193].

A flexible interface architecture frees the user from the restriction of a rigid order of commands. On the other hand, as seen in the SuperBook discussion, such an architecture must provide guidelines, to help get the user started, give hints about valid ways to proceed, and prevent the user from making errors. The graphical portion of the DLITE interface makes liberal use of animation to help guide the user. For example, if the user attempts to drop a query in the document summarizer icon — an illegal operation — rather than failing and giving the user an accusatory error message [185], the system takes control of the object being dropped, refusing to let it be placed on the representation for the target application, and moves the object left, right, and left again, mimicking a 'shake-the-head-no' gesture. Animation is also used to help the user understand the state of the system, for example, in showing the progress of the retrieval of search results by moving the result set object away from the service from which it was invoked.

DLITE uses a separate Web browser window for the display of detailed information about the retrieved documents, such as their bibliographic citations and their full text. The browser window is also used to show Scatter/Gather-style cluster results and to allow users to select documents for relevance feedback. Earlier designs of the system attempted to incorporate text display into the direct manipulation portion, but this was found to be infeasible because of the space required [192]. Thus, DLITE separates the control portion of the information access process from the scanning and reading portion. This separation allows for reusable query construction and service selection, while at the same time allowing for a legible view of documents and relationships among retrieved documents. The selection in the display view is linked to the graphical control portion, so a document viewed in the display could be used as part of a query in a query constructor.

DLITE also incorporates the notion of a workspace, or 'workcenter,' as it is known in this system. Different workspaces are created for different kinds of tasks. For example, a workspace for buying computer software can be equipped with source icons representing good sources of reviews of computer software, good Web sites to search for price information and link to the user's online credit service.

The SketchTrieve Interface

The guiding principle behind the SketchTrieve interface [365] is the depiction of information access as an informal process, in which half-finished ideas and partly explored paths can be retained for later use, saved and brought back to compare to later interactions, and the results can be combined via operations on graphical objects and connectors between them. It has been observed [584, 722] that users use the physical layout of information within a spreadsheet to organize information. This idea motivates the design of SketchTrieve, which allows users to arrange retrieval results in a side-by-side manner to facilitate comparison and recombination (see Figure 10.30).

The notion of a canvas or workspace for the retention of the previous context should be adopted more widely in future. Many issues are not easily solved, such as how to show the results of a set of interrelated queries, with minor modifications based on query expansion, relevance feedback, and other forms of modification. One idea is to show sets of related retrieval results as a stack of

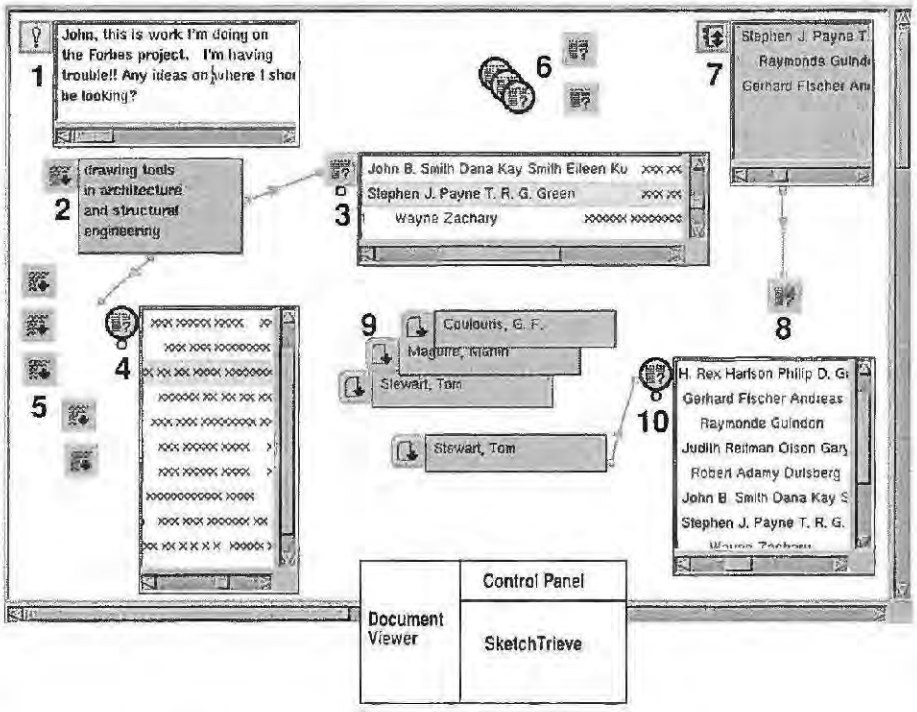


Figure 10.30 The SketchTrieve interface [365].

cards within a folder and allow the user to extract subsets of the cards and view them side by side, as is done in SketchTrieve, or compare them via a difference operation.

10.8.4 Examples of Poor Use of Overlapping Windows

Sometimes conversion from a command-line-based interface to a graphical display can cause problems. Hancock-Beaulieu *et al.* [337] describe poor design decisions made in an overlapping windows display for a bibliographic system. (An improvement was found with a later redesign of the system that used a monolithic interface [336].) Problems can also occur when designers make a 'literal' transformation from a TTY interface to a graphical interface. The consequences can be seen in the current LEXIS-NEXIS interface, which does not make use of the fact that window systems allow the user to view different kinds of information simultaneously. Instead, despite the fact that it occupies the entire screen, the interface does not retain window context when the user switches from one function to another. For example, viewing a small amount of metadata about a list of retrieved titles causes the list of results to disappear, rather than overlaying the information with a pop-up window or rearranging the available space with resizable tiles. Furthermore, this metadata is rendered in poorly-formatted ASCII instead of using the bit-map capabilities of a graphical interface. When a user opts to see the full text view of a document, it is shown in a small space, a few paragraphs at a time, instead of expanding to fill the entire available space.

10.8.5 Retaining Search History

Section 10.3 discusses information seeking strategies and behaviors that have been observed by researchers in the field. This discussion suggests that the user interface should show what the available choices are at any given point, as well as what moves have been made in the past, short-term tactics as well as longer-term strategies, and allow the user to annotate the choices made and information found along the way. Users should be able to bundle search sessions as well as save individual portions of a given search session, and flexibly access and modify each. There is also increasing interest in incorporating personal preference and usage information both into formulation of queries and use of the results of search [277].

For the most part these strategies are not supported well in current user interfaces; however some mechanisms have been introduced that begin to address these needs. In particular, mechanisms to retain prior history of the search are useful for these tasks. Some kind of history mechanism has been made available in most search systems in the past. Usually these consist of a list of the commands executed earlier. More recently, graphical history has been introduced, that allows tracking of commands and results as well. Kim and Hirtle

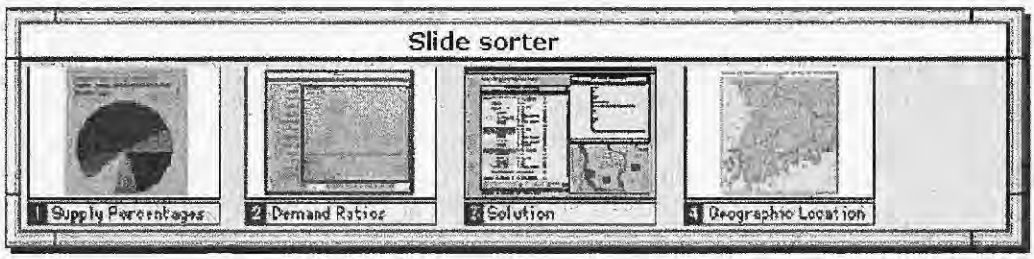


Figure 10.31 The VISAGE interaction history visualization [685].

[440] present a summary of graphical history presentation mechanisms. Recently, a graphical interface that displays Web page access history in a hierarchical structure was found to require fewer page accesses and require less time when returning to pages already visited [370].

An innovation of particular interest for information access interfaces is exemplified by the saving of state in miniature form in a 'slide sorter' view as exercised by the VISAGE system for information visualization [685] (see Figure 10.31). The VISAGE application has the added advantage of being visual in nature and so individual states are easier to recognize. Although intended to be used as a presentation creation facility, this interface should also be useful for retaining search action history.

10.8.6 Integrating Scanning, Selection, and Querying

User interfaces for information access in general do not do a good job of supporting strategies, or even of sequences of movements from one operation to the next. Even something as simple as taking the output of retrieval results from one query and using them as input to another query executed in a later search session is not well supported in most interfaces.

Hertzum and Frokjaer [368] found that users preferred an integration of scanning and query specification in their user interfaces. They did not, however, observe better results with such interactions. They hypothesized that if interactions are too unrestricted this can lead to erroneous or wasteful behavior, and interaction between two different modes requires more guidance. This suggests that more flexibility is needed, but within constraints (this argument was also made in the discussion of the SuperBook system in section 10.6).

There are exceptions. The new Web version of the Melyvl system provides ways to take the output of one query and modify it later for re-execution (see Figure 10.32). The workspace-based systems such as DLITE and Rooms allow storage and reuse of previous state. However, these systems do not integrate the general search process well with scanning and selection of information from auxiliary structures. Scanning, selection, and querying needs to be better integrated in general. This discussion will conclude with an example of an interface that does attempt to tightly couple querying and browsing.

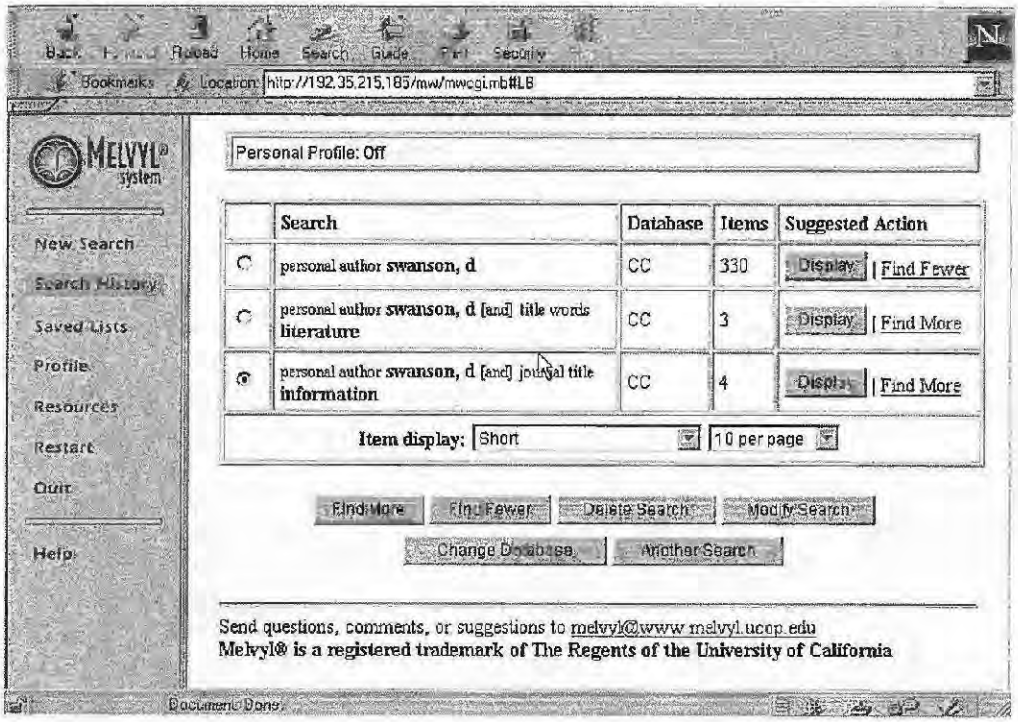


Figure 10.32 A view of query history revision in the Web-based version of the Melvyl bibliographic catalog. Copyright ©, The Regents of the University of California.

The Cat-a-Cone interface integrates querying and browsing of very large category hierarchies with their associated text collections. The prototype system uses 3D+animation interface components from the Information Visualizer [144], applied in a novel way, to support browsing and search of text collections and their category hierarchies. See Figure 10.33. A key component of the interface is the separation of the graphical representation of the category hierarchy from the graphical representation of the documents. This separation allows for a fluid, flexible interaction between browsing and search, and between categories and documents. It also provides a mechanism by which a *set* of categories associated with a document can be viewed along with their hierarchical context.

Another key component of the design is assignment of first-class status to the representation of text content. The retrieved documents are stored in a 3D+animation book representation [144] that allows for compact display of moderate numbers of documents. Associated with each retrieved document is a page of links to the category hierarchy and a page of text showing the document contents. The user can 'ruffle' the pages of the book of retrieval results and see corresponding changes in the category hierarchy, which is also represented in 3D+animation. All and only those parts of the category space that reflect the semantics of the retrieved document are shown with the document.

The system allows for several different kinds of starting points. Users can start by typing in a name of a category and seeing which parts of the category hierarchy match it. For example, Figure 10.34 shows the results of searching on

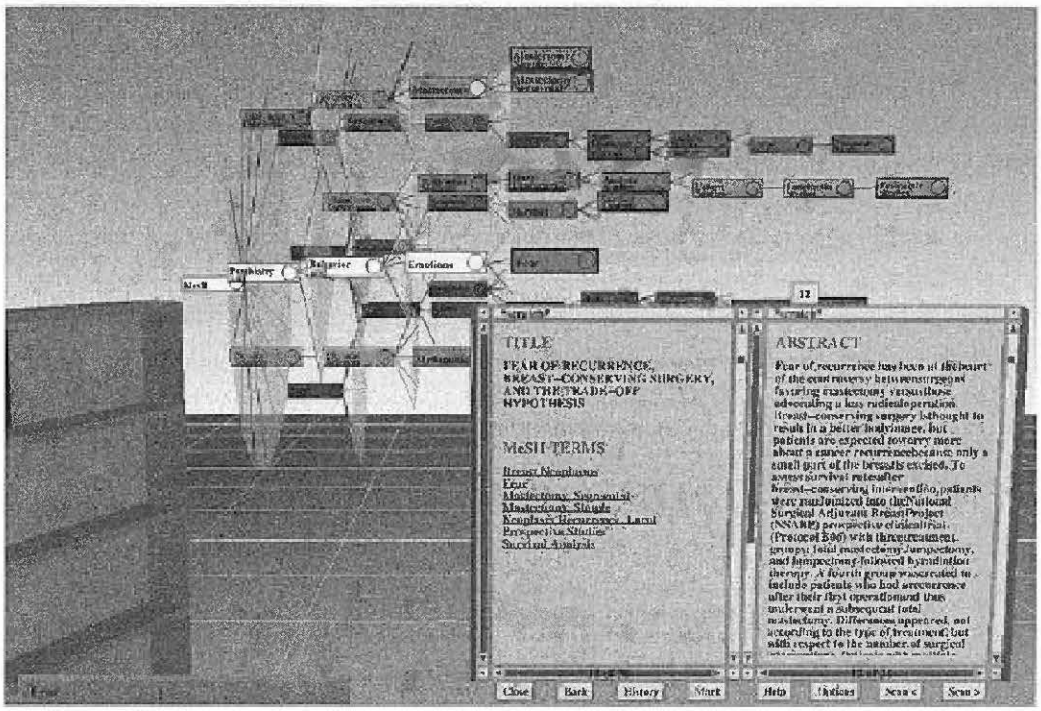


Figure 10.33 The Cat-a-Cone interface for integrating category and text scanning and search [358].

'Radiation' over the MeSH terms in this subcollection. The word appears under four main headings (*Physical Sciences*, *Diseases*, *Diagnostics*, and *Biological Sciences*). The hierarchy immediately shows why 'Radiation' appears under *Diseases* — as part of a subtree on occupational hazards. Now the user can select one or more of these category labels as input to a query specification.

Another way the user can start is by simply typing in a free text query into an entry label. This query is matched against the collection. Relevant documents are retrieved and placed in the book format. When the user 'opens' the book to a retrieved document, the parts of the category hierarchy that correspond to the retrieved documents are shown in the hierarchical representation. Thus, multiple intersecting categories can be shown simultaneously, in their hierarchical context. Thus, this interface fluidly combines large, complex metadata, starting points, scanning, and querying into one interface. The interface allows for a kind of relevance feedback, by suggesting additional categories that are related to the documents that have been retrieved. This interaction model is similar to that proposed by [5].

Recall the evaluation of the Kohonen feature map representation discussed in section 10.4. The experimenters found that some users expressed a desire for a visible hierarchical organization, others wanted an ability to zoom in on a subarea to get more detail, and some users disliked having to look through the entire map to find a theme, desiring an alphabetical ordering instead. The subjects liked the ease of being able to jump from one area to another without

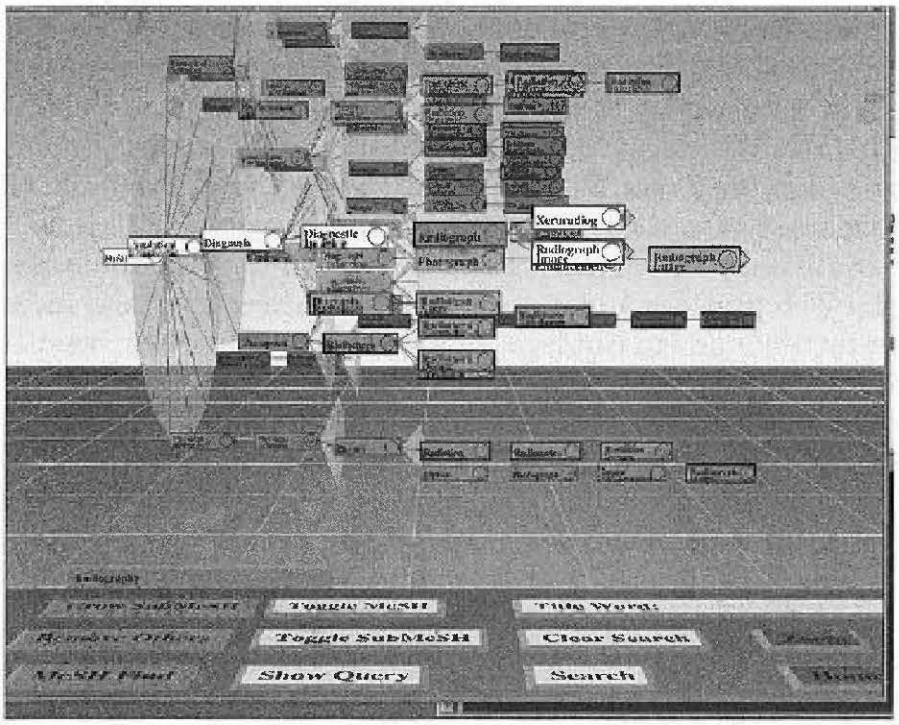


Figure 10.34 An interface for a starting point for searching over category labels [358].

having to back up (as is required in Yahoo!) and liked the fact that the maps have varying levels of granularity.

These results all support the design decisions made in the Cat-a-Cone. Hierarchical representation of term meanings is supported, so users can choose which level of description is meaningful to them. Furthermore, different levels of description can be viewed simultaneously, so more familiar concepts can be viewed in more detail, and less familiar at a more general level. An alphabetical ordering of the categories coupled with a regular expression search mechanism allows for straightforward location of category labels. Retrieved documents are represented as first-class objects, so full text is visible, but in a compact form. Category labels are disambiguated by their ancestor/descendant/sibling representation. Users can jump easily from one category to another and can in addition query on multiple categories simultaneously (something that is not a natural feature of the maps). The Cat-a-Cone has several additional advantages as well, such as allowing a document to be placed at the intersection of several categories, and explicitly linking document contents with the category representation.

10.9 Trends and Research Issues

The importance of human computer interaction is receiving increasing recognition within the field of computer science [587]. As should be evident from the

contents of this chapter, the role of the user interface in the information access process has only recently begun to receive the attention it deserves. Research in this area can be expected to increase rapidly, primarily because of the rise of the Web. The Web has suddenly made vast quantities of information available globally, leading to an increase in interest in the problem of information access. This has lead to the creation of new information access paradigms, such as the innovative use of relevance feedback seen in the Amazon.com interface. Because the Web provides a platform-independent user interface, investment in better user interface design can have an impact on a larger user population than before.

Another trend that can be anticipated is an amplified interest in organization and search over personal information collections. Many researchers are proposing that in future a person's entire life will be recorded using various media, from birth to death. One motivation for this scenario is to enable searching over everything a person has ever read or written. Another motivation is to allow for searching using contextual clues, such as 'find the article I was reading in the meeting I had on May 1st with Pam and Hal'. If this idea is pursued, it will require new, more sophisticated interfaces for searching and organizing a huge collection of personal information.

There is also increasing interest in leveraging the behavior of individuals and groups, both for rating and assessing the quality of information items, and for suggesting starting points for search within information spaces. Recommender systems can be expected to increase in prevalence and diversity. User interfaces will be needed to guide users to appropriate recommended items based on their information needs.

The field of information visualization needs some new ideas about how to display large, abstract information spaces intuitively. Until this happens, the role of visualization in information access will probably be primarily confined to providing thematic overviews of topic collections and displaying large category hierarchies dynamically. Breakthroughs in information visualization can be expected to have a strong impact on information access systems.

10.10 Bibliographic Discussion

The field of human-computer interaction is a broad one, and this chapter touches on only a small subset of pertinent issues. For further information, see the excellent texts on user interface design by Shneiderman [725], information seeking behavior by Marchionini [542], and digital libraries by Lesk [501]. An excellent book on visual design is that of Mullet and Sano [580]. Tufte has written thought-provoking and visually engaging books on the power of information visualization [769, 770] and a collection of papers on information visualization has been edited by Card *et al.* [141].

This chapter has discussed many ideas for improving the human-computer interaction experience for information seekers. This is the most rapidly

developing area of information access today, and improvements in the interface are likely to lead the way toward better search results and better-enabled information creators and users. Research in the area of human-computer interaction is difficult because the field is relatively new, and because it can be difficult to obtain strong results when running user studies. These challenges should simply encourage those who really want to influence the information access systems of tomorrow.

Acknowledgements

The author gratefully acknowledges the generous and helpful comments on the contents of this chapter by Gary Marchionini and Ben Shneiderman, the excellent administrative assistance of Barbara Goto, and the great faith and patience of Ricardo Baeza-Yates and Berthier Ribeiro-Neto.

Chapter 11
Multimedia IR: Models and Languages

by Elisa Bertino, Barbara Catania, and Elena Ferrari

11.1 Introduction

The need for an integrated management for multimedia data is rapidly growing in several application environments such as offices, CAD/CAM applications, and medical applications. For this reason, multimedia information systems are widely recognized to be one of the most promising fields in the area of information management.

The most important characteristic of a multimedia information system is the variety of data it must be able to support. Multimedia systems must have the capability to store, retrieve, transport, and present data with very heterogeneous characteristics such as text, images (both still and moving), graphs, and sound. For this reason, the development of a multimedia system is considerably more complex than a traditional information system. Conventional systems only deal with simple data types, such as strings or integers. On the contrary, the underlying data model, the query language, and the access and storage mechanisms of a multimedia system must be able to support objects with a very complex structure. The need then arises for developing *Multimedia Information Retrieval* (Multimedia IR for short) systems specifically for handling multimedia data. Traditional IR systems (see Chapter 2) only deal with textual, unstructured data; therefore, they are unable to support the mix of structured and unstructured data, and different kinds of media, typical of a Multimedia IR system. For instance, a traditional IR system does not support metadata information such as that provided by database schema, which is a fundamental component in a database management system (DBMS). On the other hand, Multimedia IR systems require some form of database schema because several multimedia applications need to structure their data at least partially. However, the notion of schema may need to be weakened with respect to the traditional notion to ensure a higher degree of flexibility in structuring data. Moreover,

a Multimedia IR system requires handling metadata which is crucial for data retrieval, whereas traditional IR systems do not have such requirement.

The architecture of a Multimedia IR system depends on two main factors: first, the peculiar characteristics of multimedia data, and second, the kinds of operations to be performed on such data. In what follows, we briefly deal with both these aspects.

Data Modeling

A Multimedia IR system should be able to represent and store multimedia objects in a way that ensures their fast retrieval. The system should be therefore able to deal with different kinds of media and with *semi-structured data*, i.e., data whose structure may not match, or only partially match, the structure prescribed by the data schema. In order to represent semi-structured data, the system must typically extract some features from the multimedia objects. A related issue is how these features are extracted and efficiently maintained by the system.

Data Retrieval

The main goal of a Multimedia IR system is to efficiently perform *retrieval*, based on user requests, exploiting not only data attributes, as in traditional DBMSs, but also the *content* of multimedia objects. This poses several interesting challenges, due to the heterogeneity of data, the fuzziness of information, the loss of information in the creation of indexes, and the need of an interactive refinement of the query result. Data retrieval relies on the following basic steps:

(1) **Query specification**. In this step, the user specifies the request. The query interface should allow the user to express fuzzy predicates for proximity searches (for example, 'Find all images similar to a car'), content-based predicates (for example, 'Find multimedia objects containing an apple'), conventional predicates on the object attributes (for example, conditions on the attribute 'color' of an image, such as 'Find all red images'), and structural predicates (for example, 'Find all multimedia objects containing a video clip').

(2) **Query processing and optimization**. Similarly to traditional systems, the query is parsed and compiled into an internal form. In generating this internal representation, the query is also optimized, choosing the best evaluation plan. Note that, due to the presence of fuzzy terms, content-based predicates, and structural predicates, query processing is a very complex activity. A great amount of work has been done on query processing both in traditional [402] and spatial databases [247, 82, 118, 361, 623]. However, little work has been done on query processing strategies for multimedia databases. The main problem is the heterogeneity of data: different query processing strategies, one for each data type, should be combined together in some way.

(3) **Query answer**. The retrieved objects are returned to the user in decreasing order of relevance. Relevance is measured as a distance function from the query object to the stored ones.

(4) **Query iteration**. In traditional DBMSs, the query process ends when the system returns the answer to the user. In a Multimedia IR system, due to the inevitable lack of precision in the user request, the query execution is iterated until the user is satisfied. At each iteration the user supplies the system with additional information by which the request is refined, reducing or increasing the number of returned answers.

From the previous discussion it follows that a Multimedia IR system differs from a traditional IR system in two main aspects. First, the structure of multimedia objects is more complex than the structure of typical textual data, handled by traditional IR systems. This complexity requires the integration of traditional IR technology with the technology of multimedia database management systems to adequately represent, manage, and store multimedia objects. Note that the use of a DBMS also provides update functionalities and transaction management which are in general not covered by typical IR systems. Second, object retrieval is mainly based on a similarity approach. Moreover, the objects retrieved by a query are usually returned to the user in a ranked form. These aspects are successfully handled by IR techniques (see Chapter 2). However, IR systems have initially been developed to support libraries of articles, journals, and encyclopedic knowledge bases (see Chapter 2). In those systems, the fundamental unit is the *textual document*. Thus, the techniques developed for traditional IR systems should be extended to deal with documents containing other media.

Multimedia IR systems should therefore combine both the DBMS and the IR technology, to integrate the data modeling capabilities of DBMSs with the advanced and similarity-based query capabilities of IR systems. The resulting system will be able to answer attribute-based queries as well as content-based queries. The whole architecture of the resulting system, in particular the query optimizer, must take this aspect into account in order to efficiently support user requests.

In this chapter, we discuss modeling and query language issues for multimedia objects, pointing out the differences and the analogies between a traditional IR system and a multimedia one. Problems related to feature extraction and searching are covered by Chapter 12.

The first part of the chapter is devoted to the presentation of the most relevant models proposed in the literature for multimedia data, with particular attention to commercial proposals.

The second part of the chapter investigates the peculiarities of multimedia query languages with respect to traditional ones. Then, as an example, two different language proposals are presented. Also in this case, we focus on commercial proposals and we discuss how the new standard SQL3 could be used to deal with multimedia data retrieval.

11.2 Data Modeling

As we have already remarked, the complex nature of multimedia data may benefit from the use of DBMS functions for data representation and querying. However, the integration of multimedia data in a traditional DBMS is not an easy task. Indeed, traditional DBMSs are mainly targeted to support conventional data. Multimedia data is inherently different from conventional data. The main difference is that information about the content of multimedia data are usually not encoded into attributes provided by the data schema (*structured data*). Rather, text, image, video, and audio data are typically *unstructured*. Therefore, specific methods to identify and represent content features and semantic structures of multimedia data are needed. Another distinguishing feature of multimedia data is its large storage requirements. One single image usually requires several Kbytes of storage, whereas a single second of video can require several Mbytes of storage. Moreover, the content of multimedia data is difficult to analyze and compare, in order to be actively used during query processing.

Addressing data modeling issues in the framework of Multimedia IR systems entails two main tasks. First, a data model should be defined by which the user can specify the data to be stored into the system. Such a data model should have the ability of an integrated support for both conventional and multimedia data types and should provide methods to analyze, retrieve, and query such data. Second, the system should provide a model for the internal representation of multimedia data. The definition of such a model is crucial for the efficiency of query processing.

As far as the first aspect is concerned, a promising technology with respect to the modeling requirements of multimedia data is the object-oriented one [89]. The richness of the data model provided by OODBMSs makes them more suitable than relational DBMSs for modeling both multimedia data types and their semantic relationships. Moreover, the concept of class can be naturally used to define ad hoc data types for multimedia data in that a class is characterized by both a set of attributes and a set of operations that can be performed on these attributes. Classes can, moreover, be related into inheritance hierarchies, thus allowing the definition of a multimedia class as a specialization of one or more superclasses. However, the performance of OODBMs in terms of storage techniques, query processing, and transaction management is not comparable to that of relational DBMSs. Another drawback of OODBMs is that they are highly non-standard. Indeed, even though a standard language has been defined by the Object Database Management Group (ODMG), very few systems support it.

For all the above reasons, a lot of effort has been devoted to the extension of the relational model with capabilities for modeling complex objects, typical of the object-oriented context. The goal of the so-called *object-relational* technology is to extend the relational model with the ability of representing complex data types by maintaining, at the same time, the performance and the simplicity of relational DBMSs and related query languages. The possibility of defining abstract data types inside the relational model allows one to define ad hoc data types for multimedia data. For instance, such data types can provide support for

content-dependent queries. In the following section, we will give some examples of such extensions.

The second problem related to data modeling is how multimedia data are represented inside the system. Due to the particular nature of multimedia data, it is not sufficient to describe it through a set of attributes as usually done with traditional data. Rather, some information should be extracted from the objects and used during query processing. The extracted information is typically represented as a set of *features*; each multimedia object is therefore internally represented as a list of features, each of which represents a point in a multi-dimensional space. Multi-attribute access methods can then be used to index and search for them (see Chapter 12). Features can be assigned to multimedia objects either manually by the user, or automatically by the system. In general, a hybrid approach is used, by which the system determines some of the values and the user corrects or augments them. In both cases, values assigned to some specific features, such as the shape of an image or the style of an audio object, are assigned to the object by comparing the object with some previously classified objects. For instance, to establish whether an image represents a car or a house, the shape of the image is compared with the shapes of already classified cars and houses before taking a decision. Finally, it is important to recall that feature extraction cannot be precise. Therefore, a weight is usually assigned to each feature value representing the uncertainty of assigning such a value to that feature. For example, if we are 80% sure that a shape is a square, we can store this value together with the recognized shape.

From the previous discussion, it follows that data modeling in a Multimedia IR system is an articulated activity that must take into account both the complex structure of data and the need of representing features extracted from multimedia objects.

In the following, we give a brief overview of some proposals to model multimedia data. We start by reviewing the support for multimedia data provided by commercial DBMSs. Then, as an example of a research proposal, we survey the data model developed in the context of the MULTOS project.

11.2.1 Multimedia Data Support in Commercial DBMSs

Most current relational DBMSs support variable-length data types which can be used to represent multimedia data. The way these data are supported by commercial DBMSs is mostly non-standard in that each DBMS vendor uses different names for such data types and provides support for different operations on them.

For example, the Oracle DBMS provides the VARCHAR2 data type to represent variable length character strings. The maximum length of VARCHAR2 data is 4000 bytes. The RAW and LONG RAW data types are used for data that is not to be interpreted by Oracle. These data types can be used to store graphics, sounds, or unstructured objects. LOB data types can be used to store Large unstructured data OBjects up to four gigabytes in size. BLOBs are used to store unstructured Binary Large OBjects, whereas CLOBs are used to store Character Large OBject data.

The Sybase SQL server supports IMAGE and TEXT data types to store images and unstructured text, respectively, and provides a limited set of functions for their searching and manipulation.

However, the support provided by the above mentioned data types is very limited in that the DBMS does not provide any interpretation of the data content. Moreover, operations that can be performed on such data by means of the built-in functions provided by the DBMS are very simple.

As we have already remarked, most commercial relational DBMSs vendors are investing a lot of effort in extending the relational model with the capability of modeling complex objects, typical of the object-oriented context. Such efforts have given rise to the upcoming SQL3 standard. From a data modeling point of view, the major improvement provided by SQL3 with respect to its predecessor SQL-92, is the support for an *extensible type system*. Extensibility of the type system is achieved by providing constructs to define user-dependent abstract data types, in an object-oriented like manner. In SQL3, each type specification consists of both attribute and function specifications. A strong form of encapsulation is provided, in that attribute values can only be accessed by using some system functions. Moreover, user-defined functions can be either visible from any object or only visible in the object they refer to. Both single and multiple inheritance can be defined among user-defined types and dynamic late binding is provided [89].

SQL3 also provides three types of collection data types: sets, multisets, and lists. The elements of a collection must have compatible types. Several system-defined operations are provided to deal with collections. Besides the definition of user-dependent abstract data types, SQL3 provides a restricted form of object identifier that supports sharing and avoids data duplication.

Although SQL3 has not yet been officially published, most commercial products have already implemented their proprietary versions of SQL3. An example in such direction is the *data cartridges* provided by Oracle for multimedia data handling, or the *data blades* supported by Illustra.†

Oracle provides data cartridges for text, spatial data, image, audio and video data. To give a concrete example, Oracle8 provides a ConText cartridge, which is a text management solution combining data management capabilities of a traditional DBMS with advanced text retrieval and natural-language process technology. The ConText cartridge supports the most popular document formats, including ASCII, MS Word, and HTML. One of the most relevant feature of the ConText cartridge is its ability to find documents about a specific topic (thus providing a form of content-based retrieval). Content-based queries on text documents can be combined with traditional queries in the same SQL statement and can be efficiently executed due to the use of indexing techniques specific for texts. Such techniques are based on the notion of *inverted files* (see Chapter 8) which map a given word to the documents containing it, thus allowing a fast retrieval of all the documents containing a particular word.

† Illustra was acquired by Informix in 1995.

Illustra provides 3D and 2D spatial data blades for modeling spatial data. The supported data types include boxes, vectors, quadrangles, etc., and examples of supported operations are INTERSECT, CONTAINS, OVERLAPS, CENTER, and so on. Spatial data blades also implement R-trees for performing efficient spatial queries [330, 717]. The text data blade provides data types for representing unstructured text and performing content-based queries. For example, the method *ContainWords* can be used to search for all the documents containing a particular word. Moreover, Illustra supports a data blade which can be used to query images by content.

The object-relational technology and its extensive type system is now starting to be widely used both in industrial and research projects. An example of this trend is the La Scala archive project, currently under development at the Laboratorio di Informatica Musicale of the University of Milano [254]. The goal of this project is the development of the multimedia archive of Teatro alla Scala, one of the best known musical temples of the world, using the Oracle technology and the related data cartridges. The system is organized around La Scala *nights*. Each night encompasses the phonic items, score, and other graphical and video items related to the performance. When a new performance has to be prepared, the musicians can easily access all the materials (such as CD-ROMs, video, photos, and scores) of previous editions of the same performance. Accessing such information has required the development of ad hoc cartridges to represent and query non-conventional data. For instance, we are currently developing a data cartridge that allows content-based queries on music scores. We apply pattern matching techniques to music scores to enable the user to sing a few bars into a microphone linked to the computer and see all the music scores containing a piece of music close to the one being sung. Users can then view the retrieved musical graphic scores, or excerpts from them, and simultaneously play the corresponding music.

As an example of a data model suitable for a multimedia environment, in the following we consider the data model developed in the context of the MULTOS project [759].

11.2.2 The MULTOS Data Model

MULTOS (MULTimedia Office Server) is a multimedia document‡ server with advanced document retrieval capabilities, developed in the context of an ESPRIT project in the area of Office Systems [759].

MULTOS is based on a *client/server* architecture. Three different types of document servers are supported: *current servers*, *dynamic servers*, and *archive servers*, which differ in storage capacity and document retrieval speed. Such servers support filing and retrieval of multimedia objects based on document collections, document types, document attributes, document text, and images.

‡ As MULTOS deals with office services, in the following we use the words *object* and *document* as synonymous.

The MULTOS data model allows the representation of high level concepts present in the documents contained in the database, the grouping of documents into classes of documents having similar content and structure, and the expression of conditions on free text.

Each document is described by a logical structure, a layout structure, and a conceptual structure. The *logical* structure determines arrangements of logical document components (e.g., title, introduction, chapter, section, etc.). The *layout* structure deals with the layout of the document content and it contains components such as pages, frames, etc. The *conceptual* structure allows a semantic-oriented description of the document content as opposed to the syntax-oriented description provided by the logical and layout structure. The conceptual structure has been added to provide support for document retrieval by content. MULTOS provides a formal model, based on a data structuring tool available in semantic data models, to define the document conceptual structure. The logical and layout structures are defined according to the ODA document representation [398].

Documents having similar conceptual structures are grouped into *conceptual types*. In order to handle types in an effective manner, conceptual types are maintained in a hierarchy of generalization, where a subtype inherits from its supertypes the conceptual structure and can then refine it. Types can be *strong* or *weak*. A strong type completely specifies the structure of its instances. A weak type, on the other hand, partially specifies the structure of its instances. Moreover, components of unspecified type (called *spring component types*) can appear in a document definition.

Example 1 *The conceptual structure of the type* `Generic_Letter` *is shown in Figure 11.1. The node* `Letter_Body` *is a spring conceptual component. The complete conceptual structure in Figure 11.2 corresponds to the type* `Business_Product_Letter`. *This type has been obtained from* `Generic_Letter` *by specialization of* `Letter_Body` *into a complex conceptual component, defined as an aggregation of five conceptual components. According to the conceptual model, the document type* `Business_Product_Letter` *is linked to the document type* `Generic_Letter` *by an 'is-a' relationship. In this example, the '+' symbol attached to the* `Receiver` *component means that it is multivalued. Notice also that the* `Name` *and the* `Address` *appear in two subtrees having as roots the conceptual components* `Receiver` *and* `Sender`, *respectively.*

For document retrieval, conceptual types play the role of the database schema which enables the use of efficient access structures. Moreover, conceptual types are the basis for formulating queries at an abstract level.

MULTOS also provides a sophisticated approach to deal with image data. First, an image analysis process is performed, consisting of two phases: *low level image analysis* and *high level image analysis*. During the low level image analysis phase, the basic objects composing a given image and their relative positions are identified. The high level image analysis phase deals with image interpretation according to the Dempster-Shafter theory of evidence [60, 312].

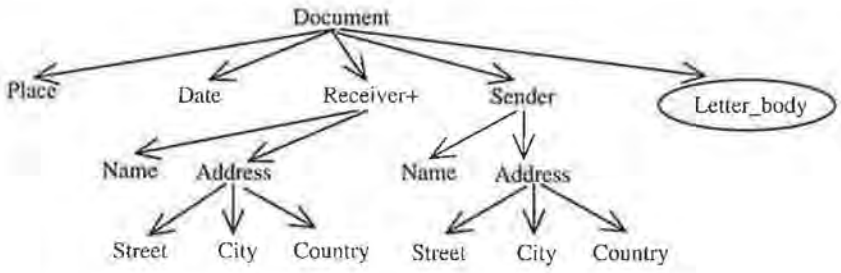


Figure 11.1 Conceptual structure of the type `Generic_Letter`.

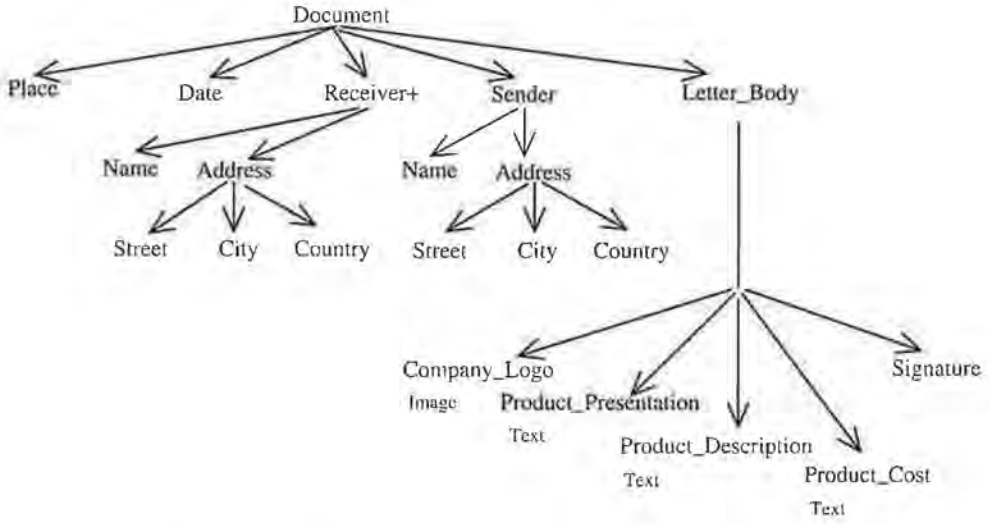


Figure 11.2 Complete conceptual structure of the type `Business_Product_Letter`.

At the end of the image analysis process, images are described in terms of the objects recognized, with associated belief and plausibility values, and the classes to which they belong. The information is then exploited in image access.

Image access information is stored in an *image header*, associated with the image file. Access structures are then built for a fast access to image headers. Two types of index are constructed:

- **Object index**. For each object a list is maintained. Each element of the lists is a pair (BI,IMH), where IMH is a pointer to the header of the image containing the object, and BI is the associated belief interval, representing the probability that the image considered really contains the object.
- **Cluster index**. For each image class, a list of pairs (MF,IMH) is maintained. IMH is a pointer to an image header corresponding to an image with a non-null degree of membership to the class, and MF is the value of the membership degree. The membership degree of an image to a given class is computed by comparing the image interpretation resulting from the analysis phase, with the class description, using techniques analogous to the ones used in text IR systems [698] (see Chapter 6).

11.3 Query Languages

Queries in relational or object-oriented database systems are based on an *exact match* mechanism, by which the system is able to return exactly those tuples or objects satisfying some well specified criteria given in the query expression and nothing more. In general, query predicates specify which values the object attributes must contain.

Because of the semi-structured nature of multimedia objects, the previous approach is no longer adequate in a Multimedia IR system. In this context, the user should still be able to query the content of multimedia objects by specifying values of semantic attributes but he/she should also be able to specify additional conditions about the content of multimedia data. Thus, the exact match is only one of the possible ways of querying multimedia objects. More often, a similarity-based approach is applied that considers both the structure and the content of the objects. Queries of the latter type are called *content-based* queries since they retrieve multimedia objects depending on their global content. Information on the global content of an object is not represented as attribute values in the database system. Rather, as we have already remarked in section 11.2, a set of information, called features, is extracted and maintained for each object. When the query is submitted, the features of the query object are matched with respect to the features of the objects stored in the database and only the objects that are *more similar* to the query one are returned to the user (see Chapter 12).

The characteristics of content-based query processing impacts the definition of a multimedia query language and, in general, of the user interface. In particular, in designing a multimedia query language, three main aspects require attention:

- How the user enters his/her request to the system, i.e., which interfaces are provided to the user for query formulation.
- Which conditions on multimedia objects can be specified in the user request. The conditions that can be expressed depend on the support the system provides for content-based retrieval (see Chapter 12).
- How uncertainty, proximity, and weights impact the design of the query language.

In the following, we discuss the above aspects in detail. Then, we present two examples of multimedia query languages. First, we illustrate how traditional relational query languages can be extended to deal with multimedia data, discussing the main characteristics of the upcoming SQL3 query language. Then, as an example of a research proposal, we introduce the query language supported by MULTOS (see section 11.2.2).

11.3.1 Request Specification

Two different interfaces can be presented to the user for querying multimedia objects. The first type of interface is based on *browsing and navigation*. Usually, due to the complex structure of multimedia objects, it may be useful to let users browse and navigate inside the structure of multimedia objects to locate the desired objects. Such an approach is typically used in CAD/CAM/CASE environments due to the complex structure of the objects under consideration.

Navigation, however, is not always the best way to find multimedia objects, in that it may be heavily time consuming when the object desired is deeply nested. The second approach for selecting objects is therefore based, as traditionally in DBMSs, on specifying the conditions the objects of interest must satisfy, by means of queries.

Queries, in turn, can be specified in two different ways: the first, typical of a traditional database context, is to enter the query by using a specific query language. However, in some cases (especially when images and audio data are considered), a *query by example* approach is preferred. Under this approach, queries are specified by using actual data inside a visual environment; the user provides the system with an object example that is then used to retrieve all the stored objects similar to the given one. For example, the user may choose a house and pose the query: 'Retrieve all houses of similar shape and different color.' This approach requires the use of a GUI environment where the user can pick examples and compose the query object. In order to pick examples, the system must supply some *domains*, i.e., sets of typical values, one for each object feature (see section 11.2).

11.3.2 Conditions on Multimedia Data

Multimedia query languages should provide predicates for expressing conditions on the attributes, the content, and the structure of multimedia objects. In general, query predicates can be classified into three different groups:

- **Attribute predicates** concern the attributes (i.e., the structured content) of multimedia objects.
- **Structural predicates** concern the structure of the data being considered.
- **Semantic predicates** concern the semantic and unstructured content of the data involved.

By the term *attribute predicates* we mean predicates against traditional attributes, i.e., attributes for which an exact value is supplied for each object. Examples of attributes are the speaker of an audio object, the size of an object, or its type. By querying these predicates, the system applies an exact-match retrieval, using the same techniques as traditional DBMSs.

Structural predicates concern the structure of multimedia objects. Such predicates can be answered by using some form of metadata [99, 442] and

information about the database schema. With respect to traditional databases, structural queries play a fundamental role in multimedia query processing, due to the complex structure of multimedia objects. An example of use of a structural predicate is the query: 'Find all multimedia objects containing at least one image and a video clip.'

On the other hand, *semantic predicates* concern the semantic content of the queried data, depending on the features that have been extracted and stored for each multimedia object. An example of a semantic query is 'Find all the objects containing the word OFFICE.' Note that the word 'OFFICE' may appear either in a textual component of the object or as a text attribute of some image components. The query 'Find all the red houses' is a query on the image content. This query can be executed only if color and shape are features that have been previously extracted from images.

Current systems support semantic predicates only with respect to specific features, such as the color, the shape, the texture, and sometimes the motion. For example, QBIC allows the retrieval of images with similar shapes or similar textures with respect to the object example specified in the query [257]. More innovative approaches include the Name-it project, whose aim is to process a video clip and automatically associate spoken or typed names with their corresponding faces [708].

The main difference between attribute predicates and semantic predicates is that, in the latter case, an exact match cannot be applied. This means that there is no guarantee that the objects retrieved by this type of predicate are 100% correct or precise. In general, the result of a query involving semantic predicates is a set of objects, each of which has an associated degree of relevance with respect to the query. The user can subsequently select the better matches and submit the query again.

Structural and semantic predicates can also refer to spatial or temporal properties of multimedia objects. Spatial semantic predicates specify conditions about the relative positions of a set of objects in an image or a video. Examples of spatial semantic predicates are: `contain`, `intersect`, `is contained in`, `is adjacent to`. Temporal semantic predicates are mainly related to continuous media, like audio and video. They allow one to express temporal relationships among the various frames of a single audio or video. For example, the query 'Find all the objects that contain an audio component, where the hint of the discussion is first policy, and then economy' is a temporal audio query.

From the point of view of structural predicates, spatial and temporal predicates can be used to specify temporal synchronization properties and spatial layout properties for the presentation of multimedia objects [87, 88]. For instance, in the query: 'Find all the objects containing an image overlapping the associated text', a spatial structural predicate is used to impose a condition on the spatial layout of the retrieved objects. Analogously, the query: 'Find all the objects in which a jingle is played for the duration of an image display' is an example of a structural temporal query. Note, moreover, that temporal and spatial predicates can be combined to express more articulated requirements. An example is the query: 'Find all the objects in which the logo of a car company

is displayed and, when it disappears, a graphic showing the increases in the company sales is shown in the same position where the logo was.'

Due to the complex structure of multimedia objects, all the previous types of predicates can refer either to the whole object or, if the underlying data model supports complex object representation, to some subcomponents of the object. In the last case, the query language must also be able to navigate the object structure. A typical example in this direction is represented by path expressions in object-oriented systems [89].

11.3.3 Uncertainty, Proximity, and Weights in Query Expressions

As we have already remarked, the execution of a content-dependent query returns a set of relevant objects. An interesting aspect in designing a multimedia query language is how it is possible to specify the degree of relevance of the retrieved objects. In general, this can be done in three different ways:

- By using some imprecise terms and predicates, such as `normal`, `unacceptable`, `typical`. Each of those terms does not represent a precise value but a set of possible acceptable values with respect to which the attribute or the feature has to be matched.
- By specifying particular proximity predicates. In this case, the predicate does not represent a precise relationship between objects or between attributes/features and values. Rather, the relationship represented is based on the computation of a semantic *distance* between the query object and the stored ones, on the basis of the extracted features. The *Nearest object search* is an example of proximity predicate, by which the user requests all the objects which are closest or within a certain distance of a given object. Indexing support for this kind of query is discussed in Chapter 12.
- By assigning each condition or term a given *weight*, specifying the degree of precision by which a condition must be verified by an object. For example, the query 'Find all the objects containing an image representing a screen (HIGH) and a keyboard (LOW)' [657], can be used to retrieve all the objects containing an image representing a screen and a keyboard. However, the objects containing only a screen are also retrieved and returned to the user, after the ones containing both the screen and the keyboard, since the condition imposing the containment of a keyboard is weaker than the condition imposing the containment of a screen.

The use of imprecise terms and relationships, as well as the use of weights, allows the user to drive the similarity-based selection of relevant objects. The corresponding query is executed by assigning some importance and preference values to each predicate and term. Then, objects are retrieved and presented to the user as an ordered list. This ordering is given by a score associated with each object, giving a measure of the matching degree between the object and

the query. The computation of the score is based on probabilistic models, using the preference values assigned to each predicate.

11.3.4 Some Proposals

In the following we briefly survey some query languages supporting retrieval of multimedia objects. In order to describe how standard languages are evolving to support multimedia applications, we first describe the facilities provided by the upcoming standard SQL3 to support such kinds of applications. Then, we present the query language supported by the MULTOS system [90], introduced in section 11.2.2.

The SQL3 Query Language

As we have seen in section 11.2.1, the extensible type system and in general the ability to deal with complex objects make SQL3 suitable for modeling multimedia data. From the query language point of view, the major improvements of SQL3 with respect to SQL-92 can be summarized as follows:

- **Functions and stored procedures.** SQL3 allows the user to integrate external functionalities with data manipulation. This means that functions of an external library can be introduced into a database system as *external functions*. Such functions can be either implemented by using an external language, and in this case SQL3 only specifies which is the language and where the function can be found, or can be directly implemented by using SQL3 itself. In this way, impedance mismatch between two different programming languages and type systems is avoided. Of course, this approach requires an extension of SQL with imperative programming languages constructs.
- **Active database facilities.** Another important property of SQL3 is the support of active rules, by which the database is able to react to some system- or user-dependent events by executing specific actions. Active rules, or triggers, are very useful to enforce integrity constraints.

From the multimedia perspective point of view, the aspects described make SQL3 suitable for being used as an interface language for multimedia applications. In particular, the ability to deal with external functions and user-defined data types enables the language to deal with objects with a complex structure, as multimedia objects. Note that, without this characteristic, the ability to deal with BLOB would have been useless since it reduces the view of multimedia data to single large uninterpreted data values, which are not adequate for the rich semantics of multimedia data. By the use of triggers, spatial and temporal constraints can be enforced, thus preserving the database consistency. Finally, as SQL3 is a widespread standard, it allows one to model multimedia objects in the framework of a well understood technology.

Though the above facilities make SQL3 suitable for use as an interface for multimedia applications, there are also some limitations. The main drawback is related to retrieval support and, as a consequence, optimization. Indeed, no IR techniques are integrated into the SQL3 query processor. This means that the ability to perform content-based search is application dependent. As a consequence, objects are not ranked and are therefore returned to the application as a unique set. Moreover, specialized indexing techniques can be used but they are not transparent to the user.

Bearing in mind the previous limitations, several projects have already been started with the aim of integrating SQL3 with IR facilities. An example of such a project is represented by SQL/MM Full Text [190]. Text is in this case considered as a nested sequence of words, sentences, and paragraphs. In order to precisely capture the structure and the meaning of the words, SQL/MM Full Text is also able to view the text as a tree structure entity. The structure of this entity is controlled by a grammar. These facilities allow one to easily express queries to perform selection on the basis of the text content and/or text structure.

There have also been several proposals for introducing spatial data types and predicates into the SQL framework. Among them, we recall Probe [623], Spatial SQL [231], Pictorial SQL [687], and QBE [418].

The MULTOS Query Language

The development of the MULTOS query language has been driven by a number of requirements: first, it should be possible to easily navigate through the document structure. *Path-names* can be used for this purpose. Path-names can be total, if the path identifies only one component, or partial, if several components are identified by the path. Path-names are similar to object-oriented path expressions. Queries both on the content and on document structure must be supported.

Query predicates on complex components must be supported. In this case, the predicate applies to all the document subcomponents that have a type compatible with the type required by the query. This possibility is very useful when a user does not recall the structure of a complex component.

In general, a MULTOS query has the form:

```
FIND DOCUMENTS VERSION version-clause
SCOPE scope-clause
TYPE type-clause
WHERE condition-clause
WITH component
```

where:

- The *version-clause* specifies which versions of the documents should be considered by the query.

- The *scope-clause* restricts the query to a particular set of documents. This set of documents is either a user-defined document collection or a set of documents retrieved by a previous query.
- The *type-clause* allows the restriction of a query to documents belonging to a prespecified set of types. The conditions expressed by the *condition-clause* only apply to the documents belonging to these types and their subtypes. When no type is specified, the query is applied to all document types.
- The *condition-clause* is a Boolean combination of simple conditions (i.e., predicates) on documents components. Predicates are expressed on conceptual components of documents. Conceptual components are referenced by path-names. The general form of a predicate is:

  component restriction

  where *component* is a path-name and *restriction* is an operator followed by an expression.
- The *with-clause* allows one to express structural predicates. *Component* is a path-name and the clause looks for all documents structurally containing such a component.

Different types of conditions can be specified in order to query different types of media. In particular, MULTOS supports three main classes of predicates: predicates on data attributes, on which an exact match search is performed; predicates on textual components, determining all objects containing some specific strings; and predicates on images, specifying conditions on the image content. Image predicates allow one to specify conditions on the class to which an image should belong or conditions on the existence of a specified object within an image and on the number of occurrences of an object within an image. The following example illustrates the basic features of the MULTOS query language.

Example 2 *Consider the conceptual structure* `Generic_letter`, *presented in example 1. The following is an example of query:*

```
FIND DOCUMENT VERSIONS LAST WHERE
Document.Date > 1/1/1998 AND
(*Sender.Name = "Olivetti" OR
*Product_Presentation CONTAINS "Olivetti") AND
*Product_Description CONTAINS "Personal Computer" AND
(*Address.Country = "Italy" OR TEXT CONTAINS "Italy") AND
WITH *Company_Logo.
```

According to this query, the user looks for the last version of all documents, dated after January 1998, containing a company logo, having the word 'Olivetti' either as sender name or in the product presentation (which is a textual component), with the word 'Personal Computer' in the product description section

(which is another textual component) and with the word 'Italy' either constituting the country in the address or contained in any part of the entire document. Symbol '' indicates that the path-name is not complete, that is, it could identify more than one component.*

The query language provided by MULTOS also supports the specification of imprecise queries that can be used when the user has an uncertain knowledge about the content of the documents he/she is seeking [657]. Such uncertainty is expressed by associating both a *preference* and an *importance* value with the attributes in the query. Such values are then used for ranking the retrieved documents. The following example illustrates the discussion.

Example 3 *The query:*

```
FIND DOCUMENT VERSIONS LAST WHERE
(Document.Date BETWEEN (12/31/1998,1/31/98) PREFERRED
BETWEEN (2/1/1998,2/15/98) ACCEPTABLE) HIGH AND
(*Sender.Name = "Olivetti" OR
*Product_Presentation CONTAINS "Olivetti") HIGH AND
(*Product_Description CONTAINS "Personal Computer") HIGH AND
(*Product_Description CONTAINS "good ergonomics") LOW AND
(*Address.Country = "Italy" OR TEXT CONTAINS "Italy") HIGH
AND
WITH *Company_Logo HIGH
(IMAGE MATCHES
screen HIGH
keyboard HIGH
AT LEAST 2 floppy_drives LOW) HIGH
```

finds the last versions of all documents written in January, but possibly even at the beginning of February 1998, containing a company logo, having the word 'Olivetti' either as sender name or in the product presentation, with the word 'Personal Computer' in the product description section, and with the word 'Italy' either constituting the country in the address or contained in any part of the entire document. Personal Computers are described in the product description section as products having good ergonomics. Moreover, the document should contain a picture of the Personal Computer, complete with screen and keyboard, with at least two floppy drives. The value 'LOW' *associated with the condition on 'good ergonomics' indicates that the user formulating the query is not completely sure about this description of PC. By contrast, he/she is sure of all the conditions whose associated value is* 'HIGH.'

11.4 Trends and Research Issues

In this chapter we have discussed the main issues in developing a Multimedia IR system. We have observed that only the integration of DBMS and IR technologies provides the ability to represent, store, and manipulate multimedia data and, at the same time, to retrieve those data by applying content-based searches.

We then discussed the main issues arising in defining a data model for multimedia data. Since multimedia data has, in general, a complex structure, the data model must be able to reflect and manage this complexity. Object-oriented or object-relational data models represent the right technology for multimedia data representation. Additional relevant requirements include the support of semi-structured data and metadata. Another important requirement is the ability to internally represent the content of multimedia data in a way that ensures fast retrieval of the stored data and efficient processing of content-based queries. To achieve this goal, semantic features can be extracted from the data, stored inside the system, and used during query processing.

The second topic discussed in this chapter is related to multimedia query languages. We observed that a multimedia query language is characterized by the type of interface presented to the user and the types of predicates it allows in a query. Such predicates are used to perform content-based searches and to let the user drive the selection of relevant objects.

Examples of commercial and prototype systems have been discussed, with respect to the data modeling and query language capabilities.

Several aspects require further investigation. For example, even though SQL3 supports multimedia data representation, it cannot be taken as the basis for the definition of a Multimedia IR system. Additional research is needed to integrate SQL3 with specific language constructs and underlying techniques to perform information retrieval and query optimization.

Another topic is related to XML (see Chapter 6), the new standard format for data on the Web [304]. XML is a text-based format, providing a standard data model to encode the content, the semantics, and the schema of ordinary documents, structured records, and metacontent information about a Web site. The extension of such a standard to support multimedia data and content-based queries is an important research direction.

A further direction concerns the techniques for ranking the objects returned by a partial-match query. Such ranking usually only takes into account the degree of similarity of the objects retrieved with the query request. However, other factors can be considered, such as the profile of the user submitting the query, or the history of the previous queries specified by the user. Taking into account these aspects is very important, since it gives rise to a *customized ranking* which is closer to the user needs.

11.5 Bibiographic Discussion

As we have seen, due to their complex nature, the object-oriented paradigm seems the right approach to model multimedia data. Details about object-oriented database models and architectures can be found in [89]. The object database standard, as defined by the Object Database Management Group, is presented in [150].

On the research side, several models have been proposed for multimedia

data. Such proposals range from data models suitable for a particular media type, like data models for videos [211, 238, 297, 621], data models for images [170] or models for spatial data [623], to general-purpose multimedia data models [169, 296, 397, 545, 759, 827].

Issues related to the definition and the classification of metadata in the multimedia context are extensively discussed in [99, 442]. Among the systems supporting similarity-based queries, we recall QBIC [257], Name-It [708], QBE [418], Probe [623], and PICQUERY [418]. For additional details about video and image multimedia databases we refer the reader to [405] and [438], respectively. Details about modeling and architectural aspects of the MULTOS system can be found in [759].

Chapter 12
Multimedia IR: Indexing and Searching

by Christos Faloutsos

12.1 Introduction

The problem we focus on here is the design of fast searching methods that will search a database of multimedia objects to locate objects that match a query object, exactly or approximately. Objects can be two-dimensional color images, gray-scale medical images in 2D or 3D (e.g., MRI brain scans), one-dimensional time series, digitized voice or music, video clips, etc. A typical query by content would be, e.g., '*in a collection of color photographs, find ones with the same color distribution as a sunset photograph.*'

Specific applications include image databases; financial, marketing and production time series; scientific databases with vector fields; audio and video databases; DNA/Genome databases; etc. In such databases, typical queries would be '*find companies whose stock prices move similarly,*' or '*find images that look like a sunset,*' or '*find medical X-rays that contain something that has the texture of a tumor.*' Searching for similar patterns in such databases as the above is essential, because it helps in predictions, computer-aided medical diagnosis and teaching, hypothesis testing and, in general, in 'data mining' [8] and rule discovery.

Of course, the distance of two objects has to be quantified. We rely on a domain expert to supply such a distance function $\mathcal{D}()$:

Definition *Given two objects, O_1 and O_2, the distance (= dissimilarity) of the two objects is denoted by*

$$\mathcal{D}(O_1, O_2) \tag{12.1}$$

For example, if the objects are two (equal-length) time series, the distance $\mathcal{D}()$ could be their Euclidean distance (the root of the sum of squared differences).

Similarity queries can been classified into two categories:

- **Whole match** Given a collection of N objects $O_1, O_2, \ldots, O_N$ and a query object Q, we want to find those data objects that are within distance ε from Q. Notice that the query and the objects are of the same type: for example, if the objects are 512×512 gray-scale images, so is the query.
- **Sub-pattern match** Here the query is allowed to specify only part of the object. Specifically, given N data objects (e.g., images) $O_1, O_2, \ldots, O_N$, a query (sub-)object Q and a tolerance ε, we want to identify the parts of the data objects that match the query. If the objects are, e.g., 512×512 gray-scale images (like medical X-rays), in this case the query could be, e.g., a 16×16 subpattern (e.g., a typical X-ray of a tumor).

Additional types of queries include the '*nearest neighbors*' queries (e.g., '*find the five most similar stocks to IBM's stock*') and the '*all pairs*' queries or '*spatial joins*' (e.g., '*report all the pairs of stocks that are within distance ε from each other*'). Both the above types of queries can be supported by the approach we describe next. As we shall see, we reduce the problem into searching for multi-dimensional points, which will be organized in R-trees; in this case, nearest-neighbor search can be handled with a branch-and-bound algorithm and the spatial join query can be handled with recent, highly fine-tuned algorithms, as discussed in section 12.8. Thus, we do not focus on nearest-neighbor and 'all-pairs' queries.

For all the above types of queries, the ideal method should fulfill the following requirements:

- It should be *fast*. Sequential scanning and distance calculation with each and every object will be too slow for large databases.
- It should be '*correct*.' In other words, it should return all the qualifying objects, without missing any (i.e., no 'false dismissals'). Notice that 'false alarms' are acceptable, since they can be discarded easily through a post-processing step. Of course, as we see, e.g. in Figure 12.5, we try to keep their number low (but not necessarily minimal), so that the total response time is minimized.
- The ideal method should require a small space overhead.
- The method should be dynamic. It should be easy to insert, delete, and update objects.

As we see next, the heart of the presented 'GEMINI' approach is to use f feature extraction functions to map objects into points in f-dimensional space; thus, we can use highly fine-tuned database spatial access methods to accelerate the search.

The remainder of the chapter is organized as follows. Section 12.2 gives some background material on past related work on spatial access methods. Section 12.3 describes the main ideas for GEMINI, a generic approach to indexing multimedia objects. Section 12.4 shows the application of the approach for 1D time series indexing. Section 12.5 gives another case study, for color images,

within the QBIC project. Section 12.6 presents 'FastMap', a method to do automatic feature extraction. Section 12.7 summarizes the conclusions and lists problems for future research and section 12.8 provides pointers to the related bibliography.

12.2 Background — Spatial Access Methods

As mentioned earlier, the idea is to map objects into points in f-D space, and to use multiattribute access methods (also referred to as *spatial access methods* or SAMs) to cluster them and to search for them.

Thus, a brief introduction to multidimensional indexing methods (or spatial access methods) is in order. The prevailing methods form three classes: (1) R^*-trees and the rest of the R-tree family, (2) linear quadtrees, and (3) grid-files.

Several of these methods explode exponentially with the dimensionality, eventually reducing to sequential scanning. For linear quadtrees, the effort is proportional to the hypersurface of the query region [244]; the hypersurface grows exponentially with the dimensionality. Grid files face similar problems, since they require a directory that grows exponentially with the dimensionality. The R-tree-based methods seem to be most robust for higher dimensions, provided that the fanout of the R-tree nodes remains > 2. Below, we give a brief description of the R-tree method and its variants, since it is one of the typical representatives of spatial access methods.

The R-tree represents a spatial object by its minimum bounding rectangle (MBR). Data rectangles are grouped to form parent nodes, which are recursively grouped, to form grandparent nodes and, eventually, a tree hierarchy. The MBR of a parent node completely contains the MBRs of its children; MBRs are allowed to overlap. Nodes of the tree correspond to disk pages. Disk pages, or 'disk blocks', are consecutive byte positions on the surface of the disk that are typically fetched with one disk access. The goal of the insertion, split, and deletion routines is to give trees that will have good clustering, with few, tight parent MBRs. Figure 12.1 illustrates data rectangles (in black), organized in an R-tree with fanout 3. Figure 12.2 shows the file structure for the same R-tree, where nodes correspond to disk pages.

A range query specifies a region of interest, requiring all the data regions that intersect it. To answer this query, we first retrieve a superset of the qualifying data regions: we compute the MBR of the query region, and then we recursively descend the R-tree, excluding the branches whose MBRs do not intersect the query MBR. Thus, the R-tree will give us quickly the data regions whose MBR intersects the MBR of the query region. The retrieved data regions will be further examined for intersection with the query region.

Algorithms for additional operations (nearest neighbor queries, spatial joins, insertions, and deletions) are more complicated and are still under research (see the Bibliographic Discussion).

The original R-tree paper inspired much follow-up work, as described in

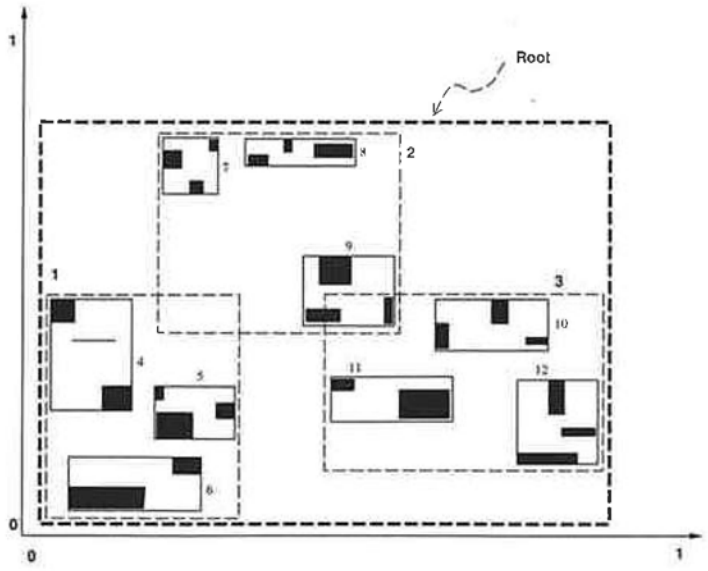


Figure 12.1 Data (dark rectangles) organized in an R-tree with fanout = 3. Solid, light-dashed, and heavy-dashed lines indicate parents, grandparents and great-grandparent (the root, in this example).

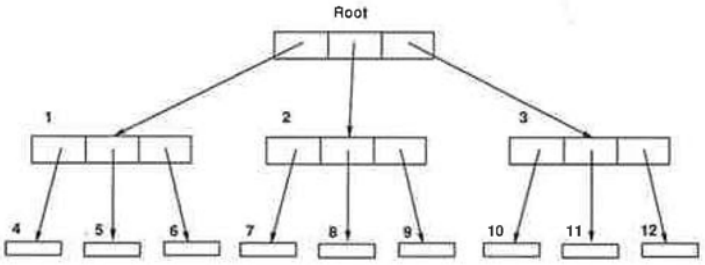


Figure 12.2 The file structure for the R-tree of the previous figure (fanout = 3).

section 12.8. It is important to highlight, however, that *any* spatial access method can be used (like R^*-trees, X-trees, SR-trees, and so on).

12.3 A Generic Multimedia Indexing Approach

To illustrate the basic idea, we shall focus on 'whole match' queries. For such queries the problem is defined as follows:

- We have a collection of N objects: O_1, O_2, ..., O_N.
- The distance/dissimilarity between two objects (O_i, O_j) is given by the function $\mathcal{D}(O_i, O_j)$, which can be implemented as a (possibly, slow) program.
- The user specifies a query object Q, and a tolerance ε.

Our goal is to find the objects in the collection that are within distance ε from the query object. An obvious solution is to apply sequential scanning: For each and every object O_i $(1 \leq i \leq N)$, we can compute its distance from Q and report the objects with distance $\mathcal{D}(Q, O_i) \leq \varepsilon$.

However, sequential scanning may be slow, for two reasons:

(1) The distance computation might be expensive. For example, as discussed in Chapter 8, the editing distance in DNA strings requires a dynamic programming algorithm, which grows like the product of the string lengths (typically, in the hundreds or thousands, for DNA databases).
(2) The database size N might be huge.

Thus, we are looking for a faster alternative. The GEMINI (GEneric Multimedia object INdexIng) approach we present next, is based on two ideas, each of which tries to avoid each of the two disadvantages of sequential scanning:

- a 'quick-and-dirty' test, to discard quickly the vast majority of non-qualifying objects (possibly, allowing some false alarms);
- the use of spatial access methods, to achieve faster-than-sequential searching.

The case is best illustrated with an example. Consider a database of time series, such as yearly stock price movements, with one price per day. Assume that the distance function between two such series S and Q is the Euclidean distance

$$\mathcal{D}(S, Q) \equiv \left(\sum_{i=1} (S[i] - Q[i])^2 \right)^{1/2} \tag{12.2}$$

where $S[i]$ stands for the value of stock S on the i-th day. Clearly, computing the distance of two stocks will take 365 subtractions and 365 squarings in our example.

The idea behind the quick-and-dirty test is to characterize a sequence with a single number, which will help us discard many non-qualifying sequences. Such a number could be, e.g., the average stock price over the year. Clearly, if two stocks differ in their averages by a large margin, it is impossible that they will be similar. The converse is not true, which is exactly the reason we may have false alarms. Numbers that contain some information about a sequence (or a multimedia object, in general), will be referred to as '*features*' for the rest of this chapter. Using a good feature (like the 'average,' in the stock prices example), we can have a quick test, which will discard many stocks, with a single numerical comparison for each sequence (a big gain over the 365 subtractions and squarings that the original distance function requires).

If using one feature is good, using two or more features might be even better, because they may reduce the number of false alarms (at the cost of

making the quick-and-dirty test a bit more elaborate and expensive). In our stock prices example, additional features might be, e.g., the standard deviation, or, even better, some of the discrete Fourier transform (DFT) coefficients, as we shall see in section 12.4.

The end result of using f features for each of our objects is that we can map each object into a point in f-dimensional space. We shall refer to this mapping as $\mathcal{F}()$ (for 'F'eature):

Definition *Let $\mathcal{F}()$ be the mapping of objects to f-dimensional points, that is, $\mathcal{F}(O)$ will be the f-D point that corresponds to object O.*

This mapping provides the key to improve on the second drawback of sequential scanning: by organizing these f-D points into a spatial access method, we can cluster them in a hierarchical structure, like the R^*-trees. Upon a query, we can exploit the R^*-tree, to prune out large portions of the database that are not promising. Thus, we do not even *have* to do the quick-and-dirty test on all of the f-D points!

Figure 12.3 illustrates the basic idea: Objects (e.g., time series that are 365 points long) are mapped into 2D points (e.g., using the average and the standard deviation as features). Consider the 'whole match' query that requires all the objects that are similar to S_n within tolerance ε: this query becomes an f-D sphere in feature space, centered on the image $\mathcal{F}(S_n)$ of S_n. Such queries on multidimensional points is exactly what R-trees and other SAMs are designed to answer efficiently. More specifically, the search algorithm for a whole match query is as follows:

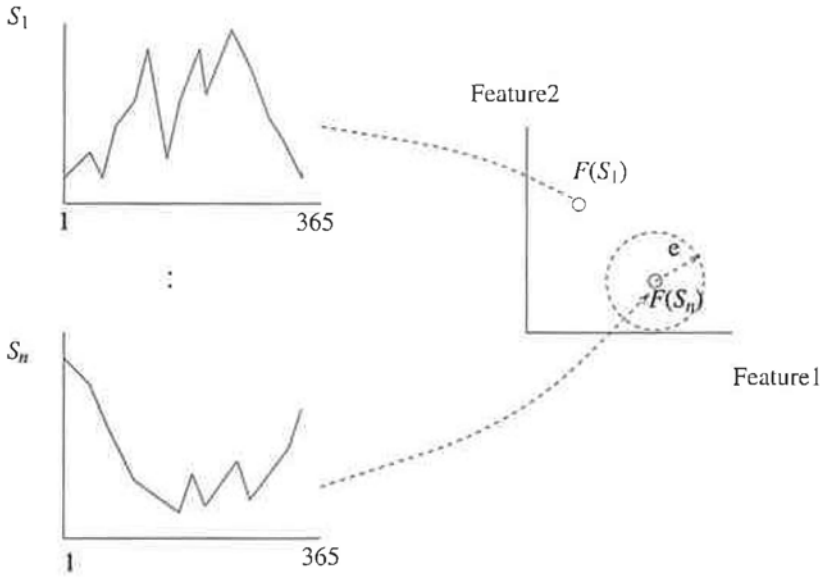


Figure 12.3 Illustration of the basic idea: a database of sequences $S_1, \ldots, S_N$; each sequence is mapped to a point in feature space; a query with tolerance ε becomes a sphere of radius ε.

Algorithm 1 *Search:*

(1) Map the query object Q into a point $\mathcal{F}(Q)$ in feature space.

(2) Using a spatial access method, retrieve all points within the desired tolerance ε from $\mathcal{F}(Q)$.

(3) Retrieve the corresponding objects, compute their actual distance from Q and discard the false alarms.

Intuitively, the method has the potential to relieve both problems of the sequential scan, presumably resulting in much faster searches. The only step that we have to be careful with is that the mapping $\mathcal{F}()$ from objects to f-D points does not distort the distances. Let $\mathcal{D}()$ be the distance function of two objects, and $\mathcal{D}_{feature}()$ be the (say, Euclidean) distance of the corresponding feature vectors. Ideally, the mapping should preserve the distances exactly, in which case the SAM will have neither false alarms nor false dismissals. However, requiring perfect distance preservation might be difficult. For example, it is not obvious which features we have to use to match the editing distance between two DNA strings. Even if the features are obvious, there might be practical problems: for example, in the stock price example, we could treat every sequence as a 365-dimensional vector; although in theory a SAM can support an arbitrary number of dimensions, in practice they all suffer from the 'dimensionality curse,' as discussed earlier.

The crucial observation is that we can guarantee that there will be no false dismissals if the distance in feature space matches or underestimates the distance between two objects. Intuitively, this means that our mapping $\mathcal{F}()$ from objects to points *should make things look closer* (i.e., it should be a contractive mapping).

Mathematically, let O_1 and O_2 be two objects (e.g., same-length sequences) with distance function $\mathcal{D}()$ (e.g., the Euclidean distance) and $\mathcal{F}(O_1)$, $\mathcal{F}(O_2)$ be their feature vectors (e.g., their first few Fourier coefficients), with distance function $\mathcal{D}_{feature}()$ (e.g., the Euclidean distance, again). Then we have:

Lemma 12.1 (Lower Bounding) *To guarantee no false dismissals for whole-match queries, the feature extraction function $\mathcal{F}()$ should satisfy the following formula:*

$$\mathcal{D}_{feature}(\mathcal{F}(O_1), \mathcal{F}(O_2)) \leq \mathcal{D}(O_1, O_2) \tag{12.3}$$

As proved in [249], lower-bounding the distance works correctly for range queries. Will it work for the other queries of interest, like 'all pairs' and 'nearest neighbor' ones? The answer is affirmative in both cases. An 'all pairs' query can easily be handled by a 'spatial join' on the points of the feature space: using a similar reasoning as before, we see that the resulting set of pairs will be a superset of the qualifying pairs. For the nearest neighbor query, the following algorithm guarantees no false dismissals: (1) find the point $\mathcal{F}(P)$ that is the

nearest neighbor to the query point $\mathcal{F}(Q)$, (2) issue a range query, with query object Q and radius $\varepsilon = \mathcal{D}(Q, P)$ (i.e., the actual distance between the query object Q and data object P).

In conclusion, the GEMINI approach to indexing multimedia objects for fast similarity searching is as follows:

Algorithm 2 *(GEMINI) GEneric Multimedia object INdexIng approach:*

(1) Determine the distance function $\mathcal{D}()$ between two objects.

(2) Find one or more numerical feature-extraction functions, to provide a 'quick-and-dirty' test.

(3) Prove that the distance in feature space lower-bounds *the actual distance $\mathcal{D}()$, to guarantee correctness.*

(4) Use a SAM (e.g., an R-tree), to store and retrieve the f-D feature vectors.

The first two steps of GEMINI deserve some more discussion: the first step involves a domain expert. The methodology focuses on the *speed* of search only; the quality of the results is completely relying on the distance function that the expert will provide. Thus, GEMINI will return *exactly the same* response set (and therefore, the same quality of output, in terms of precision-recall) that would be returned by a sequential scanning of the database; the only difference is that GEMINI will be faster.

The second step of GEMINI requires intuition and imagination. It starts by trying to answer the question (referred to as the '*feature-extracting*' question for the rest of this chapter):

> **'Feature-extracting' question:** *If we are allowed to use only one numerical feature to describe each data object, what should this feature be?*

The successful answers to the above question should meet two goals: first, they should facilitate step 3 (the distance lower-bounding), and second, they should capture most of the characteristics of the objects.

We give case studies of steps 2 and 3 of the GEMINI algorithm in the following sections. The first involves 1D time series, and the second focuses on 2D color images. We shall see that the philosophy of the quick-and-dirty filter, in conjunction with the lower-bounding lemma, can lead to solutions to two problems:

- the dimensionality curse (time series)
- the 'cross-talk' of features (color images).

For each case study, we first describe the objects and the distance function, then show how to apply the lower-bounding lemma, and finally give experimental results, on real or realistic data.

12.4 One-dimensional Time Series

Here the goal is to search a collection of (equal-length) time series, to find the ones that are similar to a desirable series. For example, '*in a collection of yearly stock price movements, find the ones that are similar to IBM.*'

12.4.1 Distance Function

According to GEMINI (algorithm 2), the first step is to determine the distance measure between two time series. A typical distance function is the Euclidean distance (equation 12.2), which is routinely used in financial and forecasting applications. Additional, more elaborate distance functions, that, for example, include time-warping, are discussed in section 12.8.

12.4.2 Feature Extraction and Lower-bounding

Having decided on the Euclidean distance as the dissimilarity measure, the next step is to find some features that can lower-bound it. We would like a set of features that first, preserve/lower-bound the distance, and second, carry much information about the corresponding time series (so that the false alarms are few). The second requirement suggests that we use 'good' features, that have much discriminatory power. In the stock price example, a 'bad' feature would be, e.g., the first day's value: the reason being that two stocks might have similar first-day values, yet they may differ significantly from then on. Conversely, two otherwise similar sequences may agree everywhere, except for the first day's values. At the other extreme, we could use the values of *all* 365 days as features. However, although this would perfectly match the actual distance, it would lead to the 'dimensionality curse' problem.

Clearly, we need some better features. Applying the second step of the GEMINI algorithm, we ask the feature-extracting question: '*If we are allowed to use only one feature from each sequence, what would this feature be?*' A natural answer is the average. By the same token, additional features could be the average of the first half, of the second half, of the first quarter, etc. Or, in a more systematic way, we could use the coefficients of the Fourier transform, and, for our case, the Discrete Fourier Transform (DFT). For a signal $\vec{x} = [x_i]$, $i = 0, \ldots, n-1$, let X_F denote the n-point DFT coefficient at the F-th frequency ($F = 0, \ldots, n-1$).

The third step of the GEMINI methodology is to show that the distance in feature space lower-bounds the actual distance. The solution is provided by Parseval's theorem, which states that the DFT preserves the energy of a signal, as well as the distances between two signals:

$$\mathcal{D}(\vec{x}, \vec{y}) = \mathcal{D}(\vec{X}, \vec{Y}) \tag{12.4}$$

where $\vec{X}$ and $\vec{Y}$ are Fourier transforms of $\vec{x}$ and $\vec{y}$ respectively.

Thus, if we keep the first $f (f \leq n)$ coefficients of the DFT as the features, we lower-bound the actual distance:

$$\begin{aligned} \mathcal{D}_{feature}(\mathcal{F}(\vec{x}), \mathcal{F}(\vec{y})) &= \sum_{F=0}^{f-1} |X_F - Y_F|^2 \\ &\leq \sum_{F=0}^{n-1} |X_F - Y_F|^2 \\ &= \sum_{i=0}^{n-1} |x_i - y_i|^2 \end{aligned}$$

and finally

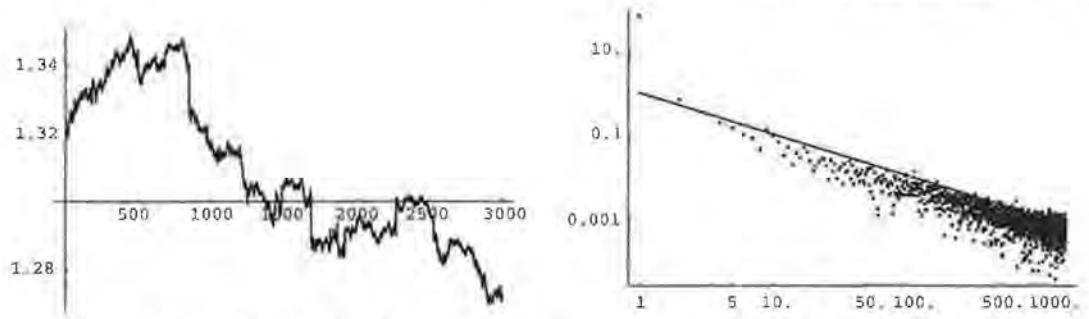
$$\mathcal{D}_{feature}(\mathcal{F}(\vec{x}), \mathcal{F}(\vec{y})) \leq \mathcal{D}(\vec{x}, \vec{y}) \quad \textbf{(12.5)}$$

because we ignore positive terms from equation 12.2. Thus, there will be *no false dismissals*, according to lemma 12.1.

Notice that the GEMINI approach can be applied with *any* orthonormal transform, such as, the Discrete Cosine Transform (DCT), the wavelet transform etc., because they all preserve the distance between the original and the transformed space. In fact, our response time will improve with the ability of the transform to concentrate the energy: the fewer the coefficients that contain most of the energy, the more accurate our estimate for the actual distance, the fewer the false alarms, and the faster our response time. Thus, the performance results presented next are just pessimistic bounds; better transforms will achieve even better response times.

In addition to being readily available, (e.g., in 'Mathematica,' 'S,' 'maple,' 'matlab' etc.), the DFT concentrates the energy in the first few coefficients, for a large class of signals, the *colored noises*. These signals have a skewed energy spectrum ($O(F^{-b})$, as follows:

- For $b = 2$, we have the so-called *random walks* or *brown noise*, which model successfully stock movements and exchange rates (e.g., [541]).
- With even more skewed spectrum ($b > 2$), we have the *black noises* [712]. Such signals model successfully, for example, the water level of rivers and the rainfall patterns as they vary over time [541].
- With $b = 1$, we have the *pink noise*. Birkhoff's theory [712] claims that 'interesting' signals, such as musical scores and other works of art, consist of *pink noise*, whose energy spectrum follows $O(F^{-1})$. The argument of the theory is that white noise with $O(F^0)$ energy spectrum is completely unpredictable, while brown noise with $O(F^{-2})$ energy spectrum is too predictable and therefore 'boring.' The energy spectrum of pink noise lies in between.



(a) time plot (lin-lin) (b) amplitude spectrum (log-log)

Figure 12.4 (a) The Swiss-franc exchange rate (7 August 1990 to 18 April 1991 – first 3000 values out of 30,000) and (b) log-log amplitude of its Fourier transform, along with the $1/F$ line.

As an illustration of the above observations, Figure 12.4(a) plots the movement of the exchange rate between the Swiss franc and the US dollar starting 7 August 1990 (3000 measurements); Figure 12.4(b) shows the amplitude of the Fourier coefficients as a function of the frequency F, as well as the $1/F$ line, in a logarithmic-logarithmic plot. Notice that, since it is successfully modeled as a random walk, the amplitude of the Fourier coefficients follow the $1/F$ line. The above data set is available through anonymous ftp from `sfi.santafe.edu`.

In addition to 1D signals (stock price movements and exchange rates), it is believed that several families of real n-D signals belong to the family of 'colored noises', with skewed spectrum. For example, 2D signals, like photographs, are far from white noise, exhibiting a few strong coefficients in the lower spatial frequencies. The JPEG image compression standard exploits this phenomenon, effectively ignoring the high frequency components of the discrete cosine transform, which is closely related to the Fourier transform. If the image consisted of white noise, no compression would be possible at all.

12.4.3 Experiments

Performance results with the GEMINI approach on time series are reported in [6]. There, the method is compared to a *sequential scanning* method. The R^*-tree was used for the spatial access method within GEMINI. The sequences were artificially generated random walks, with length $n = 1024$; their number N varied from 50 to 400.

Figure 12.5 shows the break-up of the response time, as a function of the number f of DFT coefficients kept. The diamonds, triangles, and squares indicate total time, post-processing time, and R^*-tree time, respectively. Notice that, as we keep more features f, the R^*-tree becomes bigger and slower, but more accurate (fewer false alarms, and therefore shorter post-processing time). This tradeoff reaches an equilibrium for $f = 2$ or 3. For the rest of the experiments, the $f = 2$ Fourier coefficients were kept for indexing, resulting in a four-dimensional R^*-tree (two real numbers for each complex DFT coefficient).

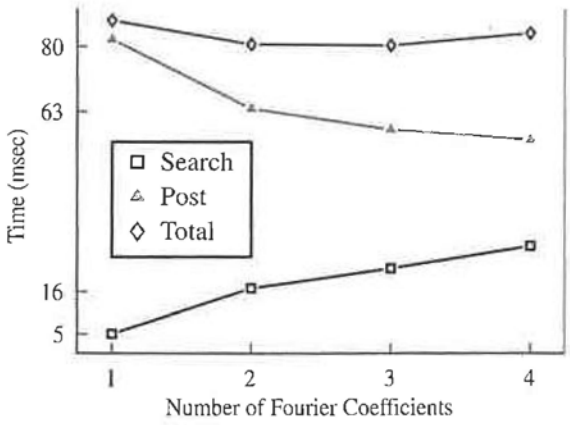


Figure 12.5 Breakup of the execution time, for range query (db size $N = 400$ sequences).

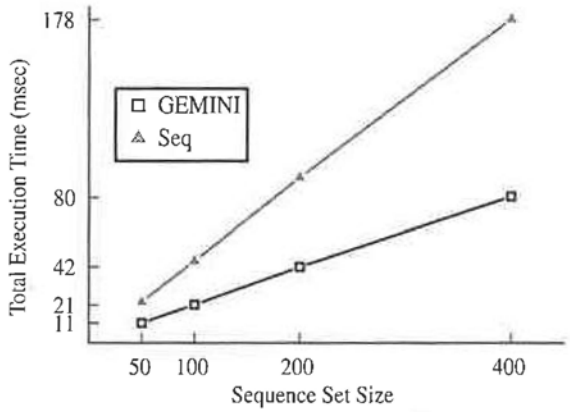


Figure 12.6 Search time per query vs. number N of sequences, for whole-match queries; GEMINI (black line) and sequential scanning (gray line).

Figure 12.6 shows the response time for the two methods (GEMINI and sequential scan), as a function of the number of sequences N. Clearly, GEMINI outperforms the sequential scanning.

The major conclusions from the application of GEMINI on time series are the following:

(1) GEMINI can be successfully applied to time series, and specifically to the ones that behave like 'colored noises' (stock prices movements, currency exchange rates, water level in rivers etc.).

(2) For signals with skewed spectrum like the above ones, the minimum in the response time is achieved for a small number of Fourier coefficients ($f = 1, 2, 3$). Moreover, the minimum is rather flat, which implies that

a suboptimal choice for f will give search time that is close to the minimum. Thus, with the help of the lower-bounding lemma and the energy-concentrating properties of the DFT, we managed to avoid the 'dimensionality curse.'

(3) The success in 1D series suggests that GEMINI is promising for 2D or higher-dimensionality signals, if those signals also have skewed spectrum. The success of JPEG (that uses DCT) indicates that real images indeed have a skewed spectrum.

Finally, the method has been extended to handle subpattern matching; for time sequences, the details are in [249]. We only mention the main idea here. Assuming that query patterns have length of at least w, we preprocess every sequence of the database, by allowing a sliding window of length w at each and every possible position, and by extracting the f features for a given positioning of the window. Thus, every sequence becomes a trail in the f-dimensional feature space, which can be further approximated by a set of few MBRs that cover it. Representing each sequence by a few MBRs in feature space may allow false alarms, but no false dismissals. The approach can be generalized for subpattern matching in 2D signals (and, in general, in n-dimensional vector fields).

12.5 Two-dimensional Color Images

GEMINI has also been applied for color images, within the QBIC project of IBM. The QBIC (Query By Image Content) project studies methods to query large online image databases using the images' content as the basis of the queries. Examples of the content include color, texture, shape, position, and dominant edges of image items and regions. Potential applications include medical ('*Give me other images that contain a tumor with a texture like this one*'), photojournalism ('*Give me images that have blue at the top and red at the bottom*'), and many others in art, fashion, cataloging, retailing, and industry.

Here we will discuss methods on databases of still images, with two main datatypes: 'images' (≡ 'scenes') and 'items.' A scene is a (color) image, and an item is a part of a scene, for example, a person, a piece of outlined texture, or an apple. Each scene has zero or more items. The identification and extraction of items is beyond the scope of this discussion (see [603] for more details).

In this section we give an overview of the indexing aspects of QBIC, and specifically the distance functions and the application of the GEMINI approach. More details about the algorithms and the implementation of QBIC are in [257].

12.5.1 Image Features and Distance Functions

We mainly focus on the color features, because color presents an interesting problem (namely, the '*cross-talk*' of features), which can be resolved by the GEMINI

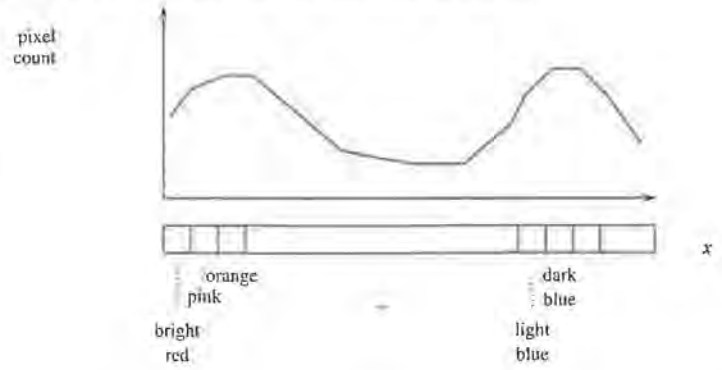


Figure 12.7 An example of a color histogram of a fictitious sunset photograph: many red, pink, orange, purple, and blue-ish pixels; few yellow, white, and green-ish ones.

approach (algorithm 2). For color, we compute a k-element color histogram for each item and scene, where $k = 256$ or 64 colors. Each component in the color histogram is the percentage of pixels that are most similar to that color. Figure 12.7 gives an example of such a histogram of a fictitious photograph of a sunset: there are many red, pink, orange, and purple pixels, but only a few white and green ones.

Once these histograms are computed, one method to measure the distance between two histograms ($k \times 1$ vectors) $\vec{x}$ and $\vec{y}$ is given by

$$d^2_{hist}(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^t \mathcal{A} (\vec{x} - \vec{y}) = \sum_i^k \sum_j^k a_{ij}(x_i - y_i)(x_j - y_j) \qquad \textbf{(12.6)}$$

where the superscript t indicates matrix transposition, and the color-to-color similarity matrix $\mathcal{A}$ has entries a_{ij} which describe the similarity between color i and color j.

12.5.2 Lower-bounding

In applying the GEMINI method for color indexing, there are two obstacles: first, the *'dimensionality curse'* (k may be large, e.g. 64 or 256 for color features) and, most importantly, the *quadratic nature of the distance function.* The distance function in the feature space involves cross-talk among the features (see equation 12.6), and thus it is a full quadratic form involving all cross terms. Not only is such a function much more expensive to compute than a Euclidean (or any L_p) distance, but it also precludes efficient implementation of commonly used spatial access methods. Figure 12.8 illustrates the situation. To compute the distance between the two color histograms $\vec{x}$ and $\vec{q}$, the, e.g., bright-red component of $\vec{x}$ has to be compared not only to the bright-red component of $\vec{q}$, but also to the pink, orange, etc. components of $\vec{q}$.

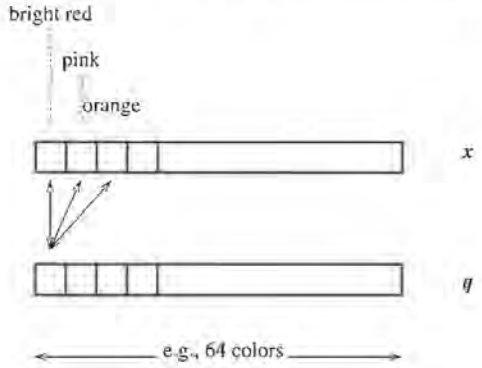


Figure 12.8 Illustration of the 'cross-talk' between two color histograms.

To resolve the cross-talk problem, we try to apply the GEMINI approach (algorithm 2). The first step of the algorithm has been done: the distance function between two color images is given by equation 12.6, that is, $\mathcal{D}() = d_{hist}()$. The second step is to find one or more numerical features, whose Euclidean distance would lower-bound $d_{hist}()$. Thus, we ask the feature-extracting question again: *If we are allowed to use only one numerical feature to describe each color image, what should this feature be?* Taking a cue from the previous section on time series, we can consider some average value, or the first few coefficients of the two-dimensional DFT transform. Since we have three color components, (e.g., Red, Green, and Blue), we could consider the average amount of red, green, and blue in a given color image.

Notice that different color spaces (such as Munsell) can be used, with absolutely no change in our indexing algorithms. Thus, we continue the discussion with the RGB color space. This means that the color of an individual pixel is described by the triplet (R,G,B) (for 'R'ed, 'G'reen, 'B'lue). The average color vector of an image or item $\bar{x} = (R_{avg}, G_{avg}, B_{avg})^t$, is defined in the obvious way, with

$$R_{avg} = (1/P)\sum_{p=1}^{P} R(p),$$

$$G_{avg} = (1/P)\sum_{p=1}^{P} G(p),$$

$$B_{avg} = (1/P)\sum_{p=1}^{P} B(p)$$

where P is the number of pixels in the item, and $R(p)$, $G(p)$, and $B(p)$ are the red, green and blue components (intensities, typically in the range 0–255) respectively of the p-th pixel. Given the average colors $\bar{x}$ and $\bar{y}$ of two items, we define $d_{avg}()$ as the Euclidean distance between the three-dimensional average

color vectors,

$$d^2_{avg}(\bar{x}, \bar{y}) = (\bar{x} - \bar{y})^t(\bar{x} - \bar{y}) \quad (12.7)$$

The third step of the GEMINI algorithm is to prove that our simplified distance $d_{avg}()$ lower-bounds the actual distance $d_{hist}()$. Indeed, this is true, as an application of the so-called Quadratic Distance Bounding or QDB Theorem (see [244]).

The result is that, given a color query, our retrieval proceeds by first filtering the set of images based on their average (R, G, B) color, then doing a final, more accurate matching using their full k-element histogram. The resulting speedup is discussed next.

12.5.3 Experiments

We now present experimental results [244] with GEMINI on color, using the bounding theorem, The experiments compare the relative performance (in terms of CPU time and disk accesses) between first, simple sequential evaluation of d_{hist} for all database vectors (referred to as *'naive'*), and second, GEMINI.

The experiments report the total and the CPU times required by the methods, by performing simulations on a database of $N = 924$ color image histograms, each of $k = 256$ colors, of assorted natural images.

Results are shown in Figure 12.9, which presents the total response time as a function of the selectivity (ratio of actual hits over the database size N). The figure also shows the CPU time for each method. Notice that, even for a selectivity of 5% (which would return ≈ 50 images to the user), the GEMINI method is much faster than the straightforward, sequential computation of the histogram distances. In fact, it requires from a fraction of a second up to ≈ 4 seconds, while the naive method requires consistently ≈ 10 seconds. Moreover, notice that for larger databases, the naive method will have a linearly increasing response time.

Thus, the conclusions are the following:

- The GEMINI approach (i.e., the idea to extract some features for a quick-and-dirty test) motivated a fast method, using the average RGB distance; it also motivated a strong theorem (the so-called QDB theorem [244]) which guarantees the correctness in our case.
- In addition to resolving the cross-talk problem, GEMINI solved the 'dimensionality curse' problem at no extra cost, requiring only $f = 3$ features, as opposed to $k = 64$ or 256 that $d_{hist}()$ required.

12.6 Automatic Feature Extraction

GEMINI is useful for any setting that we can extract features from. In fact, algorithms for automatic feature extraction methods exist, like the 'Multidimen-

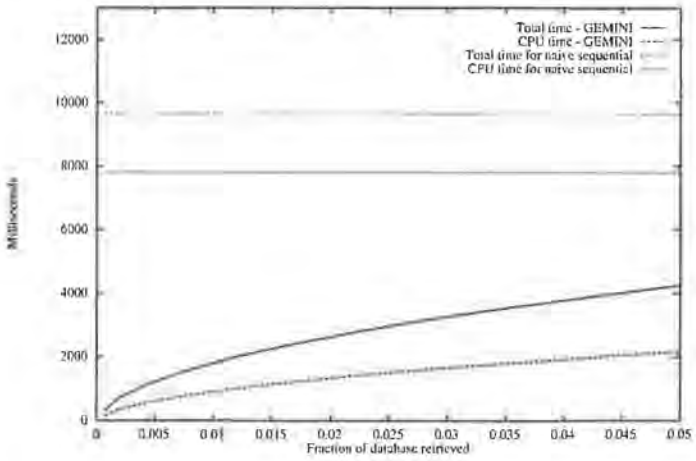


Figure 12.9 Response time vs. selectivity, for the sequential ('naive') retrieval and for GEMINI.

sional Scaling' (MDS) and 'FastMap.' Extracting features not only facilitates the use of off-the-shelf spatial access methods, but it also allows for visual data mining: we can plot a 2D or 3D projection of the data set, and inspect it for clusters, correlations, and other patterns.

Figure 12.10 shows the results of FastMap on 35 documents of seven classes, after deriving $k = 3$ features/dimensions. The classes include basketball reports ('Bbr'), abstracts of computer science technical reports ('Abs'), cooking recipes ('Rec'), and so on. The distance function was a decreasing function of the cosine similarity. The figure shows the 3D scatter-plot, (a) in its entirety and (b) after zooming into the center, to highlight the clustering abilities of FastMap. Notice that the seven classes are separated well, in only $k = 3$ dimensions.

12.7 Trends and Research Issues

In this chapter we focused on how to accelerate queries by content on image databases and, more general, on multimedia databases. Target queries are, e.g., '*find images with a color distribution of a sunset photograph*;' or, '*find companies whose stock price moves similarly to a given company's stock.*'

The method expects a distance function $\mathcal{D}()$ (given by domain experts), which should measure the dissimilarity between two images or objects O_1, O_2. We mainly examined *whole match, range* queries (that is, *'queries by example'* where the user specifies the ideal object and asks for all objects that are within distance ε from the ideal object). Extensions to other types of queries (nearest neighbors, all pairs and subpattern match) are briefly discussed. We focused on the GEMINI approach, which combines two ideas:

- The first is to devise a '*quick-and-dirty*' test, which will eliminate several

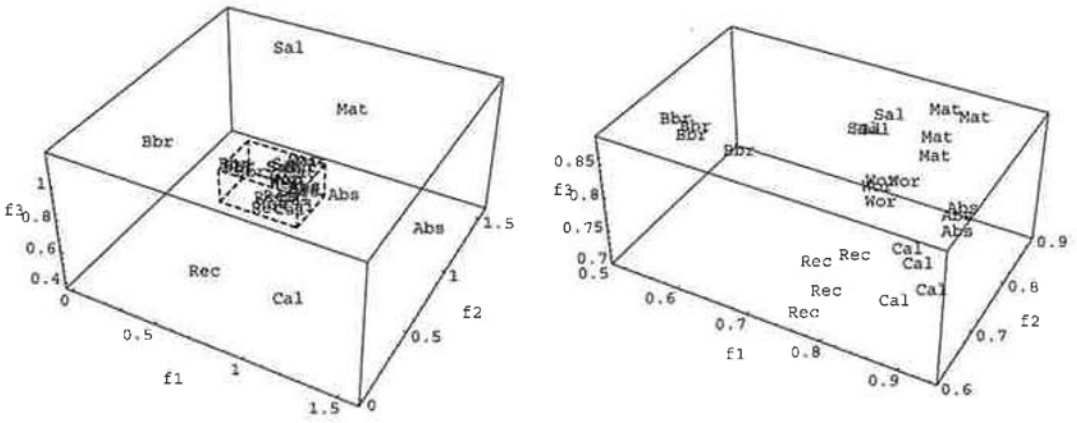


Figure 12.10 A collection of documents, after FastMap in 3-D space: (a) the whole collection and (b) magnification of the dashed box.

non-qualifying objects. To achieve that, we should extract f numerical features from each object, which should somehow describe the object (for example, the first few DFT coefficients for a time sequence, or for a gray-scale image). The key question to ask is *'If we are allowed to use only one numerical feature to describe each data object, what should this feature be?'*

- The second idea is to further accelerate the search, by organizing these f-dimensional points using state-of-the art spatial access methods [400], like the R^*-trees. These methods typically group neighboring points together, thus managing to discard large unpromising portions of the address space early.

The above two ideas achieve fast searching. Moreover, we need to consider the condition under which the above method will be not only fast, but also *correct*, in the sense that it will not miss any qualifying object. Notice that false alarms are acceptable, because they can be discarded, in the obvious way. The answer is provided by the *lower-bounding* lemma, which intuitively states that the mapping $\mathcal{F}()$ of objects to f-D points should *make things look closer.*

In the rest of the chapter, we discussed how to apply GEMINI for a variety of environments, like 1D time sequences and 2D color images. As discussed in the bibliographic notes, GEMINI has been applied to multiple other settings, like tumor-like shapes, time sequences with the time-warping distance function, 2D medical images, and so on. Moreover, it is one of the main reasons behind a strong recent interest on high-dimensionality index structures.

With respect to future trends, probably the most notable and most challenging trend is data mining in multimedia and mixed-media data sets. For example, given a collection of medical records, with demographic data, text data

(like history), 2D images (like X-rays), and 1D signals (electrocardiograms), we want to find correlations, clusters, patterns, and outliers. Successful detection of such patterns is the basis for forecasting, for hypothesis formation, anomaly detection, and several other knowledge discovery operations. GEMINI, insisting on turning every data type into a feature vector, should prove extremely useful: the reason is that it opens the door for off-the-shelf statistical and machine learning packages, which typically expect a set of vectors as input. Typical such packages are the 'Principal Component Analysis' (PCA, also known as 'Latent Semantic Indexing' (LSI), 'Karhunen-Loeve Transform' (KLT), and 'Singular Value Decomposition' (SVD)), Artificial Neural Networks, tree classifiers, to name a few.

12.8 Bibliographic Discussion

Spatial Access Methods

Structures and Algorithms

For a recent, very thorough survey of spatial access methods, see [290]. For the introduction of R-trees, see the seminal paper by Guttman [330]. Among the numerous follow-up variations, the R^*-tree [69] seems to be one of the best performing methods, using the idea of deferred splitting with 'forced-reinsert,' thus achieving higher space utilization, and therefore more compact, shorter, and faster trees. Another strong contender is the Hilbert R-tree [427], which achieves even higher space utilization and often outperforms the R^*-tree. A generalized framework and implementation for all these methods is the GiST tree [362] which is available, at the time of writing, at `http://gist.cs.berkeley.edu:8000/gist`.

With respect to algorithms, the range search is trivial in R-trees. Nearest neighbors queries require more careful record keeping, with a branch-and-bound algorithm (e.g., [686]). Spatial joins (e.g., 'find all pairs of points within distance ϵ') have also attracted a lot of interest: see the filtering algorithms in [119] and the methods in [521] and [458].

Indexing high-dimensional address spaces has attracted a lot of recent interest: the TV-trees [519] adaptively use only a few of the available dimensions. The SR-trees [431] use spheres in conjunction to rectangles, as bounding regions. The more recent X-trees [83] gracefully switch to sequential scanning for extremely high dimensionalities.

For the analysis of spatial access methods and selectivity estimation, the concept of 'fractal dimension' has given very accurate results in every case it was tried: range queries [247], nearest neighbor queries [628], spatial joins [79], quadtrees [245]. The idea behind the fractal dimension is to consider the intrinsic dimensionality of the given set of points. For example, consider the points on the diagonal of a 3D cube: their 'embedding' dimensionality is $E = 3$; however, their intrinsic dimensionality is $D = 1$. Using the appropriate definition for the dimensionality, like the Hausdorff fractal dimension, or the correlation fractal

dimension [712], it turns out that real data sets have a fractional dimensionality: the value is 1.1–1.2 for coastlines, ≈2.7 for the brain surface of mammals, ≈1.3 for the periphery of rain patches, ≈1.7 for the end-points of road segments, to name but a few [247].

Metric Trees
Finally, a class of access methods that operate on the distance function directly seems promising. These methods require only a distance function, and they typically build a cluster hierarchy, that is, a tree structure of 'spheres', which include the children spheres, and so on, recursively. This class includes the Burkhard-Keller methods [131], the Fixed-query trees [47], the GNAT trees [116], the MVP trees [112], and the M-trees [172]. The technology is still young: most of the above methods are designed for static data sets. On the positive side, they don't need feature extraction; on the negative side, they don't provide for visualization and data mining, like GEMINI and FastMap do (see Figure 12.10).

Multimedia Indexing, DSP and Feature Extraction

GEMINI — Feature Extraction
Probably the earliest paper that suggested feature extraction for fast indexing is [400], for approximate matching in shapes. The proof of the lower bounding lemma is in [249].

Algorithms for automatic feature extraction include the traditional, Multidimensional Scaling (MDS), see, e.g., [462]. MDS has attracted tremendous interest, but it is $O(N^2)$, quadratic on the number of database objects N. Thus, it is impractical for large data sets. An $O(N)$ alternative is the so-called FastMap [248], which was used to produce Figure 12.10.

Time Sequences
For additional, more elaborate distance functions, that include time-warping, see Chapter 8 or [706]. An indexing method with the time-warping distance function has recently been developed [840], using FastMap.

For linear time sequence forecasting, see the classic book on the Box-Jenkins methodology [109]. For more recent, non-linear forecasting methods, see the intriguing volumes from the Santa-Fe Institute [149, 808].

Digital Signal Processing (DSP)
Powerful tools for the analysis of time sequences and n-D signals in general include the traditional Fourier transform (see, e.g., [622]), the popular discrete cosine transform, which is the basis for the JPEG image compression standard [802], and the more recent, and even more effective, wavelet transform (DWT) [689]. An excellent introduction to all these methods, as well as source code, is available in [651].

Image Features and Similarity Functions
There is a lot of work in machine vision on feature extraction and similarity measures. Classic references are e.g., [53, 224, 285]. A recent survey on image registration and image comparison methods is in [125]. The proof for quadratic distance bounding theorem of section 12.5 is in [244].

Other Applications of Multimedia Indexing
There are numerous papers on indexing in multimedia databases. A small sample of them include the following: for time sequences allowing scaling or subpattern matching, see [305], [7], [246]. For voice and video see, e.g., [800]. For shapes see, e.g., [244]. For medical image databases see, e.g., [381], [454], [635]. For multimedia searching on the Web, see, e.g., [4, 733, 80, 714].

Data Mining
Finally, there is a lot of work on traditional machine learning [565] and statistics (e.g., [408]).

Chapter 13
Searching the Web

13.1 Introduction

The World Wide Web dates from the end of the 1980s [85] and no one could have imagined its current impact. The boom in the use of the Web and its exponential growth are now well known. Just the amount of textual data available is estimated to be in the order of one terabyte. In addition, other media, such as images, audio, and video, are also available. Thus, the Web can be seen as a very large, unstructured but ubiquitous database. This triggers the need for efficient tools to manage, retrieve, and filter information from this database. This problem is also becoming important in large intranets, where we want to extract or infer new information to support a decision process, a task called data mining. As mentioned in Chapter 1, we make the important distinction between data and information retrieval. We are interested in the latter case, in which the user searches for data that fulfills his information need.

We focus on text, because although there are techniques to search for images and other non-textual data, they cannot be applied (yet) on a large scale. We also emphasize syntactic search. That is, we search for Web documents that have user-specified words or patterns in their text. As discussed in Chapter 2, such words or patterns may or may not reflect the intrinsic semantics of the text. An alternative approach to syntactic search is to do a natural language analysis of the text. Although the techniques to preprocess natural language and extract the text semantics are not new, they are not yet very effective and they are also too costly for large amounts of data. In addition, in most cases they are only effective with well structured text, a thesaurus, and other contextual information.

There are basically three different forms of searching the Web. Two of them are well known and are frequently used. The first is to use search engines that index a portion of the Web documents as a full-text database. The second is to use Web directories, which classify selected Web documents by subject. The third and not yet fully available, is to search the Web exploiting its hyperlink†

† We will use hyperlink or link to denote a pointer (anchor) from a Web page to another Web page.

structure. We cover all three forms of Web search here.

We first discuss the challenges of searching the Web, followed by some Web statistics and models which can be used to understand the complexity of the problem. Next, we discuss in detail the main tools used today to search the Web. The discussion includes search engines, Web directories, hybrid systems, user interfaces, and searching examples. We continue with new query languages that exploit the graphical structure of the Web. Finally, we survey current trends and research issues. As Web research is a very dynamic field, we may have missed some important work, for which we apologize in advance.

13.2 Challenges

We now mention the main problems posed by the Web. We can divide them in two classes: problems with the data itself and problems regarding the user and his interaction with the retrieval system. The problems related to the data are:

- **Distributed data**: due to the intrinsic nature of the Web, data spans over many computers and platforms. These computers are interconnected with no predefined topology and the available bandwidth and reliability on the network interconnections varies widely.
- **High percentage of volatile data**: due to Internet dynamics, new computers and data can be added or removed easily (it is estimated that 40% of the Web changes every month [424]). We also have dangling links and relocation problems when domain or file names change or disappear.
- **Large volume**: the exponential growth of the Web poses scaling issues that are difficult to cope with.
- **Unstructured and redundant data**: most people say that the Web is a distributed hypertext. However, this is not exactly so. Any hypertext has a conceptual model behind it, which organizes and adds consistency to the data and the hyperlinks. That is hardly true in the Web, even for individual documents. In addition, each HTML page is not well structured and some people use the term *semi-structured data*. Moreover, much Web data is repeated (mirrored or copied) or very similar. Approximately 30% of Web pages are (near) duplicates [120, 723]. Semantic redundancy can be even larger.
- **Quality of data**: the Web can be considered as a new publishing medium. However, there is, in most cases, no editorial process. So, data can be false, invalid (for example, because it is too old), poorly written or, typically, with many errors from different sources (typos, grammatical mistakes, OCR errors, etc.). Preliminary studies show that the number of words with typos can range from 1 in 200 for common words to 1 in 3 for foreign surnames [588].

- **Heterogeneous data**: in addition to having to deal with multiple media types and hence with multiple formats, we also have different languages and, what is worse, different alphabets, some of them very large (for example, Chinese or Japanese Kanji).

Most of these problems (such as the variety of data types and poor data quality) are not solvable simply by software improvements. In fact, many of them will not change (and they should not, as in the case of language diversity!) because they are problems (also features) intrinsic to human nature.

The second class of problems are those faced by the user during the interaction with the retrieval system. There are basically two problems: (1) how to specify a query and (2) how to interpret the answer provided by the system. Without taking into account the semantic content of a document, it is not easy to precisely specify a query, unless it is very simple. Further, even if the user is able to pose the query, the answer might be a thousand Web pages. How do we handle a large answer? How do we rank the documents? How do we select the documents that really are of interest to the user? In addition, a single document could be large. How do we browse efficiently in large documents?

So, the overall challenge, in spite of the intrinsic problems posed by the Web, is to submit a good query to the search system, and obtain a manageable and relevant answer. Moreover, in practice we should try to achieve the latter goal even for poorly formulated queries. In the rest of this chapter, we use the term Web pages for HTML documents (HTML is described in Chapter 6). To denote all possible data types available on the Web, we use the term Web documents.

13.3 Characterizing the Web

13.3.1 Measuring the Web

Measuring the Internet and in particular the Web, is a difficult task due to its highly dynamic nature. Nowadays, there are more than 40 million computers in more than 200 countries connected to the Internet, many of them hosting Web servers. The estimated number of Web servers ranges from 2.4 million according to NetSizer [597] (November 1998) to over three million according to the Netcraft Web survey [596] (October 1998). This wide range might be explained when we consider that there are many Web sites that share the same Web server using virtual hosts, that not all of them are fully accessible, that many of them are provisional, etc. Other estimations were made by sampling 0.1% of all Internet numeric addresses obtaining about 2 million unique Web sites [619] or by counting domain names starting with `www` which in July 1998 were 780,000 according to the Internet Domain survey [599]. However, since not all Web servers have this prefix, the real number is even higher. Considering that in July 1998 the number of Internet hosts was estimated at 36.7 million [599], there is about one Web server per every ten computers connected to the

Internet. The characterization of the Web is a new task of the Web Consortium [797].

In two interesting articles, already (sadly) outdated, Bray [114] and Woodruff *et al.* [834] studied different statistical measures of the Web. The first study uses 11 million pages while the second uses 2.6 million pages, with both sets gathered in November 1995. Their characterization of Web pages is partially reproduced in the following paragraphs. A first question is how many different institutions (not Web servers) maintain Web data. This number is smaller than the number of servers, because many places have multiple servers. The exact number is unknown, but should be more than 40% of the number of Web servers (this percentage was the value back in 1995). The exact number of Web pages is also not known. Estimates at the beginning of 1998 ranged from 200 to 320 million, with 350 million as the best current estimate (July 1998 [91]). The latter study used 20,000 random queries based on a lexicon of 400,000 words extracted from Yahoo!. Those queries were submitted to four search engines and the union of all the answers covered about 70% of the Web. Figure 13.1 gives an approximation of how the number of Web servers and the number of pages have changed in recent years. Between 1997 and 1998, the size of the Web doubled in nine months and is currently growing at a rate of 20 million pages per month. On the other hand, it is estimated that the 30,000 largest Web sites (about 1% of the Web) account for approximately 50% of all Web pages [619].

The most popular formats for Web documents are HTML, followed by GIF and JPG (both for images), ASCII text, and Postscript, in that order. The most popular compression tools used are GNU zip, Zip, and Compress. What is a typical HTML page? First, most HTML pages are not standard, meaning that they do not comply with all the HTML specifications. In ad-

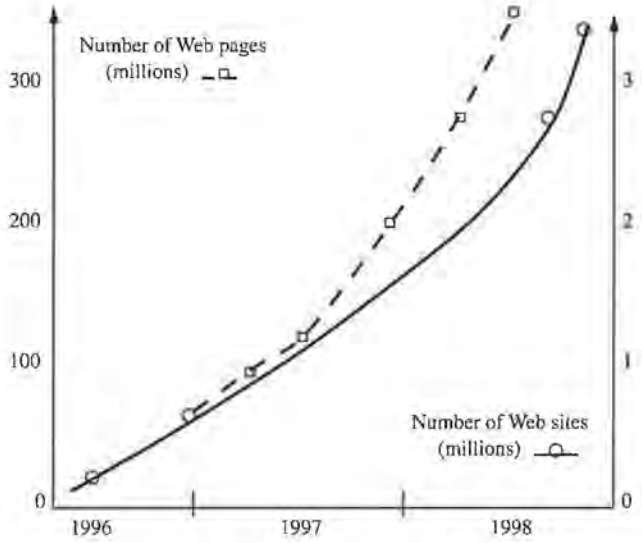


Figure 13.1 Approximate growth of the Web.

dition, although HTML is an instance of SGML, HTML documents seldom start with a formal document type definition. Second, they are small (around 5 Kbs on average with a median of 2 Kbs) and usually contain few images (between one and two on average with an average size of 14 Kb). The pages that have images use them for presentation issues such as colored bullets and lines. An average page has between five and 15 hyperlinks (more than eight links on average) and most of them are local (that is, they point to pages in their own Web server hierarchy). On average, no external server points to any given page (typically, there are only local links pointing to a given page). This is true even for home pages of Web sites. In fact, in 1995, around 80% of these home pages had fewer than ten external links pointing to each of them.

The top ten most referenced sites are Microsoft, Netscape, Yahoo!, and top US universities. In these cases we are talking about sites which are referenced by at least 100,000 places. On the other hand, the site with most links to outside sites is Yahoo!. In some sense, Yahoo! and other directories are the glue of the Web. Without them we would have many isolated portions (which is the case with many personal Web pages). If we assume that the average HTML page has 5 Kb and that there are 300 million Web pages, we have at least 1.5 terabytes of text. This is consistent with other measures obtained from search engines. Note that this volume does not include non-textual documents.

Regarding the languages used in Web pages, there have been three studies made. The first study was done by Funredes [637] from 1996 to 1998. It uses the AltaVista search engine and is based on searching different words in different languages. This technique might not be significant statistically, but the results are consistent with the second study which was carried out by Alis Technology [11] and is based on automatic software that can detect the language used. One of the goals of the study was to test such software (done in 8000 Web servers). The last study was done by OCLC in June of 1998 [619] by sampling Internet numeric addresses and using the SILC language identification software. Table 13.1 gives the percentages of Web pages written in each language (with the exception of the OCLC data that counts Web sites), as well as the number of people (millions) who speak the language. The variations for Japanese might be due to an inability to detect pages written in Kanji. Some languages, in particular Spanish and Portuguese, are growing fast and will surpass French in the near future. The total number of languages exceeds 100.

13.3.2 Modeling the Web

Can we model the document characteristics of the whole Web? Yes, as has already been discussed partially in Chapter 6. The Heaps' and Zipf's laws are also valid in the Web. In particular, the vocabulary grows faster (larger β) and the word distribution should be more biased (larger θ). However, there are no experiments on large Web collections to measure these parameters.

Language	Funredes (1998, %)	Alis Tech. (June 1997, %)	OCLC (June 1998, %)	Spoken by (millions)
English	76.4	82.3	71	450
Japanese	4.8	1.6	4	126
German	4.4	4.0	7	118
French	2.9	1.5	3	122
Spanish	2.6	1.1	3	266
Italian	1.5	0.8	1	63
Portuguese	0.8	0.7	2	175

Table 13.1 Languages of the Web.

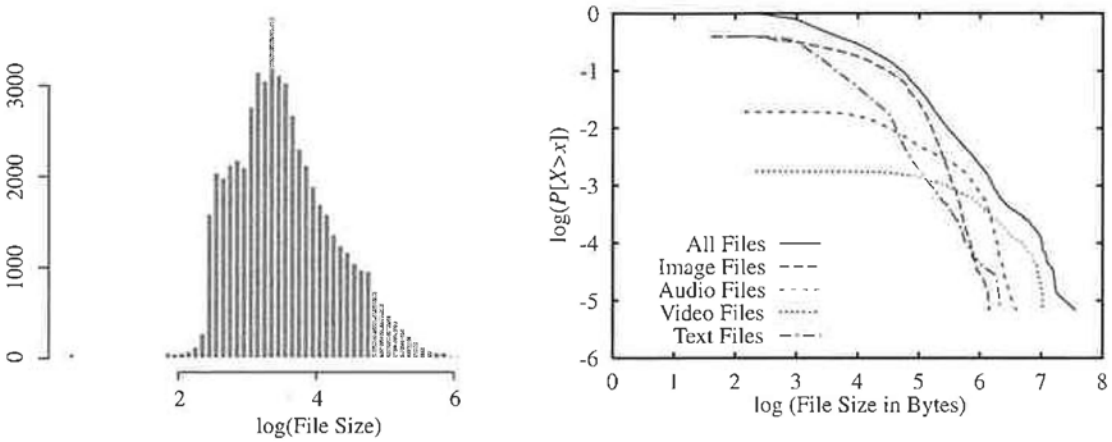


Figure 13.2 Left: Distribution for all file sizes (courtesy of M. Crovella, 1998). Right: Right tail distribution for different file types (from Crovella and Bestavros, 1996). All logarithms are in base 10.

An additional model is related to the distribution of document sizes. According to this model, the document sizes are self-similar [201], that is, they have a large variance (a similar behavior appears in Web traffic). This can be modeled by two different distributions. The main body of the distribution follows a logarithmic normal distribution, such that the probability of finding a document of size x bytes is given by

$$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp -(\ln x - \mu)^2/2\sigma^2$$

where the average (μ) and standard deviation (σ) are 9.357 and 1.318, respectively [59]. Figure 13.2 (left) shows the size distribution of the experimental data.

The right tail of the distribution is 'heavy-tailed.' That is, the majority of documents are small, but there is a non-trivial number of large documents. This is intuitive for image or video files, but it is also true for HTML pages. A good fit is obtained with the Pareto distribution

$$p(x) = \frac{\alpha k^{\alpha}}{x^{1+\alpha}}$$

where x is measured in bytes and k and α are parameters of the distribution [59] (see Figure 13.2 (right)). For text files, α is about 1.36, being smaller for images and other binary formats [201, 819]. Taking all Web documents into account, we get $\alpha = 1.1$ and $k = 9.3$ Kb [58]. That is, 9.3 Kb is the cut point between both distributions, and 93% of all the files have a size below this value. In fact, for less than 50 Kb, images are the typical files, from 50 to 300 Kb we have an increasing number of audio files, and over that to several megabytes, video files are more frequent. The parameters of these distributions were obtained from a sample of more than 54,000 Web pages requested by several users in a period of two months of 1995. Recent data collected in 1998 show that the size distributions have the same form, but parameters change [58]. Related information can be found on Web benchmarks such as WebSpec96 and the Sun/Inktomi Inkbench [395].

13.4 Search Engines

In this section we cover different architectures of retrieval systems that model the Web as a full-text database. One main difference between standard IR systems and the Web is that, in the Web, all queries must be answered without accessing the text (that is, only the indices are available). Otherwise, that would require either storing locally a copy of the Web pages (too expensive) or accessing remote pages through the network at query time (too slow). This difference has an impact on the indexing and searching algorithms, as well as on the query languages made available.

13.4.1 Centralized Architecture

Most search engines use a centralized crawler-indexer architecture. Crawlers are programs (software agents) that traverse the Web sending new or updated pages to a main server where they are indexed. Crawlers are also called robots, spiders, wanderers, walkers, and knowbots. In spite of their name, a crawler does not actually move to and run on remote machines, rather the crawler runs on a local system and sends requests to remote Web servers. The index is used in a centralized fashion to answer queries submitted from different places in the Web. Figure 13.3 shows the software architecture of a search engine based on the AltaVista architecture [17]. It has two parts: one that deals with the users,

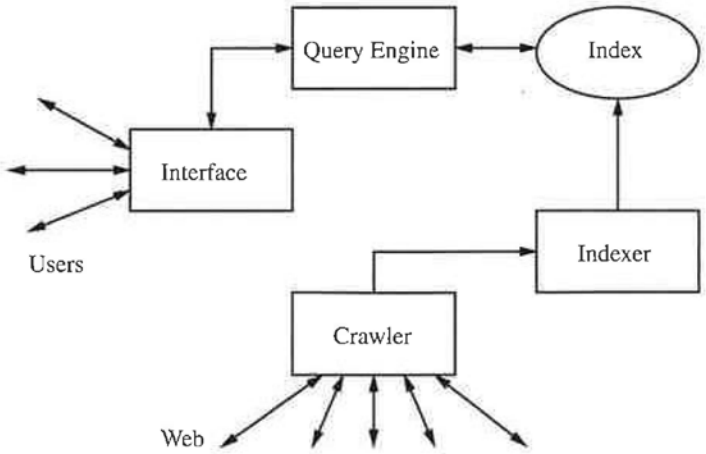


Figure 13.3 Typical crawler-indexer architecture.

consisting of the user interface and the query engine and another that consists of the crawler and indexer modules. In 1998, the overall AltaVista system was running on 20 multi-processor machines, all of them having more than 130 Gb of RAM and over 500 Gb of disk space. Only the query engine uses more than 75% of these resources.

The main problem faced by this architecture is the gathering of the data, because of the highly dynamic nature of the Web, the saturated communication links, and the high load at Web servers. Another important problem is the volume of the data. In fact, the crawler-indexer architecture may not be able to cope with Web growth in the near future. Particularly important is good load balancing between the different activities of a search engine, internally (answering queries and indexing) and externally (crawling).

The largest search engines, considering Web coverage in June 1998, were AltaVista [17], HotBot [380], Northern Light [608], and Excite [240], in that order. According to recent studies, these engines cover 28–55% [749] or 14–34% [490] of all Web pages, whose number was estimated at over 300 million in 1998. Table 13.2 lists the most important search engines and their estimated sizes along with their corresponding URLs. Beware that some search engines are powered by the same internal engine. For example, HotBot, GoTo, and Microsoft are powered by Inktomi [395] and Magellan by Excite's internal engine. Up to date information can be found in [749, 609].

Most search engines are based in the United States and focus on documents in English. Nevertheless, there are search engines specialized in different countries and/or languages, which are able, for instance, to query and retrieve documents written in Kanji (Chinese, Japanese, and Korean). Also there are search engines that take other approaches, like Ask Jeeves! which simulates an interview [34] or DirectHit [215] which ranks the Web pages in the answer in order of their popularity. We should also mention those search engines aimed at specific topics, for example the Search Broker [537] which allows us to search in many specific topics and DejaNews [212] which searches the USENET archives.

Search engine	URL	Web pages indexed
AltaVista	www.altavista.com	140
AOL Netfind	www.aol.com/netfind/	–
Excite	www.excite.com	55
Google	google.stanford.edu	25
GoTo	goto.com	–
HotBot	www.hotbot.com	110
Infoseek	www.infoseek.com	30
Lycos	www.lycos.com	30
Magellan	www.mckinley.com	55
Microsoft	search.msn.com	–
NorthernLight	www.nlsearch.com	67
WebCrawler	www.webcrawler.com	2

Table 13.2 URLs and estimated size (millions) of the largest search engines (May 1998).

There are also engines to retrieve specific Web pages such as personal or institutional home pages or specific objects such as electronic mail addresses, images, or software applets.

13.4.2 Distributed Architecture

There are several variants of the crawler-indexer architecture. Among them, the most important is Harvest [108]. Harvest uses a distributed architecture to gather and distribute data, which is more efficient than the crawler architecture. The main drawback is that Harvest requires the coordination of several Web servers.

The Harvest distributed approach addresses several of the problems of the crawler-indexer architecture, such as: (1) Web servers receive requests from different crawlers, increasing their load; (2) Web traffic increases because crawlers retrieve entire objects, but most of their content is discarded; and (3) information is gathered independently by each crawler, without coordination between all the search engines.

To solve these problems, Harvest introduces two main elements: gatherers and brokers. A gatherer collects and extracts indexing information from one or more Web servers. Gathering times are defined by the system and are periodic (i.e. there are harvesting times as the name of the system suggests). A broker provides the indexing mechanism and the query interface to the data gathered. Brokers retrieve information from one or more gatherers or other brokers, updating incrementally their indices. Depending on the configuration of gatherers and brokers, different improvements on server load and network traffic can be

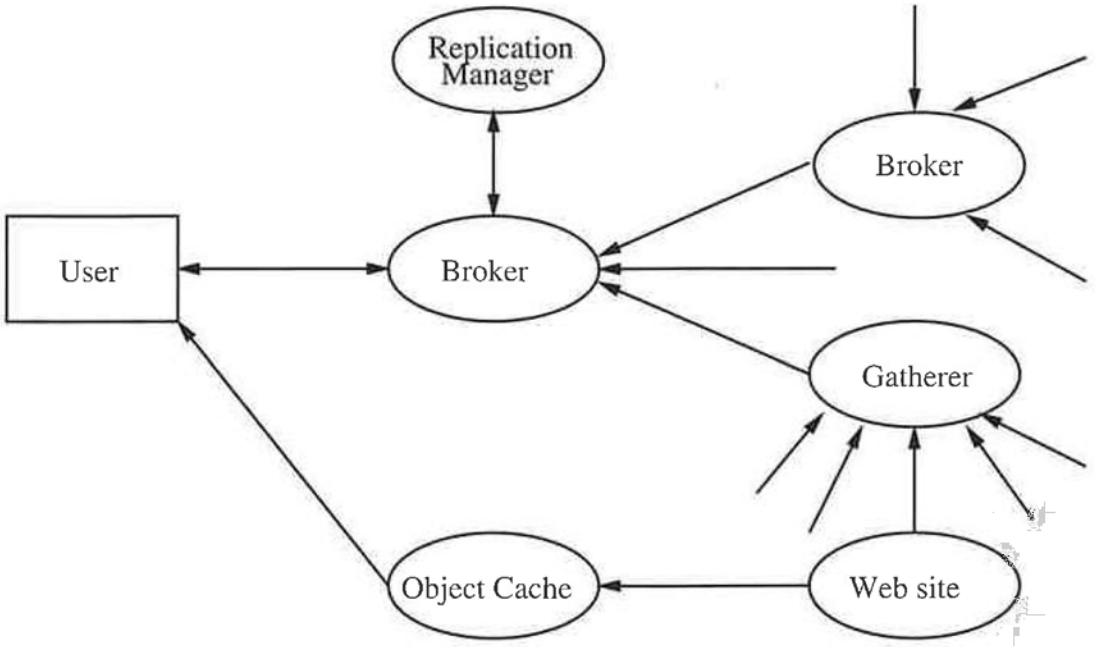


Figure 13.4 Harvest architecture.

achieved. For example, a gatherer can run on a Web server, generating no external traffic for that server. Also, a gatherer can send information to several brokers, avoiding work repetition. Brokers can also filter information and send it to other brokers. This design allows the sharing of work and information in a very flexible and generic manner. An example of the Harvest architecture is shown in Figure 13.4 [108].

One of the goals of Harvest is to build topic-specific brokers, focusing the index contents and avoiding many of the vocabulary and scaling problems of generic indices. Harvest includes a distinguished broker that allows other brokers to register information about gatherers and brokers. This is useful to search for an appropriate broker or gatherer when building a new system. The Harvest architecture also provides replicators and object caches. A replicator can be used to replicate servers, enhancing user-base scalability. For example, the registration broker can be replicated in different geographic regions to allow faster access. Replication can also be used to divide the gathering process between many Web servers. Finally, the object cache reduces network and server load, as well as response latency when accessing Web pages. More details on the system can be found in [108].

Currently, there are hundreds of Harvest applications on the Web (for example, the CIA, NASA, the US National Academy of Sciences, and the US Government Printing Office), as this software is on the public domain.‡ Netscape's Catalog Server is a commercial version of Harvest and Network Appliances' cache is a commercial version of the Harvest Cache.

‡ Information is available at `harvest.transarc.com`.

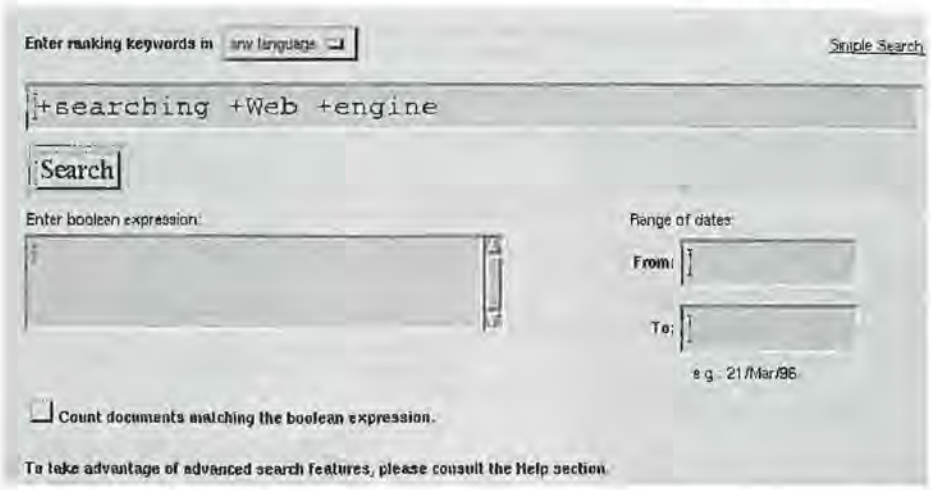


Figure 13.5 Query interface for complex queries in AltaVista.

13.4.3 User Interfaces

There are two important aspects of the user interface of search engines: the query interface and the answer interface (see also Chapter 10). The basic query interface is a box where one or more words can be typed. Although a user would expect that a given sequence of words represents the same query in all search engines, it does not. For example, in AltaVista a sequence of words is a reference to the union of all the Web pages having at least one of those words, while in HotBot it is a reference to the Web pages having all the words. Another problem is that the logical view of the text is not known, that is, some search engines use stopwords, some do stemming, and some are not case sensitive (see Chapter 7).

All search engines also provide a query interface for complex queries as well as a command language including Boolean operators and other features, such as phrase search, proximity search, and wild cards. Figures 13.5 and 13.6 show the query interfaces for complex queries for the three largest search engines. They provide several filtering functions. The results can be filtered by additional words that must be present or absent from the answer or in a particular field such as the URL or title, language, geographic region or Internet domain, date range, or inclusion of specific data types such as images or audio.

The answer usually consists of a list of the ten top ranked Web pages. Figure 13.7 shows the three top documents for the main four search engines for the query `searching and Web and engine`. Each entry in this list includes some information about the document it represents. Typically, the information includes the URL, size, the date when the page was indexed, and a couple of lines with its content (title plus first lines or selected headings or sentences). Some search engines allow the user to change the number of pages returned in the list and the amount of information per page, but in most cases this is fixed or limited to a few choices. The order of the list is typically by relevance, but sorting by URL or date is also available in some engines. In addition, most search engines also have an option to find documents similar to each Web page in the answer.

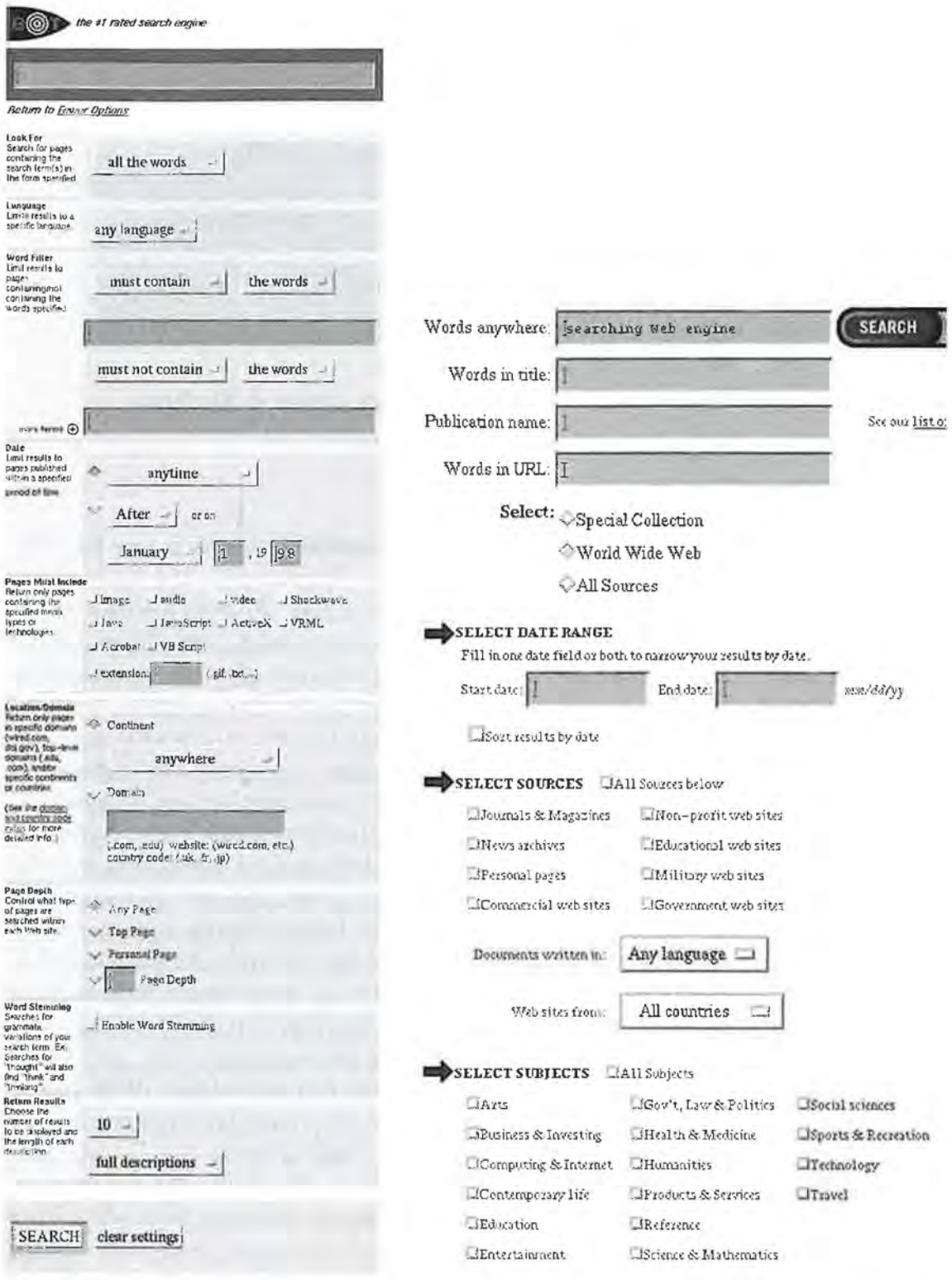


Figure 13.6 Query interface for complex queries for HotBot (left) and NorthernLight (right).



Figure 13.7 Output for the query `searching and Web and engine` for the four main search engines; from top to bottom: AltaVista, HotBot, NorthernLight, and Excite.