Incorporating String Search in a Hypertext System: User Interface and Signature File Design Issues

Christos Faloutsos*, Raymond Lee#,
Catherine Plaisant+ and Ben Shneiderman*

*Department of Computer Science
University of Maryland Institute for Advanced Computer Studies
Human-Computer Interaction Laboratory

#Department of Computer Science
Human-Computer Interaction Laboratory

+Human-Computer Interaction Laboratory

University of Maryland, College Park, MD 20742, USA

Abstract :

Hypertext systems provide an appealing mechanism for informally browsing databases by traversing selectable links. However, in many fact finding situations string search is an effective complement to browsing. This paper describes the application of the signature file method to achieve rapid and convenient string search in small personal computer hypertext environments. The method has been implemented in a prototype, as well as in a commercial product. Performance data for search times and storage space are presented from a commercial hypertext database. User interface issues are then discussed. Experience with the string search interface indicates that it was used sucessfully by novice users.

Address correspondence to: Christos Faloutsos
Computer Science Department, University of Maryland, College Park, MD 20742, USA
Tel: 1 (301) 454-1462 -- Fax: 1 (301) 454-8346 -- Email: christos@cs.umd.edu

Introduction

Early exploratory hypertext systems are giving way to numerous commercial systems (1), (2), (3), and the number of applications is growing rapidly. Hypertext and Hypermedia systems permit users to easily traverse, usually by simply touching or pointing and clicking, a large network of nodes containing text, graphics, and video images. Links guide the reader to appropriate nodes.

Simply following links can be effective for many tasks (4) as can exploring by way of indexes of node titles (5). However, these two mechanisms can be complemented by string search of the full text to identify nodes containing particular keywords, personal names, geographic information, chemicals, etc. String search has long been used in information retrieval over scientific abstracts, legal briefs, or medical drug descriptions, but hypertext traversal presents new requirements.

String search in hypertext has several challenges for designers. First, in traditional search situations, the users are seeking to identify a small subset of records that satisfy a search request. The goal is to collect a set of references that will be retrieved for further study. In hypertext situations, the readers are intensely involved in traversing the information and are reading and learning at every step. This distinction means that search results must come quickly and require low cognitive load or the reader's chain of thought will be disrupted. Therefore in hypertext situations, the search must be convenient and rapid, to enable readers to maintain concentration on the content material.

Second, with hypertext, the reader is traversing a large network and when a jump to a node derived from a string search is made, the reader can continue to follow further links. Since jumps can be disorienting, it is important that the readers have a clear sense of where they are jumping to and that they can easily return to where they have jumped from.

Third, in many hypertext situations there are some novel opportunities for search strategies. For example, since there is an existing path history and a current location, search results may be constrained to deal with nodes that have not yet been visited or are within a certain distance of the current location.

In this paper we focus on the design of a string search facility usable on a small personal computer. Such computers demand that the search method introduce only a small space overhead for indexes or program code. Despite the fact that more and more computers offer some graphic capabilities we tried here to design a user interface accommodating the still very large number of text only systems. A well designed text only string search interface is applicable to all PC's as well as standard terminals for dial-up and mainframe access. Most of the databases currently accessible are entirely textual anyway.

This scenario conveys the utility of string search in a hypertext environment : Imagine that you are reading a tourist guide database about major world cities in order to help you plan next summer's trip. You start by choosing something familiar such as the overview article about Europe and jump to read the article about Venice, the site of your most recent trip. The article reminds you of the beautiful piazzas and buildings and the fun and romance of the Venetian gondolas and canals. The writer compares them to the canals of Amsterdam, so you follow your interest and jump to the article on Amsterdam. Amsterdam sounds fine, but you want to travel to a more exotic destination than Europe. You choose string search and type "canal" which produces this list of articles:

* AMSTERDAM
BANGKOK
PANAMA CANAL
SUEZ CANAL
* VENICE

The asterisks next to Amsterdam and Venice indicate that you have read articles on those cities already. The Suez and Panama Canals are not what you are looking for, so you choose the article on Bangkok. You find that Bangkok is known as the Venice of the East and become fascinated by the description of the city and its Buddhist temples. Links in the article lead you to tourist agencies and airlines that deal with Thailand. You can begin to plan next summer's trip.

Our research has focused on adding a string search mechanism to our seven-year old hypertext system called Hyperties (4). It runs on IBM PCs and PS/2s as well as SUN 3 and SUN 4 workstations (6) but we will only discuss the initial PC implementation here. This paper describes Hyperties, our addition of string search to the user interface, the signature file strategy we used and performance results on a real database. We close with suggestions for further research.

1- Description of Hyperties

Hyperties (Hypertext Interactive Encyclopedia System) allows users to easily traverse a database of articles by merely pointing at highlighted words or images in context (see Figure 1). An article can be one or several screens long and include pictures. Hyperties is composed of two parts: the browser and the authoring tool.

CHIM, the photograph   Page 1 of 1

David Seymour (1911-1956), known to his friends as "Chim", applieds skilled hand, warm heart, and perceptive eye to photographing poignant and dramatic events of the 20th century. Chim's photographs have appeared in many **magazine articles**, <u>**books**</u>, and **exhibits**, and a number of memorials have been dedicated to his memory. His photographs of famous **personalities** of the 20th century are widely known.

This exhibit **"David Seymour 'Chim' The Early Years 1933-1939** will remain on view at the **International Center of Photography** through January 1987.

BOOKS WITH DAVID SEYMOUR'S PHOTOS - a description of books containing photographs taken by David Seymour

FULL ARTICLE ON BOOKS WITH DAVID SEYMOUR'S PHOTOS

NEXT PAGE   **RETURN TO CAPA, ROBERT**   **EXTRA**

Figure 1: This screen representation shows the first page of an article. The bold items are selectable. The cursor is now resting on the word **books** and the brief definition for the referenced article appears on the lower part of the screen.

Browser:

The browser allows individuals to traverse the database of articles created by the authoring tool. Some phrases within the text or regions in a picture are embedded menu items that may be selected through a variety of devices: touchscreen, mouse or jump cursors using the arrow keys of the keyboard. When an item is selected, a brief definition is displayed at the bottom of the screen. Users may continue reading the current article or choose to ask for details about the selected topic. A path is maintained of all the articles that the reader has traversed. At any point in the session, the reader may return to the article last visited, and continue up to the introductory article. The reader may also branch directly to intermediate nodes in the path by using the history facility (see below).

The "EXTRA" screen contains facilities to help users browse the database. There are currently four facilities provided: table of contents, index, history, and search. They are displayed as a stack of folders (see Figure 2). By selecting a tab menu item, the desired facility can be entered. The TABLE OF CONTENTS menu item allows a jump to a special article that has been previously specified by the author of the database and which acts like the table of contents of a book. The index facility provides an alphabetic listing of titles of all the articles in the database. From this list, the user may select articles for viewing. The history facility provides a jump facility to any node in the path of traversed articles. The search facility allows a string search of the database and is the main focus of this paper. A tab item will not appear in the "EXTRA" screen if that facility is not available for the current database. For example, if the signature files do not exist, then the search tab will be blank. The history tab may be initially blank, but will appear as soon as an article path exists.

```
TABLE OF CONTENTS        INDEX        HISTORY        SEARCH

ALPHABETIC INDEX (62 articles)                          Page 2 of 5
The introductory article is: INTRODUCTION

  * COGNETICS CORPORATION
    COMMANDS
    COMPLETE ARTICLE
    CONTENTS
    DAVID SEYMOUR
    DISPLAY A BRIEF DESCRIPTION
  * EGA GRAPHICS
    ELECTRONIC ARTS
  * ENTERTAINMENT SOFTWARE
    GO TO FULL ARTICLE
    GRAPHICS
    HARCOURT BRACE JOHANOVICH
    HENRI CARTIER-BRESSON

  TURN TO:_A_B_C_D_E_G_I_M_N_O_P_R_S_T_V_W_Y

 BACK PAGE    NEXT PAGE    RETURN TO "SMITHSONIAN MUSEUM"    QUIT
```

Figure 2: The "EXTRA" screen, now showing the index.
The stars mark the articles already seen.

Authoring Tool:

The authoring tool allows individuals to create and edit a database of articles. Links may be created between articles to build a network of references. The authoring system provides an editor in which the articles may be written, and maintains cross-reference tables and synonym lists. Embedded formatting commands are supported (such as skip lines, new paragraphs, and indents). Other features such as importation and exportation of articles are also available. Text, pictures and videodisc images may all be incorporated into articles.

2- Description of the Signature Files

This section describes the string search mechanism, which is based on the signature files approach. We were looking for a method with the following characteristics:

1- Small space overhead. Since many of the databases were designed to fit on a 360 Kb floppy diskette, space was at a premium.

2- Satisfactory search speed, for databases of the above size.

The text retrieval methods that have appeared in the literature form the following large classes: full text scanning, inversion, signature files and clustering. The first three classes are suitable for boolean queries; clustering (7) provides ways of grouping similar documents together. Clustering is suitable for the so-called "keyword searches", i.e., "find the documents that are about 'data', 'retrieval' and 'information', or as close to that as possible." Since we are interested in boolean queries, we concentrate on the first three classes.

<u>Full text scanning</u>

Given a search pattern, the whole database is scanned until the qualifying documents are discovered and returned to the user. The method requires no space overhead and minimal effort on insertions and updates, but is slow on large databases unless specialized search hardware is used (8).

<u>Inversion</u>

This method uses an index. An entry of the index consists of a word (or stem or root) along with a list of pointers to the qualifying documents. It is probably the most popular approach in commercial systems (STAIRS (9); MEDLARS, ORBIT, LEXIS (7), etc.). The main advantage is its retrieval speed. The main disadvantages are that it may require large storage overhead for the index (50%-300% of the initial file size, according to Haskin (10)) and that insertions of new documents require expensive updates of the index. For example, if the index is organized as a B-tree, some nodes of the B-tree will eventually have to be split and rewritten when a new word is inserted.

<u>Signature file</u>

The documents are stored sequentially in the "text file". Their signatures (hash-code bit patterns) are stored (usually) sequentially in the "signature file". When a query arrives, the signature file is scanned and many non-qualifying documents are discarded. The text files are checked, so that the *false drops* are discarded. False drops (or false positives or false alarms) are documents which are initially flagged as qualifying but which do not really match the query. Note that the signature methods never introduce false negatives (or false dismissals), i.e. articles which do contain the search string, but fail the signature test. The secondary search performs full text scanning on the retrieved documents, to discard the false drops. The method is faster than full text scanning but usually slower than inversion on large databases. It requires much smaller space overhead than inversion (approximately 10-20% of the text file (11)) and can handle insertions easily.

Signature files typically use superimposed coding to create the signature of a document. A brief description of the method follows; more details are in (12). For performance reasons that will be explained later, each document (in the case of Hyperties each article) is divided into "logical blocks", that is, pieces of text that contain a constant number D of distinct, non-common words. Each such word yields a "word signature", which is a bit pattern of size F, with m bits set to "1" while the rest are "0" (see Figure 3.1). F and m are design parameters.

The word signatures are ORed together to form the block signature. Block signatures are concatenated, to form the document signature. The *m* bit positions to be set to "1" by each word are determined by hash functions. Using successive, overlapping triplets as hashing elements, queries on parts of words can be handled easily (13).

Word	Signature
free	001 000 110 010
text	000 010 101 001
Block signature	001 010 111 011

Figure 3.1: Illustration of the superimposed coding method (For a logical block of D=2 words only. The signature size F is 12 bits, and m=4 bits per word.)

Searching for a word is handled by creating the signature of the word and by examining each block signature for "1"'s in those positions that the signature of the search words has a "1". More complicated Boolean queries can be handled in a similar way. In fact, conjunctive (AND) queries introduce fewer false drops than single word queries.

Figure 3.2 illustrates the file structure used. In addition to the text file (in this case the article file) and the signature file, we need the "pointer file", with pointers to the beginning of the documents.

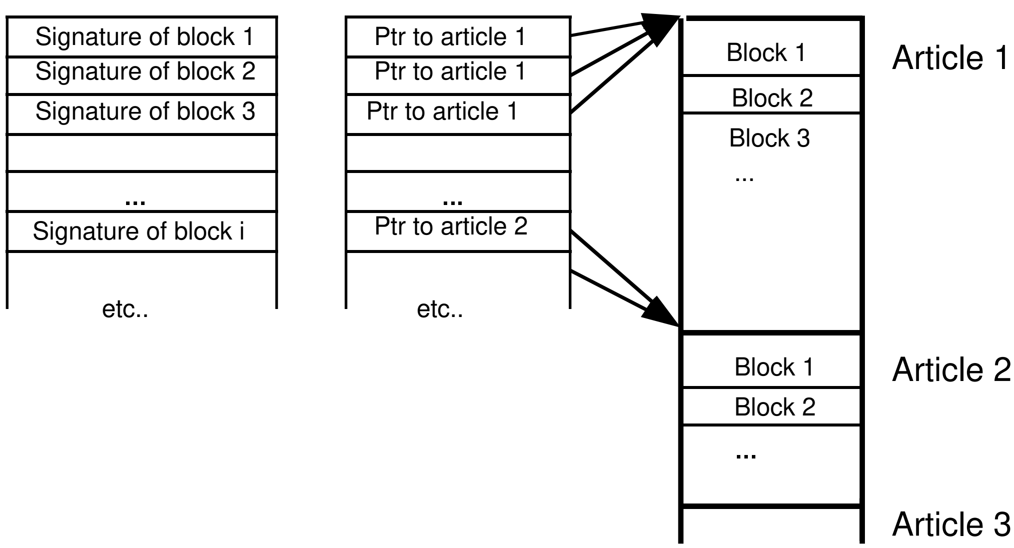

Figure 3.2 The signature files are searched, then the pointers are followed to retrieve the text files, and false drops are eliminated by direct string search.

The signature file is an F by N binary matrix (N is the number of blocks). Previous analysis showed that, for a given value of F, the optimal value of m is such that this matrix contains "1"'s with probability 50% (14). This is the reason that documents have to be divided into logical blocks: without logical blocks, a long document would have a signature full of "1"'s, and it would always create a false drop.

Practical aspects of the PC version:

The signature files are built by an independent utility routine that stands apart from the Hyperties browser. The INSERT program is started and provided a path to a database. INSERT steps through each article to create signatures for each article. All the signature information is placed in the two files sig.s and sig.p. Sig.s contains the signatures for the logical blocks and sig.p contains the pointers to those logical blocks. Search performance will improve by combining these files into one; this will be one of the first changes in future versions. The authoring tool's formatting commands are removed from the input of INSERT as well as all nonalphanumeric characters. The user may also specify a list of common words for a stop list in the file COMMON.TIE. These words are ignored by the INSERT program and so cannot be searched for later.

Performance results

Performance tests were conducted in the *Hypertext Hands-On!* database (2), containing 248 articles, with a total size of the text file of about 200K bytes after compression (about 400K before compression). The pointer file occupied 7,440 bytes and the signature file was 48K bytes long. The tests were run on an IBM-PS/2 Model 60 running DOS 3.3. Table 1 gives several search examples and the associated response times.

Single word unsuccessful searches of long words (e.g. appendicitis, faloutsos, signature) enjoyed the same response time of approximately 3 seconds: apparently, this is the time to scan the signature file - no false drops that needed any extra searching appeared. The search for non-existent short words (e.g. qwx) created many false drops, exactly because we set only one bit for each triplet in the query word - i.e. the smaller the word, the fewer bits we look for, the more false drops that will have to be retrieved and discarded, and therefore the longer it will take.

Search words of medium frequency (i.e., 5-12 hits like the string "retrieval") require more than 3 seconds, because the qualifying documents have to be scanned. The additional response time depends on the response set size as well as on the the query word length (the shorter the word, the more false drops, as we explained before).

High frequency query words require much longer response times; however, we expect that these queries will not be issued frequently, at least by experienced users. Two-word queries require 10 seconds or more, depending on the boolean connective, the size of the response set, etc. For programming simplicity, the initial version required two scans of the signature files, but in future versions we expect that a one-pass scanning algorithm will cut the search time to approximately the time for a single word search.

	SEARCHED WORD(s)	RESPONSE TIME (in sec's)	HITS
LOW FREQ.			
	appendicitis	3	0
	faloutsos	3	0
	signature	3	1
	qwx	32	0
	qwxz	28	0
	qwxzvt	7	0
	qwxzfklj	4	0
	abcde	12	0
	abcdef	7	0
	abcdefg	5	0
MED. FREQ.			
	halasz	8	7
	retrieval	6	11
	human	18	20
	factor	15	12
	why	33	12
HIGH FREQ.			
	data	27	80
	user	26	72
	hyper	24	140
	hypertext	22	130
TWO-WORD QUERIES			
	alan turing	12	0
	human factor	11	7
	human&factor	13	7
	human|factor	22	25

Table 1: Response Time Table

Discussion

Signature files are well suited for a small PC environment because they are fast for the corresponding databases and they are space-efficient. To take an extreme case, the string search can run even on an IBM PC with a low density 360K drive; excluding 93K for the Hyperties code, this leaves 267K on the floppy for the database and its index. Signature files for a text database this size are approximately 40K. Since the block size is 1K, and the signature file is (more or less) stored sequentially, access requires only 1 random disk access and 39 sequential ones to scan the whole signature file. Later a few random disk accesses are needed to read the pointer file and the actual articles.

For larger databases, the Sequential Signature Files we have described will be slow. We have studied and implemented the Bit-Sliced Signature Files (15) and the Frame-Sliced Signature Files (16), which achieve an order of magnitude speed up, at the expense of some programming complexity. Additional signature file methods include the two-level signature files (17, 18), the partitioned signature files (19), and the S-tree (20). These methods and other possibilities have not yet been incorporated in Hyperties.

3- User Interface Design

Hyperties has been designed to be used by novice users. We had to find a good compromise providing the power of the string search while keeping the simplicity of query formulation and the ease of traversal. Additionally, the small-PC environment contraints as well as the desire to design a system usable from any terminal and over a telephone line led us to the choice of a text only interface.

Description of the interface:

As mentioned in section 1, the string search appears as one facility in the "EXTRA" screen (Figure 2). The search tab item may be selected to enter the search facility. From here, the user may enter a search string and begin a search, return to the previous article, or quit Hyperties. The user may also choose another tab item to enter the table of contents, index, or history facility.

Possible formats of the string to be searched:

The user wishing to initiate a search must first enter a search string (Figure. 4). There are four search string formats which may be entered:

(1) *word*
(2) *word1 word2* (single blank separation)
(3) *word1&word2*
(4) *word1|word2*

Words are contiguous alphanumeric characters. Nonalphanumeric strings may not be searched for. Format (1) indicates that the user wishes to search for the single word *word*. Format (2) will search for articles containing *word1* immediately followed by *word2*. Formats (3) and (4) search for articles which contain both and either word, respectively. Leading substring matches are made with each word in the search string. So, the string "human factor" would match "human factors", "Humans factor", and "humanistic factories". Also note that no distinction is made between upper and lower case characters. While the underlying software supports complex boolean expressions the two word boolean limitation is imposed because larger combinations would complicate the interface; this limitation greatly simplifies training, while it still seems to satisfy most user's needs.

When the SEARCH STRING command is selected, entry of the string is expected. The last search string entered is presented for editing. The backspace key deletes the last character of the string and function key 9 will erase the entire string. The escape key aborts the entry and retains the last string entered. To prevent string format errors, entry is very restricted. A blank, '&', or '|' is not allowed to be typed as the beginning of a search string. Also, once a blank, '&', or '|' has been typed, only alphanumerics are allowed. Only two word combinations are accepted. Thus, formatting errors are eliminated.

When the search string has been entered, the string is passed to the search algorithm to begin the search. After the qualifying signatures have been identified, full-text scanning of the corresponding documents is performed, to eliminate the false drops (see section 2.)

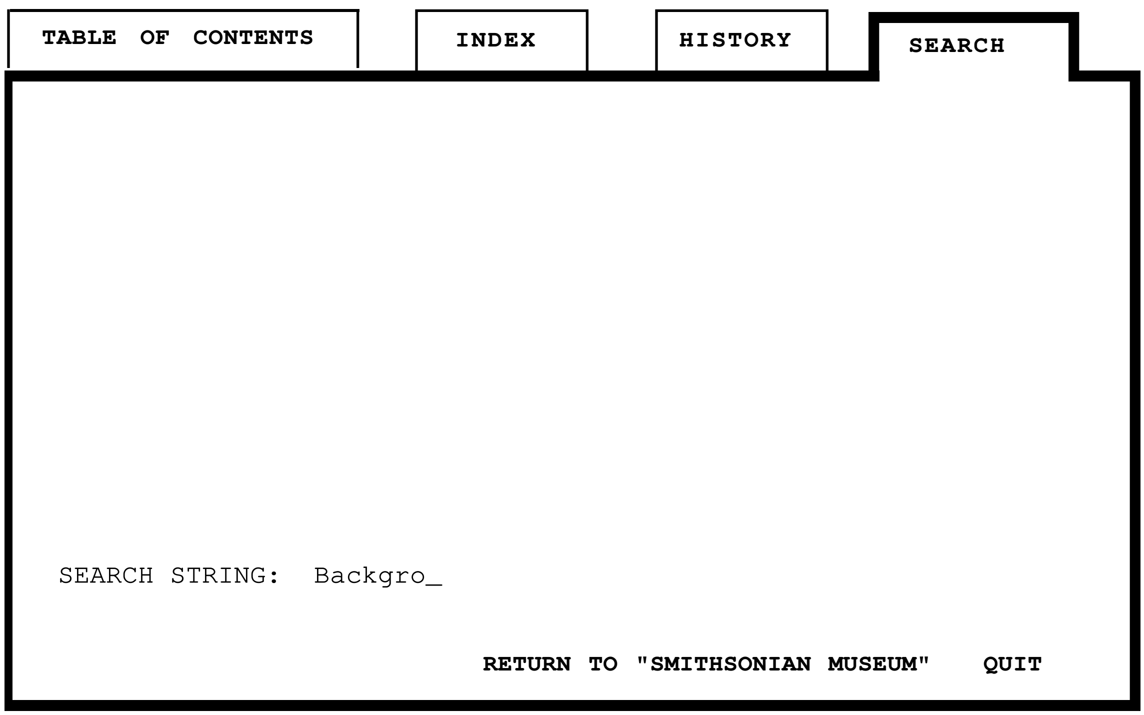

Figure 4 : The "Extra" screen, now showing the search folder. The user is entering the string to be searched.

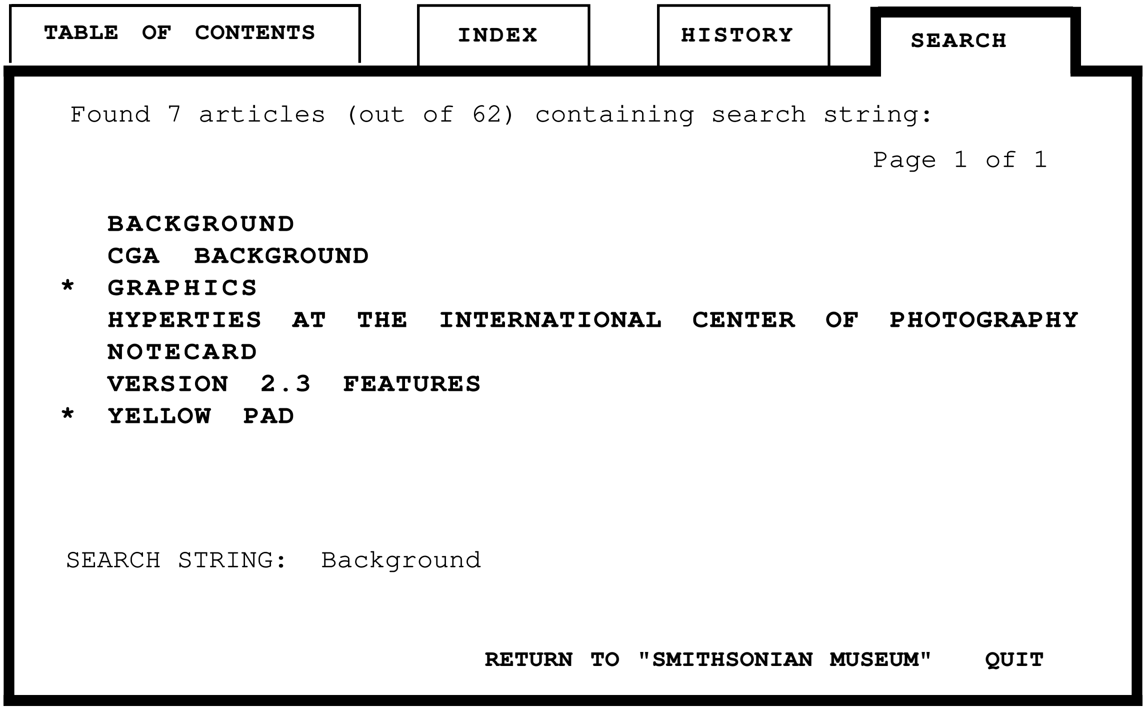

Figure 5 : The "Extra" screen, now showing the result of the string search.

Browsing the result of the search:

The final result article list is displayed on the screen (Figure 5.) This list is presented and manipulated in the same way as the general index: If the list is longer than one screen, the reader may page up and page down the list. An article may be selected to see its short definition and then the entire article may be seen. During a Hyperties session, the article list from the last search is retained so the reader may search for a string, go see an article from the list, and then return to the search facility to immediately see the list again. The search result list marks the articles that have already been visited by an asterisk and the default cursor position is on the first article not seen.

When the reader wishes to see the article, the first page of the article in which the search string appears is displayed. For two-word boolean queries (i.e., string formats (3) and (4)), the page displayed is the first page containing either word. Also, all occurrences of the search string(s) in the article are highlighted (e.g. in boldface). Users can customize the highlight style used. When a searched string is also a link, the string is highlighted as a link on the screen. Because the amount of text on a page is small there are generally only a few links per page and all highlighted words can be scanned rapidly.

Discussion

Advantage of the list of articles:

A typical design of the string-search interface (for example in Hypercard) is the direct jump to the first node containing an occurrence of the string as soon as it is found. From there the user can jump to the next occurrence, and so on. The hypertext system doesn't provide any estimate of the number of hits or how long it will take to scan the nodes. Previously read nodes will also redisplayed systematically.

In the Hyperties string search, we chose to provide the full search result list of article titles. This gives users a feeling of the screening efficacy of their string search and of the effort involved to read through all the articles. For example: if a search returns 200 articles, the user may prefer to narrow the search instead of reading the articles. Of course it would be preferable to prompt users of the quickly growing number of hits to let them take action before the end of a long search.

Another benefit of the list comes from the fact that each article has a meaningful title and not just a code number. Therefore it is possible to ignore irrelevant articles. For example: while making a search on the space shuttle using "space" and "shuttle", users may decide not to look at the article "How to go to National airport". Hyperties also provides a short description for each article, which gives to users another chance to avoid irrelevant articles.

Consistency:

An effort has been made to keep a similar user interface for all the list manipulation situations. The general index of articles and the search result list, as well as the history list (see Section 1 - "Browser", are manipulated similarly enough to reduce the user's learning effort. On the other hand, specific additions have been made to speed-up the screening of the string search result list: the searched string is located in the article, and automatic movement of the default cursor position in the list (in the keyboard version) allows rapid looping over all the articles selected by the search.

Different orderings of the search result list:

Currently the search result list of articles is displayed in alphabetical order to preserve consistency with the way the index lists articles. An alternative would be to present articles ordered by number of occurrences of the string in the article. In this strategy each article title is then followed by the number of occurrences of the string. Unfortunately this strategy slows down the speed of the search because the necessary checking of the whole article (instead of a simple reading until the first occurrence). We did not implement this ordering on the PC to keep the speed up.

Another strategy would consist of specifying in which part of the article the first occurrence of the string has been found. In the case of Hyperties, the string can be found in the title, the definition, or the content of the article itself. The articles whose titles contain the string are possibly more relevant. We plan to try this strategy.

Highlighting:

When consulting an article selected by the string search, it is convenient to see where the string was found in order to check if the article is relevant. Having the exact location of the string in the page reduces the discomfort of landing in the middle of a text ("Where should I start reading?"). In the current version, we used a special highlighting for the string (boldface). This solution is convenient as long as the text itself doesn't contain any similar highlighting. Since supplementary screen markings might cause clutter, a temporary flashing of the string or of a bracket marking the area could attract attention. Although flashing is usually not pleasant, it's brief use easily attracts the reader's attention without cluttering the screen. This would allow temporary highlighting of a string as being found and then it would be visible as a link.

Experience with users

In a pilot study for his dissertation study, Liebscher (21) had five undergraduate participants search for the answers to a number of questions in the Hyperties database Hypertext Hands On! Participants had for distinct access methods:

- Alphabetical Index (article titles, names, and subject terms),
- Subject index (Hierarchically arranged),
- String Search,
- Browsing the hypertext network.

All participants reported liking the string search facility, particularly when they knew little or nothing about the topic being searched. When given a choice of access methods, most participants selected string search as their first choice of access method. All participants believed that the 2 word limit for string searching imposed by the Hyperties system was adequate for the small specialized database they searched. However two participants commented that for searching a large or multidisciplinary database, the two word limitation could be inadequate. All participants thought the highlight feature was very useful and used it consistently. A complete experiment is currently underway.

4- Suggestions for further research

As a result of using the string search facility, several possible modifications and additions have surfaced. The difficulty lies in designing a user-interface that smoothly integrates additional functions without adding too much complexity for the user.

Operations on search result lists:

A user may want to retain or print the search result lists for future review. New search result lists could be created through the application of set operations, such as intersection and union. Additional operations might include:

- deleting articles the reader rejected
- re-ordering of the list by the reader
- saving and restoring the search results to and from a library.

Relevance weighting and ranked output:

The relevance of the retrieved documents to the user's query can be assessed using query-to-document similarity functions, e.g. the cosine function., and the documents can be presented to the user starting from the "most relevant first" (22) (7).

Restricted search domain and neighborhood searches:

Especially if the database is very large, it will be helpful to reduce the search domain. For example a search might be limited to articles on a previous search result list, or the articles already visited or not visited. The domain could also be articles within two (or any number) reference links from a specified article.

String search within an article:

This facility would allow the user to jump to another location in the current article. An initial implementation might allow a jump to the first occurrence of the search string either forward or backward from the current location. Of course the need for string search inside an article grows as the size of the articles increase.

String specification by pointing to the current text:

While reading an article, one might wish to point at a word or a string and ask for all the other articles containing that same string. Several issues still need to be resolved including how would logical conditions like AND and OR be specified and should the reader be "moved" automatically to the search screen.

Incorporation of string search in the authoring tool:

Currently, string search is limited to the browser. However, a string search mechanism could be useful when creating articles. If the search was modified to allow search and string replacement, the search facility could be very useful. For example, a term not selectable could be made selectable throughout all the articles.

Speeding the search:

If there are too many initial hits, users might not desire to conduct the secondary search to remove false positives. This approach may be desirable if the user needs only an approximate list of articles matching the search string. It may also be possible to speed up the string search by merging the signature files, sig.s and sig.p. A single signature file would require fewer accesses. Finally, improved performance could be achieved by using the Bit-Sliced, or even better the Frame-Sliced Signature Files, as mentioned before.

Conclusions

Hypertext applications will grow rapidly because of their ease of use and the power they offer for organizing and accessing information. While the essence of hypertext is to follow links, other strategies using tables of contents, indexes to node names, keyword indexes, and string search are useful complements. While many systems have implemented a simple search mechanism that only finds the next occurrence of a string, we felt that a more powerful but yet simple facility would be beneficial. Our string search produces a list of article titles that contain the search string. This initial implementation of signature files was successfully applied in several widely distributed projects such as ACM's *Hypertext on Hypertext* , Addison-Wesley's *Hypertext Hands-on!* , and many projects inside corporations and Universities. Search times are reasonable, but there is room for improvement to the search strategy and to the file organization.

There are many interesting problems in connection with hypertext search mechanisms. We plan to examine advanced features, such as neighborhood searches, similar node searches, relevance weighting, and more powerful boolean and set operations.

Finally, although this article is about string search in hypertext, the more commonly known and used query language systems have a lot to gain from a hypertext front end that would allow the user to browse directly from the result of a search without having to perform complex queries every time. For example, when a long query finally gives you the scientific paper you sought, you should be able to follow the links to the others papers referred to by that paper without having to use the query language again.

References

1. CONKLIN, J., (1987). Hypertext: An introduction and survey. *IEEE Computer,* Sept 1987, 17-41.

2. SHNEIDERMAN, B. and KEARSLEY, G., (1989b). *Hypertext Hands-On!*, Addison-Wesley Publ., Reading, MA, USA.

3. HALASZ, F., (1988). Reflections on Notecards: seven issues for the next generation of Hypermedia systems, *Communication of the ACM, 31,*7, (July 1988), 836-852.

4. SHNEIDERMAN, B., (1989a). Reflections on authoring, editing and managing hypertext. In, Barrett, Ed. (editor) *The Society of Text*, MIT Press, Cambridge, MA.

5. WANG, X., LIEBSCHER, P. , AND MARCHIONINI, G., (Jan. 1988), Improving information seeking performance in hypertext: role of display format and search strategy, Technical Report CS-TR-2006, CAR-TR-353, Dept. of Computer Sciences, University of Maryland.

6. SHNEIDERMAN, B., PLAISANT, C., BOTAFOGO, R., HOPKINS, D., WEILAND, W., Visual engagement and low cognitive load in browsing hypertext. Technical Report CAR-TR-494, CS-TR-2433, Dept. of Computer Sciences, University of Maryland, College Park, MD 20742, USA.

7. SALTON, G., MCGILL, M.J., (1983). *Introduction to Modern Information Retrieval,* McGraw-Hill, New York.

8. HOLLAAR, L. A., (1979). Text retrieval computers, *IEEE Computer*, 12, 3, (March 1979), 40-50.

9. IBM, IBM System/370 (OS/VS), Storage and Information Retrieval System /Vertical Storage (STAIRS/VS), IBM World Trade Corporation.

10. HASKIN, R. L., (1981). Special-purpose processors for text retrieval, *Database Engineering,* 4, 1, 16-29.

11. FALOUTSOS, C. and CHRISTODOULAKIS, S., (1984). Signature files: An access method for documents and its analytical performance evaluation, *ACM Trans. on Office Information Systems, 2*, 4, (Oct. 1984), 267-288.

12. FALOUTSOS, C., (1985). Access methods for text, *ACM Computing Surveys 17*, 1 (March 1985), 49-74.

13. CHRISTODOULAKIS, S., FALOUTSOS, C., (1984). Design considerations for a message file server, *IEEE Trans. on Software Engineering SE-10*, 2, (March 1984), 201-210.

14. STIASSNY, S., (Feb. 1960). Mathematical analysis of various superimposed coding methods, *American Documentation 11*, 2, 155-169.

15. FALOUTSOS, C., and CHAN, R., (1988). Fast text access methods for optical and large magnetic disks: Designs and performance comparison, *Proc. 14th International Conf. on VLDB ,* (Aug. 1988), 280-293.

16. LIN, Z., FALOUTSOS, C., (December 1988). Frame-sliced signature methods for text retrieval, CS 2146 - UMIACS TR-88-88. Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742, USA.

17. SACKS-DAVIS, R., RAMAMOHANARAO, K., (1983). A Two Level Superimposed Coding Scheme for Partial Match Retrieval, *Information Systems 8*, 4, 273-280.

18. SACKS-DAVIS,R. ,KENT, A. ,RAMAMOHANARAO, K., (1987). Multikey Access Methods Based on Superimposed Coding Techniques, *ACM Trans. on Database Systems (TODS) 12,* 4, 655-696.

19. LEE, D.L., LENG, C.-W., (April 1989). Partitioned Signature File: Designs and Performance Evaluation, *ACM Trans. on Information Systems (TOIS) 7*, 2, 158-180.

20. DEPPISCH, U., (Sept 1986). S-tree: A Dynamic Balanced Signature Index for Office Retrieval, *Proc. of ACM "Research and Development in Information Retrieval"*, Pisa, Italy, Sept. 8-10, 1986, 77-87

21. LIEBSCHER, P. , (1990), Information seeking in Hypertext: effects of multiple access methods on the usert/system/task interaction. Dissertation proposal, Center for Library and Information Science, Hornbake Library, University of Maryland, College Park, MD 20742, USA.

22. SALTON, G., (1971), *The SMART Retrieval System - Experiment in Automatic Document Processing*. Prentice Hall, Englewood Cliffs N.J.