ELECTRONIC PUBLISHING, VOL . 1(1), 45–53 (APRIL 1988)

# Linking and searching within hypertext

P. J. BROWN

*Computing Laboratory*
*The University*
*Canterbury*
*Kent, CT2 7NF*

**SUMMARY**

**The Find command is a familiar mechanism for travelling round linear documents. In hypertext documents, on the other hand, the primary method of travel is by means of built-in links. The paper considers how a Find command can be integrated into a hypertext system. There are two main issues:**

- **What does it mean to search a hypertext document?**
- **Can the two methods of travel be integrated in such a way that the user does not become disoriented?**

KEY WORDS Hypertext Find Command Searching in Hypertext Guide

Hypertext represents an attractive and new way of storing information in a computer. Like any new approach, this leads to the challenge of integrating the new practices with the old ones, and our aim here is to investigate one instance of this. Before going into details, however, we shall give a brief introduction to the nature of hypertext as an aid to readers who are unfamiliar with the concept.

## THE NATURE OF HYPERTEXT

Although hypertext is new in the sense that only recently have systems become widely available, the original idea dates from 1945 (see Bush[1]). Most of the existing techniques were pioneered in the sixties and seventies, particularly by Engelbart[2] and Nelson[3].

The essential aim of hypertext is to store documents in the computer in such a manner that a reader at a terminal can explore a document in any way he wishes. This is achieved by allowing the author to place *links* in documents, which give the reader an easy way to get from one point to another.

There are essentially two types of link. The first is a *hierarchical* link, and this can be used to present material at successively greater levels of detail. For example the author may initially present a document in the form of an overview. The reader then indicates the part(s) for which he requires more detail, and the computer then displays the required detail. (This is achieved by following a hierarchical link set by the author, with the more detailed material at the destination of the link.) It is important that the reader has a simple way of interacting with a document, and this is frequently done by having ‘buttons’ on the screen—perhaps embedded within the document. A button might, for

0894–3982/88/010045–09$05.00

*Received 1 November 1987*
*Revised 5 December 1987*

instance, be labelled '**More**' or '**Example**', and such buttons are usually activated using a mouse. The reader gradually refines the level of detail, sometimes perhaps 'undoing' an action if the level of detail gets too great, until the displayed material is tailored to his needs.

The second type of link is used for cross-referencing. One point in a document can contain an electronic cross-reference to a second point, either in the same document or another one. Thus where a book may say 'See page 123' a hypertext document has an electronic link that will actually take the interested reader to the appropriate place. Links of this sort have a wide utility, covering footnotes, citations, indexes, annotation, etc.

The essence of hypertext, therefore, is that it is not a linear text, but a directed graph connecting together material, hopefully in a well-structured way. Hypertext documents may represent a large corpus of knowledge about a particular topic. Indeed Trigg and Weiser[4], following Nelson[5], look forward to all the world's scientific papers being linked together in one massive hypertext structure.

A hypertext document may contain media other than text. In particular it will usually involve an integrated mixture of text and graphics; more ambitious systems, such as the Intermedia[6] project at Brown University, cater for sound, video, etc. (Strictly speaking the term *hypermedia* should be used to describe such systems but in practice the word 'hypertext' is usually used to encompass hypermedia.)

## THE FIND COMMAND

Before the advent of hypertext, a major method for users to travel round a document was the *Find* command. In its simplest form, this searches the current document for a given string or pattern and stops when it reaches the first match. Subsequently the user can, if he wishes, re-use the same Find command to find subsequent matches of the same string.

The Find command continues to flourish, even in hypertext environments, and almost all software for creating or viewing documents supports it.

The Find command provides an entirely different kind of link to the hypertext link. The hypertext link is an explicit link pre-set by the author (though most hypertext systems allow ordinary readers to become authors and to create their own links). The Find command represents an implicit link that is computed dynamically according to some criterion chosen by the reader (i.e. the string he chooses to search for).

The most important characteristic of the Find command is that it is a totally unstructured linking mechanism since it can go from any point in a document to any other point. We shall call these *unstructured links*, since they are totally unlike the *structural links* normally present in hypertext.

(As we have presented it above, the Find command is a 'find next occurrence' command. If, however, it is a 'find all occurrences' command it has a further difference from a hypertext link in that it is a one-to-many link rather than a one-to-one link. We are not concerned, in this paper, with this extra level of complication.)

## COMBINING STRUCTURAL AND UNSTRUCTURED LINKS

The purpose of this paper is to explore how a hypertext system can provide coherent support for both structural and unstructured links. Conklin[7], in his survey of hypertext,

characterizes the two types of link and suggests it may be possible to ‘allow design of a hypertext interface which is consistent across all link-tracing-like activities’.

Brewer[8], when discussing retrieval software, with special reference to data on CD-ROM, expresses similar views. He compares browsing through a database, which is done in a structured way, and searching, with its unstructured links. He characterises browsing as going from *where* to *what* (—you know where you are in the database and you want to know what is there) and searching as going from *what* to *where* (—you believe that you know what you want and wish to find where in the database it is). A good user interface, Brewer says, should combine both: ‘As CD-ROM matures, many users are finding that a combination of searching and browsing works best. Both approaches have their place. In fact, they can be intermixed well’.

Brewer’s comment, however, is made with regard to databases and these may not reflect all the facilities of general hypertext. Nevertheless the user’s aspirations will be similar.

### The disadvantages of combined linking

Even if users want both structural and unstructured links, it is worth pausing to consider if it is really sensible to provide them.

A parallel can be drawn between hypertext links, whether structural or unstructured, and ‘gotos’ in programming languages. Using this parallel, an unstructured link is a particularly bad kind of goto: it leaps around a document, brushing aside any structural boundaries defined by the author. Users of programming languages thought they wanted both structured control (e.g. a *while* statement) and gotos. However they were eventually persuaded that the superficial freedom given by the goto had a disastrous cost in the longer term, since program maintenance dwarfs all other costs. The cost comes, of course, when people try to read a program and get hopelessly lost. Gotos have therefore been restricted (e.g. to *break* and *return* statements) or abolished.

As with programs, the cost of maintenance of a hypertext document will doubtless dwarf the original design cost. Indeed the problem of authors getting lost is already emerging as the biggest problem in hypertext systems, and this problem is particularly acute for authors who maintain documents written by others long in the past. Is it fair to draw the parallel between gotos and hypertext links? Even if it is, the unstructured links generated by a Find command do at least have the advantage that they are transient: they may indeed cause the perpetrator of the Find command to get lost in the structuring, but they do not get preserved as part of the document; thus other users will not be affected. Paradoxically, the dangers are much greater with the ‘structural’ links that form a permanent part of the document. If the ‘structure’ defined by these links is like spaghetti, then the cost to all who try to read or maintain the document is indeed high.

A second danger with unstructured links is their inaccuracy. If the user asks for X to be found and is told it is not present in the document, he cannot be sure that the document does not effectively contain references to X using synonyms or alternative spellings. More generally, the saying ‘one can only extract something from a database if one knows what is there’ contains much wisdom. It is much better, it can be argued, to force the users to explore the document in a structured way, following links set by a skilful and knowledgeable author, than to provide an easy-to-walk but unstructured path (e.g. a Find command) that in fact leads to a minefield. This argument has merit, but its weakness is

that it assumes the author will be able to provide a linkage structure to suit all readers. This is not realistic even for highly competent authors. From experience with paper documents, very few authors are highly competent, and this holds *a fortiori* for hypertext documents, since they are more complex than linear paper documents. It is thus unrealistic to expect all readers to be satisfied by the links that the author has built into a document.

Generally, therefore, the arguments against allowing searching because of its undesirable unstructured links have been overridden by the needs of the real world and by the characteristics of human nature. The issue is not *whether* to allow searching, but *how* to minimize its dangers.

## UNITS OF HYPERTEXT

Before proceeding to more details, it is necessary to consider the units into which hypertext is broken. We shall concentrate on units as the user sees them, i.e. what comes up in windows on his screen. Most hypertext systems support (at least) two levels of unit: *micro-units* and *macro-units*. A micro-unit is the content of a node of the directed graph that represents the hypertext; by definition, a micro-unit involves no sub-structure. Micro-units are combined into macro-units which represent a coherent body of information. Macro-units have different names in different systems, e.g. 'file', 'stack', 'document'. In this paper we will use the familiar term 'document' to stand for the somewhat ugly term 'macro-unit'.

The hypertext user 'loads' one or more documents and then explores the micro-units within them. In many systems, documents can themselves contain links to other documents (or can be linked to other documents by a separate mechanism that rides on top of them all). However when the user moves to another document by following such a link he is normally made aware that he is entering a new context, i.e. that a new document has been loaded, either in addition to or in place of the current one.

In most hypertext systems a micro-unit is a windowful of information, and is called a 'frame', 'page' or 'card'. This applies, for example, to ZOG[9], NoteCards[10], and HyperCard[11]. In Guide[12], on the other hand, a window contains a document. Normally Guide's window is not large enough to display the whole document so there is a focussing mechanism (actually a scroll-bar) to allow the user to display the part he wants to view.

This difference of approach has some small effects on the searching mechanisms, but the main point is that in all systems the user will have loaded one or more documents and will have actually viewed a sub-set of the micro-units within that document.

## THE TWO STAGES

A Find command essentially works in two stages:

(1) *Searching for the destination.* A search is made to find the first (or next) occurrence of the target string or pattern.
(2) *Effecting the goto.* Assuming the destination has been found and also assuming it is not within the part of the document that is currently in view, the document must be re-focussed so that the destination is in view.

Finally, of course, the destination needs to be highlighted in some way, so that the reader can immediately pick it out, but such matters are not a concern of the current paper.

With relation to the theme of this paper, the integration of the Find command with hypertext linking, the two stages are very different. In the first, the issue is that searching needs to be adapted to fit a new non-linear world. In the second, the task is clear-cut, but the problem is to accomplish it without disorienting the user.

## STAGE 1: SEARCHING

Since this paper is concerned with hypertext rather than with matching mechanisms, we shall not worry whether the Find command performs a full-text search or a keyword search, nor whether it involves intermediate dialogues with the user (e.g. 'Define the pattern to be found', . . . , 'This matches 1,000 items, do you want to proceed?'). All we require is that the Find command identifies a point in the hypertext document that should be gone to. Our main concern is what it means to search a hypertext document, and this is the topic of the current Section of the paper. The objective is not to present a 'best' approach. Instead it is to survey the possible options, all of which are valuable in certain applications. The following are issues that arise:

### *(1) Scope of search*

The search may cover any of the following:

- the micro-units that have already been viewed, i.e. the world as the user has seen it. This is an unlikely choice as the very purpose of a search is to find new territory.
- the documents currently loaded.
- the documents currently loaded, together with all documents to which these documents link. In the extreme, following Trigg and Weiser's vision, it covers the whole of the world's literature!
- some subset of the above. Clearly if a document has a strong tree structure it is possible to confine the search to a sub-tree.

### *(2) Dynamic creation*

Some hypertext systems allow the material at the destination of a link to be created dynamically. More generally, when the user selects a link, a program may be run which, *inter alia*, generates some text (and/or pictures) that appears as the destination of the link. As a specific instance of this, a hypertext tutorial on file systems may include a link that leads to a listing of all the current user's files. This listing may be generated by executing a program (e.g. *ls* in a UNIX environment). A search of such a document might take any one of the following approaches:

- ignore dynamically generated material.
- only cover material that has already been dynamically generated.
- follow all links and thus *cause* material to be dynamically generated. (This might be tragic, in our example above, if some of the links illustrated file deletion by actually deleting some files and then generating a new listing!)

*(3) Multiple-media*

In a hypermedia system, it is possible to have a different searching mechanism for each medium. Such searches must confine their scope to the designated medium. A textual search must not, for example, come up with a chance match which in fact is a bit-pattern in the middle of a picture. Side-effects should be avoided: for example if the search passes over some sounds it should not make the sounds.

*(4) Structural boundaries*

It may be that the user can load two different micro-units so that they come one after the other on the screen. Assuming the first ends with the word X and the next begins with the word Y, will a search for XY identify this pair? A suggested answer is 'no', since the structural boundary presumably indicates a logical separation.

*(5) Searching the structure*

In a hypertext system it may be required to search the link structure itself, rather than the text at the destination of links. For example, links may have associated names, e.g. 'More' or 'Critique of X by Y', and may have properties, e.g. one property might be a program that defines how to generate the material that appears as the destination of the link. The search should, on option, cover such material. Equally desirable, but difficult to implement —especially in a friendly manner —are 'structural' searches such as 'Find all micro-units with more than four outward links' or 'Find all inward links to a given region of a document'. Halasz[13] gives an excellent discussion of such matters. (In structural searches, incidentally, the objective may be to view the hypertext structure rather than to go to a particular piece of text. Hence the 'Stage 2', as we describe it below, might be absent or curtailed.)

*(6) Ordering*

We have emphasised that the user must know what is being searched, and (1) to (5) above have raised some potentially tricky issues. There is a subsidiary issue of ordering. A user might issue commands such as 'Find the first X, find the next X, then find the first Y in the rest of the document'. The meaning of such commands is obvious in a linear document, but in a hypertext document the search order may be largely arbitrary —it certainly may not be obvious to the user. (In acute cases, such as where a hypertext document is a cyclic directed graph, the ordering will not be obvious to anyone; indeed it may be a challenging implementation problem!)

*(7) Taking advantage of extra knowledge*

Finally, there is an especially positive aspect of searching in hypertext: the links represent extra knowledge about the material, and this can be exploited in a search. For instance, within a hierarchical structure, a match at a high level may be 'better' than one at a deep level. In a non-hierarchical structure a match within a micro-unit that is the object of many inward links may be 'better' than one in a micro-unit with only one such link.

### Summary of the first stage

To summarise, searching a hypertext document involves a large array of choices. A challenge to implementors is, on the one hand, to provide as much choice as possible but, on the other, to avoid burdening the user with a mass of options.

Default options should be chosen to reflect the aim that the searching mechanism should be integrated with normal hypertext links. For instance, if a hypertext system is such that users regard a document as a 'world', and regard a link to another document as a change to a separate world, then the natural scope for a search would be the current document; however, by means of a pop-up menu or the like, the user should be given the opportunity to extend the search into any or all of the worlds that are linked with the current world.

## STAGE 2: EFFECTING THE GOTO

We now come to the second stage of the linking process. If the hypertext system allows the search to range outside the currently loaded documents, the first task may be to load the document in which the destination lies; this loading should be a straightforward matter, and, to the user, the loading process should appear to be identical, irrespective of whether it was caused by a search or by following a hypertext link to another document.

Having done this loading, if necessary, the next task is to focus the document on the destination. If the destination is in view, there is, of course, nothing to do. Otherwise re-focussing turns out to be a problem that merits considerable attention. To explore this, we shall effect a complete change of narrative style and consider the travels of a hypothetical motorist in Hyperland.

## THE HYPERLAND MOTORIST

Hyperland is a fine place for travelling. The roads are good, and there are no speed limits. Signposting is excellent: whenever a motorist comes to a junction there are clear directions on were the different roads lead. Moreover the Hyperland motorist has several advantages over ordinary motorists. Firstly he has a backtracking facility: if he decides he previously took a wrong turning he can ask to be whisked back to any junction he previously visited. Secondly the road system has some remarkable link roads: these connect together places hundreds of miles apart, yet the link roads are trivial in length.

In spite of all these existing advantages, the Hyperland motorist is to be supplied a further facility. He can issue a command such as 'Find me a church', and a supersonic eagle will immediately swoop down, pluck up the motorist and his car, and then drop them down (gently) beside a church. The motorist can subsequently instruct the eagle to 'Find the next church' and the process will be repeated. Alternatively the motorist can ask for another target, such as a Post Office.

Before their import into Hyperland, the eagles had been operating in other lands, where they provided almost the only means of travel. In such lands they blindfolded the motorist during the flights, since the motorist had no interest in the route. However motorists in Hyperland, with its excellent road system, want to know what road they are on when they are dropped by the eagle. There have been harrowing experiences of

motorists hopelessly lost because they were dumped on roads that they were unable to connect with the territory that they knew.

One possible solution to this problem is to provide the motorist with a road-map. The eagle, when he drops the motorist, can then point out the current position on the map. However cartographers in Hyperland have found it difficult to produce road-maps that the average motorist can understand. The link-roads are what cause the problems.

Other aids have been tried, such as giving the motorists a list of the roads that lead to and from the place they are dropped. These certainly help, but motorists continue to get lost, because they cannot cope with the sudden changes of environment.

The solution finally adopted has been to take care to relate each new environment to a point previously visited by the motorist. The eagle has therefore been instructed to do the following:

(1) first take the motorist to the point that has already been visited and is as close as possible to the new destination.
(2) plan a good path along the roads to get from the point chosen in (1) to the destination. This is easiest when there is some hierarchical structure to the road system, since it is then more obvious what a 'good' path is.
(3) pick up the motorist and take him along these roads. At each junction the motorist is put down and shown the road that the eagle is going to take him along.
(4) finally drop the motorist in the normal way at his destination.

This approach has the disadvantage that travel is slower, but the motorist does have the opportunity to follow the eagle's path and relate his destination to known territory. Of course, if the path to the destination involves a lot of junctions the motorist may forget some of them. However even this is not too big a problem since the Hyperland motorist can take advantage of its backtracking facilities. (If a path consists of a really huge number of junctions, fifty say, then this approach certainly does break down. However the fault then arguably lies with the designer who produced such a immensely complex road system.)

The next improvement will be to allow the motorist to control the speed of the eagle, and the amount of time that he stops at each junction. Impatient motorists can get to their destination quickly, without worrying too much about the route, whereas more thoughtful motorists can follow their course precisely.

## IMPLEMENTATION IN GUIDE

The device that prevents Hyperland motorists getting lost can be applied to any hypertext system, though it works best in systems that encourage a strong hierarchical backbone to a document. The goto caused by a Find command can be converted to the equivalent user actions to follow hypertext links to get to the same destination. These user actions can then be played back to the user so that he can follow the path.

This has been implemented in the UNIX version of Guide. In Guide all hypertext links are embedded as 'buttons' inside a document. When the user employs the Find command he is shown an animated sequence of button selections to reach the destination. (This sequence in fact only consists of selections of Guide *replace-buttons*, which define the hierarchical backbone of a Guide document.) During this sequence the normal feed-back is shown, e.g. a button is highlighted when it is selected. In Guide's case the

animated sequence can also involve scrolling as well as button-selection, since Guide is based on a linear scroll rather than a 'page' or 'card'. In particular this happens when the destination is at the end of a long replacement, covering several screenfuls of information. The overall effect is that the destination is related to previously visited material. Perhaps more important, *an unstructured link is automatically converted to a sequence of structural links*, and shown to the user in these terms. Thus the user is presented with a uniform linking mechanism.

## CONCLUSION

Integrating a Find command into a hypertext system is a worthwhile aim, in spite of its risks. It involves two problems: firstly, extending the scope and nature of the searching mechanism to cover hypertext structures; secondly, trying to prevent the reader from getting disoriented through apparently random structural leaps. This paper has elaborated the design decisions involved in tackling the first problem, and has presented an approach to solving the second.

## ACKNOWLEDGEMENTS

My colleagues at the University of Kent and at Office Workstations Ltd. provided valuable comments for this paper; especial thanks are due to Karen Mahony.

## REFERENCES

1. V. Bush, 'As we may think', *Atlantic Monthly*, **176**, 101–108 (1945).
2. D. C. Engelbart and W K English, 'A research center for augmenting human intellect', *Proceedings of the 1968 FJCC*, Montvale, N. J. 395–410 (1968).
3. T. H. Nelson, 'Getting it out of our system', *Information retrieval: a critical review*, Washington, DC. (1967).
4. R. H. Trigg and M. Weiser, 'TEXTNET: a network-based approach to text handling', *ACM Transactions on Office Systems*, **4** (1), 1–23 (1986).
5. T. H. Nelson, *Literary machines*, T. H. Nelson, Swarthmore, Pa., 1981.
6. N. Yankelovich, N. Meyrowitz, and A. van Dam, 'Reading and writing the electronic book', *IEEE Computer*, **18** (10), 15–30 (1985).
7. J. Conklin, 'Hypertext: introduction and survey', *IEEE Computer*, **20** (9), 17–41 (1987).
8. B. Brewer, 'The look and feel … and sound of the user interface', *CD-ROM Review*, **2** (3), 26–30 (1987).
9. G. Robertson, D. McCracken, and A. Newell, 'The ZOG approach to man/machine communication', *International Journal of Man-Machine Studies*, **14**, 461–488 (1981).
10. F. G. Halasz, T. P. Moran, and R. H. Trigg, 'NoteCards in a Nutshell', *CHI+GI 1987 Conference Proceedings (* special edition of *SIGCHI Bulletin)* 45–52 (1987).
11. Apple Computer Inc., *Macintosh HyperCard user's guide*. 1987.
12. P. J. Brown, 'Interactive documentation', *Software—Practice and Experience*, **16** (3), 291–299 (1986).
13. F. G. Halasz, 'Reflections on NoteCards: seven issues for the next generation of hypermedia systems', *Proceedings of Hypertext 87 Conference*, Chapel Hill, N.C. (1987).