From:
Jones D.M. and Winder R.
'People and Computers IV'
Cambridge University Press
1988. pp 421–435.

A Comparison of Hypertext, Scrolling and Folding as Mechanisms for Program Browsing

Andrew F. Monk, Paul Walsh & Alan J. Dix

Departments of Psychology and Computer Science, University of York, York YO1 5DD, U.K.

Hypertext removes some of the constraints of conventional linear text by providing mechanisms for physically realizing the conceptual links between related sections of material. This research examines the use of a hypertext browser with a literate program. A literate program has a sequential structure, in that it is divided into sections presented in a particular order, and a hierarchical structure, in that some sections 'use' other sections.

Two experiments are described which compare the performance of users browsing the same program presented either as a linear or hypertext structure. In Experiment 1 one group used a hypertext browser the other two scrolling and folding browsers. The hypertext browser is shown to be inferior to the scrolling browser under these particular circumstances. In a second experiment two further groups of users were tested, one of which was provided with an overview of the hypertext structure. This manipulation removed the disadvantage demonstrated in Experiment 1. It is concluded that while

hypertext presents many new opportunities to the interface designer, it also raises new problems. In particular, the importance of providing an overview or map of the hypertext structure is demonstrated.

Keywords: Hypertext, scrolling, folding, browsing, literate programming.

1. Introduction

This paper presents two experiments which explore the use of hypertext. A hypertext system for program browsing is compared with two alternatives schemes. A typical hypertext system is made up of screens or windows containing 'hot spots'. Selecting one of these hot spots causes some associated screen or window to be displayed. For example, one screen may contain a diagram with labels describing its components. The labels are hot spots. Moving the mouse cursor to one and clicking causes a screen of text expanding the description to be displayed. This screen may also contain keywords which are also hot spots.

Hypertext has been used for teaching (Hammond & Allinson [1988]; Yankelovich, Meyrowitz & Dam [1985]), authoring (Halasz, Moran & Trigg [1987]; Trigg & Weiser [1986]), and programming (Kuo et al. [1986]). There is also the multi-purpose information system ZOG (Akscyn, McCracken & Yoder [1987]). The salient feature of these applications is that they present the user with a physical realization of the conceptual links which can only be symbolized in conventional text. For example, this paper has a hierarchical structure as indicated by the section headings and subheadings, however, that conceptual structure is symbolically rather than physically realized in its printed form.

With printed material and most text editors the underlying object manipulated by the user has a serial or sequential structure. Thus, page one is followed by page two, line one is followed by line two and so on. Hypertext permits the use of hierarchies or any other form of connected network to access related material within the system. Further, if the links between screens can be of different types then it is possible to impose alternative structures on an object. For example, the multimedia system described by Yankelovich *et al* (Yankelovich, Meyrowitz & Dam [1985]). envisages an arrangement where the teacher provided a large data base of linked information. The student may then build a new set of links onto the same screens providing a novel perspective onto the material.

While a number of the papers referenced above discuss the advantages and disadvantages of hypertext systems there has been little systematic empirical work comparing the usability of hypertext with the alternatives. This is unfortunate, not because it is possible to do the definitive experiment showing that hypertext is better or worse than some alternative, but because systematic empirical study is the most effective way of gaining insights about how to design good systems.

2. The Vehicle for Experimentation

2.1. Literate Programming

The experiments to be described in the next section evaluate a hypertext browsing system in comparison with two alternative browsing schemes. Each browser operates on the same material. With the hypertext system the user is constrained to follow a network of links representing one way the information might be structured. In the other two browsers the user operates on a more conventional sequential information structure. The problem area chosen was browsers for literate programs (Knuth [1984]).

Knuth's idea is that computer programs should be regarded as works of literature, in which the software author strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding. To do this a literate program has two additional layers of structure above the procedural and data flow structures provided by the programming language. First there is a sequential structure. The program is divided into numbered sections and the order of these sections is chosen to explain the program as a simple expository sequence. The second layer of structure is hierarchical. The program is divided into sections which may 'use' other sections. This gives the author a mechanism, additional to the procedural structure of the programming language, to conceal inessential detail at each level of exposition (for a detailed discussion and evaluation of literate programming see (Thimbleby [1986]). A literate program is convenient for our purposes as it can have these different structures imposed on it.

In the experiments described in the next section users are asked to answer questions about a program. Each section in that program ends with a statement of in what section the current section is used, which sections it uses and, where global variables and constants are declared, a 'see also' referring to all the other sections where global variables and constants are declared. This indexing information forms the basis of the hypertext structure used in these experiments. These section numbers are 'hot spots'. Users can move from one section to another which uses it or to a section which it uses by selecting one of these numbers. Alternatively, users can move from one section to another referred to in the 'see also' information. This hypertext browser is compared with two browsers

based on a sequential model of the documented program as it might be printed out. One uses scrolling to view the document, the other folding.

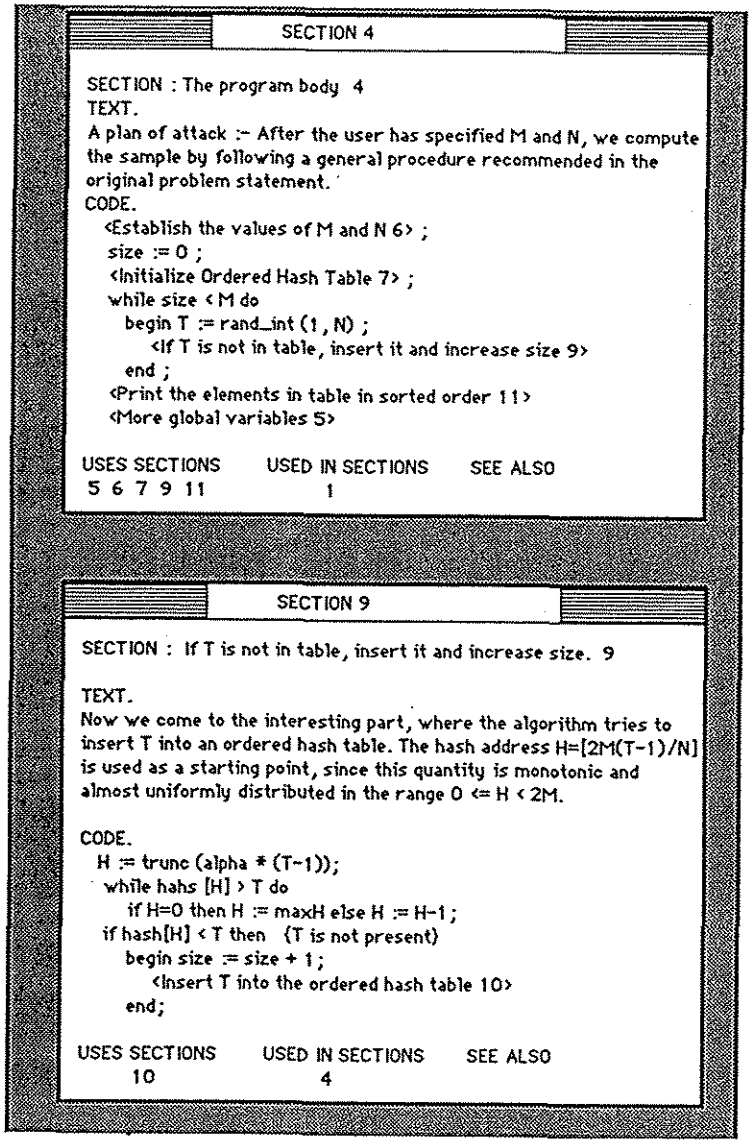

Figure 1. The Hypertext Browser. The user has opened section 9 by clicking on that number in the indexing information for section 4

2.2. The Hypertext Browser.

Figure 1 illustrates what a user of the Hypertext Browser might see at some point in time. Positioning the mouse pointer on a number in the indexing information for a section and clicking has the effect of overwriting the text in the other window with the chosen section. Since clicking in one window always causes the material in the other window to be replaced by a new choice there are never more than two sections displayed and there is no need for the concept, commonly found in multi-window environments, of an 'active window'. If the section selected is already displayed then the system beeps to indicate that no change will be visible. Each window is 24 lines deep and so would accommodate the largest section.

2.3. The Scrolling Browser.

The implied user model of the object being inspected in this case is the more conventional sequential one. The user is to imagine that they are inspecting a continuous document. The text in the Scrolling Browser is displayed in a single window (Figure 2). To make it comparable with the Hypertext Browser the window is large enough to display two sections (48 lines). Along the top of this window there is a thumb bar. Clicking in the thumb bar will scroll to the appropriate point in the program. In addition there is a second small window containing an up and a down arrow at the bottom right of the screen which we shall refer to as the scroll box. Clicking on the up arrow scrolls up 30 lines i.e., the text moves up relative to the window. Clicking on the down arrow causes the text to move down. Thus the user has a choice of navigation strategy with the Scrolling Browser. They can either (a) position the mouse pointer in a portion of the scroll box and click to scroll forwards or backwards, or (b) they can position the mouse pointer at a point on the thumb bar and click to scroll forwards to that portion of the text.

2.4. The Folding Browser.

The Folding Browser also has an implied user model which is a single sequential document. Initially there is a single window containing the twelve section titles and below them a grey portion of free space. The user browses sections by positioning the mouse pointer on the section title and clicking. The section is partly unfolded to reveal holophrasts for the 'Text' and the 'Code' subsections, plus the indexing information for that section. The action of unfolding causes the text window to encroach upon the free space slightly. The user can continue the operation of unfolding information by pointing to either the text or the code holophrast and clicking. This unfolds the chosen subsection

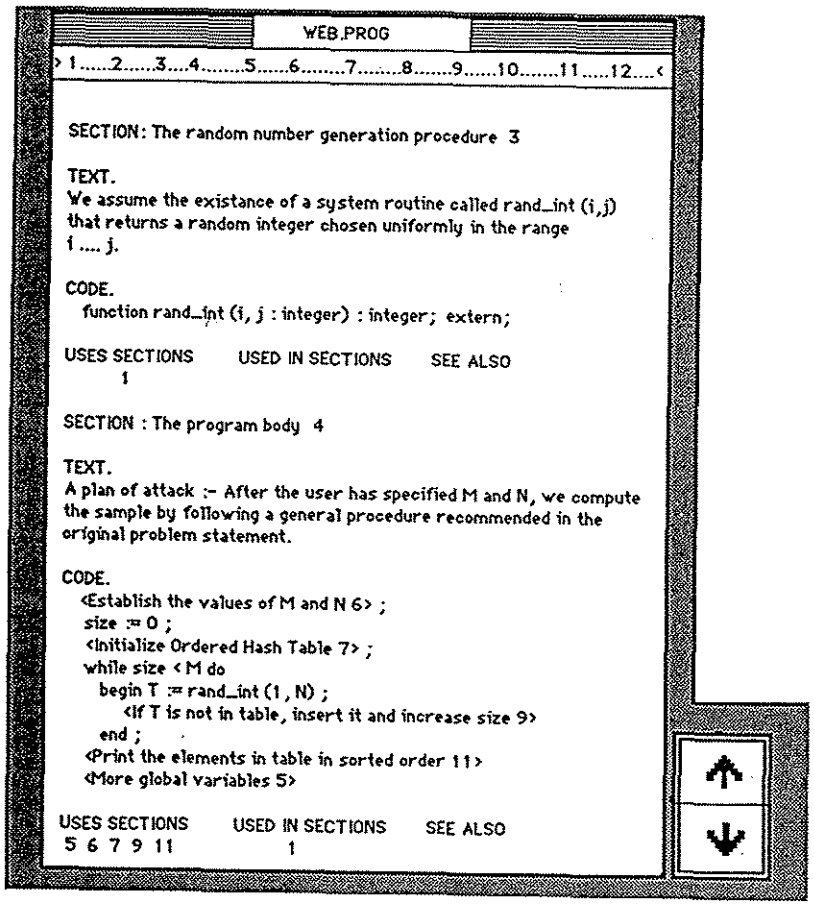

Figure 2. The Scrolling Browser

and further encroaches upon the free space (see Figure 3). When the free space is exhausted (about two sections or 50 lines unfolded) further unfolding actions result in an error message which takes the form of a beep. The user is now obliged to fold away some of the information in order to release free space. This is done in the same way as unfolding, by positioning the pointer and clicking. If the object chosen is the text or code holophrast that subsection is folded away. If the user clicks on

the section title the entire section is folded away. The user can fold away sections in an unfolded or partly-folded state, then when that section is next unfolded it will appear as it was before section folding took place.

Table 1. Program comprehension questions

1. Only one section has input and output statements in it. Which is it? (Find its number)
2. Only two sections (other than that above) have writeln statements in them. Which are they? (Give their numbers)
3. What checks are carried out on input from the user? (Specify boolean expressions involved).
4. Where is T declared? (Give section number)
5. Where is T first assigned a value? (Give section number)
6. What is T? (Give a few words of explanation)
7. What is the maximum value taken by the variable Size?
8. Where is Size incremented? (Give section number)
9. How big is the hash table?
10. What is the value of alpha? (Give an expression)
11. 11. What does alpha represent? (Give a few words of explanation)
12. T is the new candidate for insertion. Which section contains the code which detects whether T has already been inserted or not?
13. What causes the insertion process to stop? (Specify the boolean expression involved)
14. There are two cases considered when printing the results: the case where 'wraparound' has occurred and the case where it has not. What boolean condition shows whether wraparound has occurred?
15. In what two sections of code does this wraparound occur, that is wraparound in the insertion process, not wraparound in printing out the hash table? (Give section numbers and the relevant lines of code).

3. Experiment 1

3.1. Method

A short program, written by Knuth (Bently [1986]) to demonstrate the

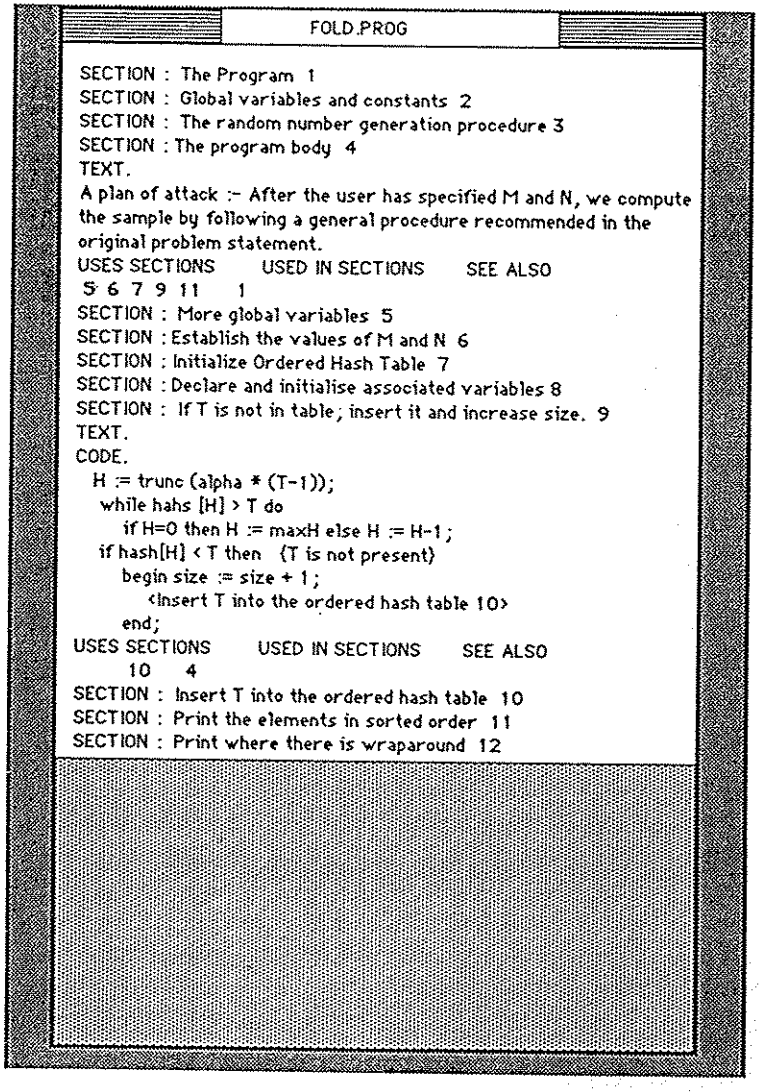

Figure 3. The Folding Browser. The user has unfolded the text portion of section 4 and the code portion of section 9

key features of literate programming, was adapted for the purposes of this experiment. The original program is reported in (Bently [1986]). Changes made were to achieve a reasonable degree of equivalence between the three browsing schemes and to adapt Knuth's Pascal-like notation to the dialect of Pascal our users were familiar with. Fifteen questions about this program, of increasing difficulty, were devised for the users to answer (see Table 1). The users tested were thirty computer science undergraduates of at least one year's Pascal programming experience. The program uses an ordered hash table. These students were familiar with the idea of a hash table but had not seen this way of using one before. None were experienced users of mouse-based systems. To avoid any carry over effects which might have arisen if the same individual was trained to use all three browsers a between subjects design was used. Ten users were allocated to each browser condition on a random basis.

After reading some instructions about the aims and methods of literate programming the users were introduced to the browser they were going to work with by means of a practice program. They then worked through the fifteen questions using the browser on the program described above. When the user obtained the answer to a question, they were instructed to tell the experimenter. The answer was recorded, but no feedback was offered. The results described below were extracted from a time stamped log generated by the system for each user.

3.2. Results

Performance data is provided in Table 2. All three groups correctly solve most of the tasks set. The rate at which the 15 tasks were performed is also given in Table 2. The original measurement in seconds was transformed to tasks per hour in order to make it more suitable for parametric statistical tests. The comparisons of interest are Hypertext vs. Scrolling and Hypertext vs. Folding. The former comparison can be shown to be significant ($p < 0.05$) but the latter is not (analysis of variance followed by Dunn's test for two non-orthogonal planned comparisons gives the minimum difference which would be significant as 17.45 tasks per hour).

The advantage experienced by the Scrolling group over the Hypertext group is surprising as examination of system logs indicates that the two groups behave in very similar ways. If the Scrolling group had basically followed the sequential expository structure, which is after all one of the major features of literate programming, then one could see how the Hypertext group might be at a disadvantage because this strategy is not available to them. In fact, both groups rely very heavily on the 'uses/used in' links which are the basis of the hypertext structure. This is

Table 2. Mean performance data for Experiments 1 and 2 (standard deviations in brackets)

	Tasks Correct (out of 15)	Tasks per hour
Experiment 1		
Hypertext	13.5 (.93)	49.2 (13.9)
Scrolling	13.2 (1.2)	68.1 (21.0)
Folding	13.1 (1.1)	56.7 (12.9)
Experiment 2		
Hypertext with map	13.7 (1.3)	69.1 (25.8)
Hypertext with list	13.3 (1.7)	51.8 (11.8)

Table 3. Percentage recall of 'Uses' / 'Used in' links

Hypertext	73
Scrolling	67
Folding	46

perhaps best illustrated in some recall data collected after the users had completed the program comprehension tasks. Each user was given the numbered section titles and asked to indicate which other sections were referenced in each i.e., to recall the hierarchical part of the hypertext structure. Mean percentage recall scores are given in Table 3. It can be seen that the hypertext group recall nearly three quarters of this information and the scrolling group do nearly as well. Clearly these two groups are paying equal attention to this part of the hypertext structure. Interestingly the folding group recall less, indicating that they were navigating through the program in some other way.

The different browsers constrain which sections can be simultaneously visible in different ways. The Scrolling Browser constrains a user to viewing sequentially adjacent sections. A user of the Hypertext Browser can only view sections which are adjacent in the hypertext space i.e., each of the two sections displayed must have a reference to the other in its 'uses', 'used in' or 'see also' indexing information. A user of the Folding Browser is unconstrained as to what sections are open at the same time. There are also different constraints upon the way a user can move about the program. It might be thought that the Hypertext Browser will require many more operations to reach some required state than the Scrolling or Folding Browsers. This is not true because the 'uses'/'used by' hierarchy is very shallow and in addition there are the 'see also' links. Figure 4 presents a map of the hypertext structure. 47% of the changes needed to open some arbitrary section, given some arbitrary screen state can be achieved in one i.e., clicking on one number. 85% can be achieved in 2 and in only one case (opening Section 12 when 2 and 10 already opened) does it take 4. Of course in practice

the transitions a user will want to make will depend on the particular strategy used to solve the task in hand.

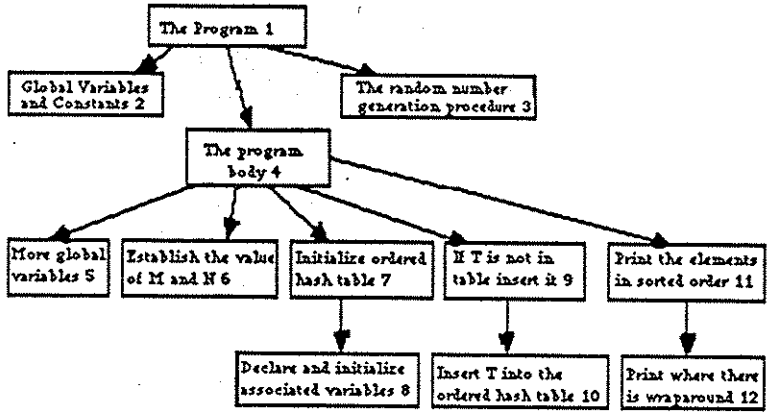

Figure 4. The 'uses/used in' hierarchy. There are also 'see also' links between sections 2, 5, 7 and 10

The better performance of the Scrolling Group when compared to the Hypertext Group could be explained in terms of these constraints. Either they impose a cognitive overhead on users resulting in generally less efficient behavior or, more trivially, the constraints simply mean that users have to engage in more system activity and the extra time taken to perform the tasks can be explained as necessary additional system response time. This latter explanation can be rejected. First, there is no evidence that the Hypertext Browser forces users into additional system activity. The system produced a time stamped record of user actions. This log was processed into a log of section visits. The smallest of the 'sections visited' scores in each group gives an indication of the minimum number of visits necessary to complete the fifteen tasks. This

was 27 and 22 for the Hypertext and Scrolling Groups respectively. For a difference of five transactions to explain the observed time difference of 323 seconds implies a system response time of around one minute per transaction! We have to conclude that the Hypertext Browser is interfering with the performance of its user in some more subtle way.

Perhaps the major difference between the Hypertext and Scrolling browsers is that the latter allows random access to the sections. Although we have ruled out the possibility that the results could be explained by the additional system response time engendered when a transition requires one or two intermediate actions to complete, there may be cognitive overheads. It is possible that the additional mental work required to complete the transition distracts the user from the main task of program comprehension and results in generally less efficient behavior. The Hypertext and Scrolling Browsers would have been much more equivalent if the thumb bar for the Scrolling Browser had an equivalent in the Hypertext Browser. This might have been a map of the hypertext structure like Figure 4. Clicking on some node in this diagram would display the corresponding section. It was not practical to generate such a radically different system for the purposes of these experiments. However, the hypothesis was tested by reducing the cognitive effort needed to make transitions within the hypertext structure by providing a non-interactive map of the structure. That is the basis of Experiment 2.

4. Experiment 2 – hypertext browsing with and without a map

4.1. Method

The Hypertext Browser was used with two further groups of subjects. One had a printed map of the hypertext structure displayed prominently to one side of the screen. This this gave the 'uses/used by' indexing information in the same form as Figure 4. This map includes the section titles and so to control for the possibility that this information alone might explain any observed improvement in performance a second control condition was introduced. This second group of 10 users had a printed list of the 12 section headings. There were twenty subjects, none of whom had participated in the first experiment.

The hierarchical map and the list of titles were introduced to the subjects as memory aids that could help them remember where information could be found. Otherwise, the present experiment proceeded in an identical fashion to the first.

4.2. Results

Table 2 includes the results of Experiment 2. Comparing the two new hypertext groups with the original in Experiment 1 we see that the rate of performance is very much improved with the addition of a map but that providing the section titles without any indication of the hypertext structure has very little effect. The former difference can be shown to be significant. Analysis of variance of the complete data set for Experiments 1 and 2 followed by Dunn's test for four non-orthogonal planned comparisons shows that the minimum difference which would be significant is 19.35 tasks per hour. The group with a map also visit fewer sections but this is not significant ($F(4, 45) = 1.34$, n.s.)

It would seem that providing a map or 'overview', to use the terminology of the Notecards system (Halasz, Moran & Trigg [1987]), is of crucial importance. An interactive map, allowing direct access to a section anywhere in the hypertext structure, might have improved performance still further. Without any kind of overview the cognitive effort required to navigate the hypertext network may outweigh the advantages of providing a non-linear text structure conforming to the demands of the task.

5. Conclusions

It would be quite wrong to conclude, on the basis of the results from Experiment 1, that hypertext will always be more difficult to use than the alternatives. Clearly the generality of any one experiment is limited to the tasks used, the user population sampled and the precise nature of the alternatives compared. The performance difference observed in Experiment 1 is interesting because it stimulated further exploration of the use of these browsers. Study of the behavior of users in the scrolling group of Experiment 1 demonstrated the salience of the 'uses/used in' links between sections, thus showing that the links the hypertext structure is based on are the important ones for users doing these tasks.

The final conclusion, that finding your way about a hypertext structure may distract from the primary task, in this case program comprehension, may be much more generalisable. With a hypertext structure of only 12 sections, providing a map resulted in a 25% improvement in performance. The improvement could be very considerable with large hypertext structures.

Hypertext provides exciting new ways of structuring information but it should be remembered that there are already well understood ways for communicating non-linear conceptual structures in conventional linear text (e.g., section headings and subheadings, forward references and so

on). While hypertext presents the designer with many new ways of helping the user, it also presents a whole new range of problems for the user and designer to solve. These problems will only come to light through systematic empirical work looking at the behavior of the users of hypertext systems. This paper is a start in that direction.

Acknowledgement

I am grateful to members of the Human-Computer Interaction Group at York who have commented on drafts of this paper, particularly Harold Thimbleby. The work was supported by the U.K. Alvey Directorate through grant GR/D/0231.7. A fuller report can be obtained by writing to the first author.

References

R M Akscyn, D L McCracken & E Yoder [November 1987], "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations," in *Proceedings of HyperTEXT '87*, Chapel Hill, North Carolina, 1–20.

J Bently [1986], "Programming Pearls: Literate Programming," *Communications of the ACM* 29, 364–369.

F G Halasz, T P Moran & R H Trigg [1987], "Notecards in a Nutshell," in *CHI + GI Conference '87*, ACM, Toronto Canada.

N V Hammond & L Allinson [1988], "Development and Evaluation of a CAL System for Non-Formal Domains: The Hitch-Hikers Guide to Cognition," *Computers and Education* 12, 215–220.

D E Knuth [1984], "Literate Programming," *The Computer Journal* 27, 97–111.

J H C Kuo, K J Leslie, M D Maggio, B G Moore & H C Tu [1986], "Information Structuring for Software Environments," in *Advanced Programming Environments*, G Goos and J Hartm, ed., Springer-Verlag.

H Thimbleby [1986], "Experiences of 'Literate Programming' Using CWEB (a variant of Knuth's WEB)," *The Computer Journal* 29, 201–211.

R H Trigg & M Weiser [1986], "TEXTNET: A Network Based Approach to Text Handling," *ACM Trans. OIS* 4, 1–23.

N Yankelovich, N Meyrowitz & A van Dam [1985], "Reading and Writing the Electronic Book," *IEEE Computer*.