AUTOMATIC TOURING IN A HYPERTEXT SYSTEM

Andrew Johnson and Farshad Fotouhi

Wayne State University
Computer Science Department, Detroit, Mi. 48202, USA

ABSTRACT: A hypertext system connects information into a graph structure where related nodes of information are connected by links. The user browses through this network of links to gain knowledge. In this paper, we propose an autonomous information gathering process which monitors the user's progress through the network, and a data structure for storing information about the user's activities in a form that can be used to aid this user, and other users of the hypertext. This information can be used to suggest anchors to the user; enhance local and global views; allow the hypertext author or user to easily create guided tours; create a global body of information making hypertexts stored across multiple sites more efficient. We show an implementation of this process which gathers the information and gives suggestions to the user on which link to use based on the previously collected information.

1 Introduction

A hypertext system connects information into a graph structure where related nodes of information are connected by links. A node (or information element) is a piece of information. A node is typically displayed as a fixed sized page, or 3by5 card. A node can contain text, a picture, a sound, a piece of animation, or a combination of the above. A node contains one or more links to other nodes. The simplest form of link moves the user from one node to another. A link is activated by selecting an anchor (a word, picture, or special icon in a node.) When the link is activated a new node appears on the screen either in addition to, or replacing, the previous node. These links connect the nodes into a network (or information-space.) For a more thorough introduction see: [5], [8], [11], [12].

Hypertext systems designed for browsing through interrelated information are supposed to make it easier for their users to access related data. Unfortunately the very flexibility of these systems causes the user to become lost in a maze of information elements. Hypertext systems lack physical markers so several navigational aids are being experimented with [10], [13]. The system can show you a history of where you've been. It can let you bookmark selected nodes so you can go back to them easily. It can give you a bird's eye view of the entire network, or a fish-eye view showing near nodes in more detail than far nodes. It can provide guided tours or paths [9], [14], [16] to show you around. It can allow higher level sets of nodes such as clusters [2] to encapsulate ideas.

Histories and bookmarks show the user where she has been. Bird's eye and fish eye views of the network show her what information is nearby. Guided tours and paths give the author of the hypertext a chance to show the user around. These navigational aids exist to help the user, so we should get her involved in the process. By keeping track of the user's past behavior, the hypertext system can make predictions of the user's future behavior. This information can also be used to create and enhance the navigational aids listed above. Other approaches to tracking the user's behavior [3], [7] rely on the user explicitly providing information to the system as it runs. We believe that much useful information can be collected without the user's conscious intervention.

This paper proposes an autonomous information gathering process, and data structure for storing the user's activity in a form that can be used to aid the users of a hypertext system. The hypertext system can use this information to suggest which anchors the user should activate; enhance local and global views of the system by highlighting previous paths and landmarks; allow the hypertext author or user to easily create guided tours through the hypertext; create a global body of information about which users are visiting which parts of the hypertext, making hypertexts stored across multiple sites more efficient. This paper then presents an implementation of this autonomous information gathering process showing how it can help the user.

Section 2 discusses what information is stored and how it is collected. Section 3 discusses how the collected information can be used. Section 4 discusses the implementation of this data collection system. Section 5 discusses extensions to this system. Finally, Section 6 gives our conclusions and plans for future work.

2 Collecting the Information

We want to see what the user is doing without bothering him too much, ideally without bothering him at all. The user wants to browse through the hypertext to get answers. If the system adds on extraneous duties, the user is less likely to use the system. The simplest way is to keep track of what nodes the user visits, and what anchors he activates. This way the user does nothing different, the system simply looks over his shoulder.

Several ideas of tracking user behavior are based on keywords or descriptors. These keywords or descriptors are grouped into concepts. The keywords for each node can be defined by the author, or the hypertext system can look for occurrences of words in the nodes to define concepts. The

user chooses which concepts he is interested in and the hypertext system starts the user off in the appropriate region of the hypertext [3], [6], [7]. With hypertexts being created in many countries these keywords can be in several languages. New systems deal with sound and video so these keywords do not even exist. Either accurate meta-information must be explicitly stored (giving the author more work to do) or we must try something else. In our system, we simply gather information on which nodes are visited, and which anchors are used, so the actual information stored in each node is unimportant. Instead of concentrating on the content of the nodes (which depends heavily on the author), we concentrate on the paths the user takes through the nodes. The previous systems require their users to make decisions affecting the promotion and demotion of keywords during their search. This is an unacceptable burden on the average user. We believe a truly useful information gathering process must work without the user's direct intervention.

Section 2.1 describes what information is stored in a simple static hypertext system where the destination anchor is always a complete node. Section 2.2 describes what changes are necessary for a more complicated dynamic hypertext system where the destination anchor can be part of a node. While there are many different types of hypertext systems in use, we feel that these two are diverse enough to show the flexibility of our method.

2.1 Static, Anchor to Node version

This simplified version assumes a static hypertext (destinations are not computed at run time) where the destination anchor is an entire node. With this assumption we can model a hypertext as a directed graph as shown in Figure 1 where:

Arabic letters (A, B) represent node IDs. We assume that each node in the hypertext is given a unique ID so the node ID labels a vertex of the graph. Each node contains 0 or more source anchors.

Greek letters (α, β) represent source anchor IDs. We assume that each anchor in a node is given a unique ID so the combination of the node ID and source anchor ID label an outgoing edge of the graph.

Each time the user browses through a particular hypertext he creates a path (a sequence of nodes visited and anchors activated). If we have a set of paths that a user has taken through a hypertext as shown in Figure 2 where:

$\int$ represents the hypertext system (where each path begins and ends.) The system includes any common starting nodes (e.g. Hypercard's Home Card), system based query commands, or other means of moving to another node without using a source anchor within the current node.

We can break the paths up into their individual 'node anchor $\rightarrow$ node' transitions and store them graphically as shown in Figure 3 where:

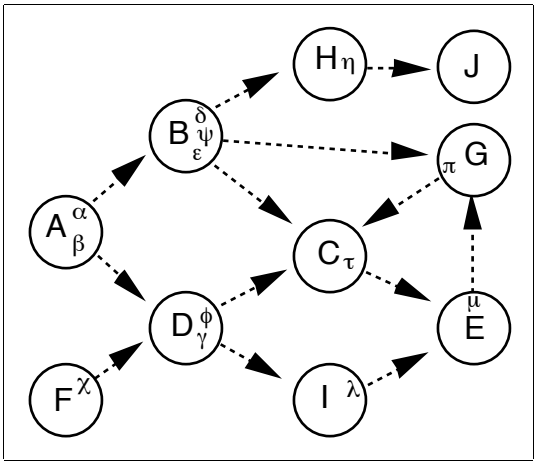


Figure 1: Graphical Representation of a Hypertext

$$\int \rightarrow A\beta \rightarrow D\gamma \rightarrow I\lambda \rightarrow E \rightarrow \int$$
$$\int \rightarrow A\beta \rightarrow D\phi \rightarrow C\tau \rightarrow E \rightarrow \int$$
$$\int \rightarrow A\alpha \rightarrow B\delta \rightarrow H\eta \rightarrow J \rightarrow \int$$
$$\int \rightarrow F\chi \rightarrow D\gamma \rightarrow I \rightarrow \int$$

Figure 2: The Paths a User Takes through the Hypertext

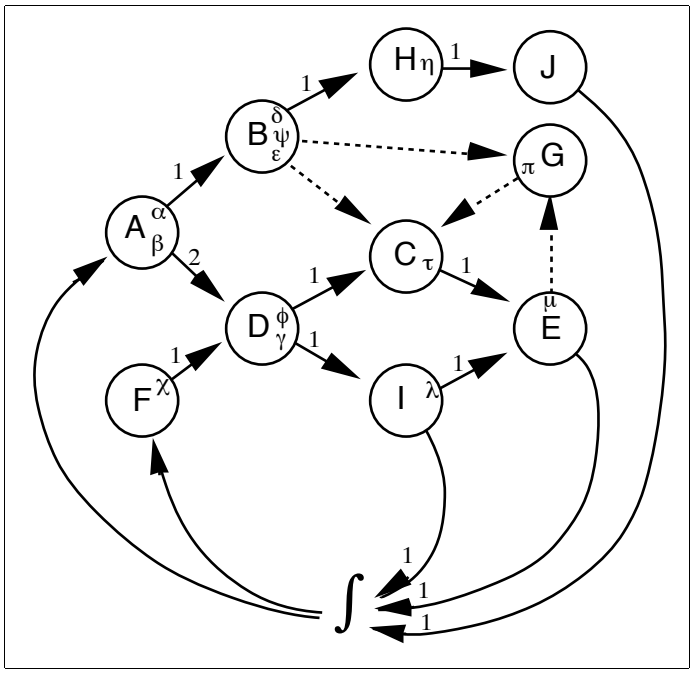


Figure 3: Graphical Representation of the Paths

A: $\int(\alpha$.1-B#β.2-D#),
B: A(δ.1-H#),
C: D(τ.1-E#),
D: A(γ.1-I#ϕ.1-C#), F(γ.1-I#),
E: C($\int$.1-$\int$#), I($\int$.1-$\int$#),
F: $\int(\chi$.1-D#),
H: B(η.1-J#),
I: D(λ.1-E#$\int$.1-$\int$#),
J: H($\int$.1-$\int$#),

Figure 4: List Representation of the Paths

Arabic numerals (1, 2) represent the number of times that a source anchor has been activated.

Using the previous node as well as the current node to give a weight to each anchor has several advantages over storing full paths, or simply using the current node to give a weight to an anchor. When a user is browsing through a hypertext, he may be thinking about one concept or several concepts. Since we do not want to bother the user by asking each time he changes concepts it would be inappropriate to store full paths in the system as each full path may contain several concepts. Our system of using the current node and previous node to give a weight to each anchor also allows the mixing of paths that have one link in common. This allows different paths to merge. However simply using the current node to give a weight to each anchor goes too far in isolating the nodes from the paths that led the user to visit them. Using only the current anchor, any paths that have a common node will be linked together and the path information will be lost. Our method of using the previous node and the current node keeps the important path information while allowing the paths to merge.

We can take the path information shown in Figure 3 and store it in a list structure as shown in Figure 4. With this list structure, it is the size of the user's exploration that determines the size of the lists. Each element of the list has the form:

$$CNID : \{PNID(\{CAID.weight - NNID\#\}^+),\}^+$$

CNID - Current Node ID
PNID - Previous Node ID
CAID - Current Anchor ID
NNID - Next Node ID

It is tempting to store the amount of time a user spends at a node, but we decided against this. We can tell which node is being displayed on the screen, but we can not tell if the user is looking at the screen. If more than one node is on the screen we can not tell which node is being looked at. We could assume that if the pointer or cursor is within a node then the user is looking at that node, but then we are asking the user to help us gather the information. To keep the information gathering process autonomic we only store the nodes visited and the anchors activated.

It is important to store both the source anchor, and the destination node (anchor.) The source anchor tells us what link to use, and the destination node tells us our destination. This gives us flexibility. If a single anchor can lead to multiple destinations (e.g., using a pop-up menu) each combination of source anchor and destination node will be given its own weight.

2.2 Dynamic, Anchor to Anchor Version

This version assumes a dynamic hypertext (destinations can be computed at run time, e.g. goto next node) where the destination anchor can be an entire node, or part of a node. A source anchor may have many different destination anchors over its lifetime. The same information is stored, but the system must do more processing with it.

For example, if a node 'A' has a dynamic link (e.g., goto next node) rather than a static link (e.g., goto node 'B') then the next node depends on the current state of the hypertext (i.e., which node is the next node from node 'A'.) If we travel from the system to node 'A' and then take anchor α to its next node, our list contains: "A: $\int(\alpha$.1-B#)," The nodes in the hypertext are then sorted or altered so that when we again travel from the system to node 'A', anchor α will take us to its next node which is no longer node 'B', but node 'C.' In our system the $\alpha \rightarrow$ C combination has a weight of 0 since anchor α has not been used to get to node C.

This same situation can arise if an anchor or node is deleted. By giving the combination of source anchor and destination anchor a weight we store values on combinations which may not currently exist in the hypertext but may exist again in the future. In a static hypertext where the destination anchor is the entire node, the *source anchor* $\rightarrow$ *destination anchor* combination never changes so the static *anchor* $\rightarrow$ *node* version of Section 2.1 is just a subset of this version.

Even when the destination anchor is part of a node, we do not store the destination anchor, only the destination node. Destination anchors are commonly used when the user wants to go to a specific part of a very large node. The user can then view all the information in that entire node and choose from any of the source anchors within that node. Since the entire node is available, the destination anchor within that node is extraneous information.

3 Using the Information

Now that we have this listing of the user's past experience we can use it for several purposes. The system can be used to enhance local and global navigation, easily create guided tours, aid in the creation of clusters, and generate statistics on the usage of the system.

3.1 Local Navigational Aids

When a user arrives at a node, the hypertext system accesses the list based on the current node and the previous node to see what anchors in the current node have been used and how often they have been used.

If the user has come to this node on a familiar path the system can lowlight any of the anchor(s) used to leave this node previously, and highlight the most common anchor(s) used to leave this node previously.

If the user has come to this node from a new direction the system can highlight the most popular anchor(s) regardless of the previous node.

If the user has never been to this node there will be no entries in the list for this node and all the destinations are equally likely, so no anchors will be highlighted.

3.2 Guided Tours

This list gives us a mechanism for creating and following simple guided tours. A hypertext can come with several predefined lists. We can use different lists for different tours or different types of users: new user, manager, student, etc. These tours can also be used as defaults for a user so he has a starting point in his searches.

Vannevar Bush [4] proposed the idea of trails where a user can link information together in his memex and then give this trail to another person to include in their memex. If two users have copies of the same hypertext, one user only needs to create a new, empty list and 'walk through' the trail he wishes to save. He can then send this list to his friend who can follow the same trail in his own hypertext.

Unlike the guided tours proposed in [14] and [9], our tours do not require a separate interface, the anchors in the nodes themselves are highlighted and the user can choose to continue the tour, or leave the tour at any point. Should the user return to the tour at the current node, or a later node, the tour will automatically continue from the user's current position. The user could also set the hypertext on 'automatic pilot' and allow the system to cycle through the path taking the highest weighted anchor at each node. Our tours are simply a walk through of the hypertext, rather than the full author enhanced guided tour proposed by [9] and [14]. While our tour offers less commentary on the hypertext, it is much simpler to create. Anyone can create a guided tour simply by creating a new, empty list and moving through the hypertext.

Our lists create branching guided tours. Given a current node, and a previous node there can be more than one source anchor with the maximum weight. The user can choose which path to take depending on his interests. Our lists are also path based rather than node based so the user can repeatedly travel back through the same node on different paths and move on to different destinations.

Zellweger states [16] that "An effective path mechanism must satisfy three major requirements: 1) It must bring expressive power to authors. 2) It must help authors create and modify paths. 3) It must help readers find paths and follow them in flexible ways." We believe our past experience lists satisfy these three requirements and gives both the author and the user the ability to easily create guided tours.

3.3 Global Navigation Aids

Bird's eye and fish eye views will be enhanced by using information from the list. By showing the user's previous paths, the system gives better landmarks. With bird's eye views, rather than just showing the 'important sights' in the hypertext, the user can see where he has been in relation to them. With fish eye views, rather than showing the local area in more detail than the remote areas, the area around the paths the user takes most often can be shown in more detail than the less explored regions.

3.4 Clustering

As hypertexts get larger there is a greater need for organizing the information into clusters - groups of highly connected related nodes loosely connected to other clusters. Currently the author decides how his hypertext will be clustered, if at all. Work being done by [2] uses the degree of connectivity among nodes to partition the hypertext into clusters. This method still relies on the number and position of links that the author has set up. Each user may have a very different idea of which nodes of information are related. We can use the previous experience lists from many users to see which nodes they move between frequently. This information allows us to combine the user's view of the information with the author's view to obtain more appropriate clustering.

If the user wants to take full advantage of this method she could create several lists for each hypertext. Each list will represent one type of work being done in that hypertext. When the experienced user starts up the system she can specify which list to use, depending on the kind of work she will be performing. The experienced user will also be able to choose a new list whenever she switches contexts. This is similar to the idea of choosing a context at the beginning of a session described in [3].

3.5 System Usage Information

For a given hypertext we can use the files from many different users to create a global usage file to get an overall view of how the hypertext is being used. We can UNION several users lists from the same hypertext together to get a picture of how people move through this hypertext (which nodes are visited most, and which anchors are used most). This information will be valuable in determining how to partition a large hypertext across several sites. These lists will also be useful in finding 'highways' and 'side streets' in the hypertext. This can be used to create new shortcut paths. We can INTERSECT two or more user's lists to find what they have in common. This will be useful to see whether people in the same group, or location, access the hypertext in the same way.

3.6 Serendipity

One of the fears about navigational enhancements is that they will limit the user's desire to explore and dilute one of

the primary advantages of a hypertext system. We can use where the user has NOT been to act as 'serendipity' pointers. At a given node, the system can show the user which anchors she has not used. We want to encourage the user to explore and find new information. Performing a NOT on the anchors of a single node is a reasonable operation since there will usually be no more than 10 source anchors in a node.

4 Implementation

Each user will need a previous experience data file for each hypertext she uses. This data file is stored as an appropriately named text file in the user's directory so it can be deleted, copied, moved, or renamed from the host operating system. Each record in the file has the format:

$$CNID : \{PNID(\{CAID.weight - NNID\#\}^+),\}^+$$

When the data file is first created it is empty. A *source anchor* $\rightarrow$ *destination anchor* combination that hasn't been taken is given weight 0 by not being in the file. As the user moves through the hypertext, the hypertext system will update the file. In order to update the data file we need to store four pieces of information in memory: PNID, CNID, NNID, and CAID. All four IDs are required to update the file since the entry for the CNID is updated when the user moves to the NNID using the CAID. Each time the user moves to a new node: the CNID becomes the PNID, the NNID becomes the CNID, the new node becomes the NNID, and the anchor we used to get to the new node becomes the CAID.

If the user visits a new node, the new line CNID : PNID (CAID.1 - NNID#), is added into the data file. If the user visits a new node from an old direction, item PNID(CAID.1-NNID#), is added into the line for the CNID. If the user visits an old node from a old direction via a new anchor, item (CAID.1-NNID#), is added into the line for the CNID. If the user visits an old node from an old direction via an old anchor, the weight of that anchor is incremented. Exploration increases the file size; repeated use does not. The actual size of the file grows linearly with the amount of the user's exploration, not the size of the hypertext. This is very important as hypertexts become much larger in the future.

We implemented this data gathering system using Hypercard. While this method is applicable to all hypertext systems we felt that the availability of Hypercard and its many stacks made it the natural choice for our initial implementation. We created a new stack called "Control" and inserted it into Hypercard's message passing hierarchy. When the 'Collect Information' switch is turned on, the Control Stack will monitor the currently open stack waiting for new cards to be opened. When a new card is opened the list is updated. The processing of the data file is performed with a combination of HyperTalk [15] code and compiled external C functions [1]. The Control Stack is only 23K including all of its external C functions.

A sample run of our data collection process is shown in Appendix A. For this example we chose Jakob Nielsen's Hypercard stack of the "Hypertext '87 trip report" [10][11]. We chose this stack because of its general availability, and its complexity. We needed to add two lines to Mr. Nielsen's stack to ensure that Hypercard's 'openCard', 'closecard' messages make it to our monitoring stack. No other changes were necessary. The user navigates through Mr. Nielsen's stack normally, clicking on buttons and moving through the hypertext. When the user quits out of the stack the list shows his movements based on the id's of the cards (nodes) he opened and the buttons (anchors) he activated.

This Control Stack also implements the two uses mentioned in Sections 3.1 and 3.2. Appendix B shows the system suggesting which anchor the user should activate based on the previous trip though the "Hypertext '87 trip report" hypertext. The Control Stack makes suggestions by flashing the suggested button on and off. The user can take this suggestion and follow that path by clicking on the highlighted button, or ignore the suggestion and click on another button.

5 Extensions to the List

Section 2.1 discusses why we decided not to store full paths, or simply current nodes. We created a method in between these extremes, however our method can be adapted to to store full paths, or only current nodes.

The weighting of the anchors in a node is currently based on one previous node. This can be extended (at considerable cost in time and space) to any number of previous nodes. This will allow us to assign a weight to an anchor in the current node based on the entire previous path. Each single PNID becomes a list of PNIDs. Alternatively, instead of increasing the number of PNIDs beyond one, we can decrease them to zero, so only the CNID is important. Each line of the list then has the form:

$$CNID : \{\{PNID =\}^*(\{CAID.weight - NNID\#\}^+),\}^+$$

We currently assume that all links are between nodes in the same hypertext. Inserting a hypertext ID (HTID) in front of each node ID, will allow the list to work with more than one hypertext. Each line of the list then has the form:

$$HTID@CNID :$$
$$\{HTID@PNID(\{CAID.weight - HTID@NNID\#\}^+),\}^+$$

6 Conclusions and Future Work

In this paper we have proposed, and implemented an autonomous information gathering process. This information aids (1) the first time user through predefined lists, or tours through the hypertext; (2) the common user by enhancing local and global navigation by showing the user where he has been, and where he most likely will go; (3) the system administrator by giving useful statistics about who is exploring which areas of the hypertext. We are currently enhancing our implementation to include the other features mentioned in Section 3.

References

[1] Bond, G. "XCMDS for hypercard." MIS:Press, 1988.

[2] Botafogo, R., Schneiderman, B. "Identifying aggregates in hypertext structures." In *Proceedings of the Hypertext '91 Conference*, (San Antonio, Texas, Dec 15-18, 1991), 63-74.

[3] Boy, G. "Indexing hypertext documents in context." In *Proceedings of the Hypertext '91 Conference*, (San Antonio, Texas, Dec 15-18, 1991), 51-62.

[4] Bush, V. "As we may think." *The Atlantic*, (July 1945), 101-108.

[5] Conklin, J. "Hypertext: an introduction and survey." *IEEE Computer*, 20, 9 (September 1987), 17-41.

[6] Croft, W., Turtle, H. "A retrieval model incorporating hypertext links." In *Proceedings of the Hypertext '89 Conference*, (Pittsburgh, Penn., Nov 5-8, 1989), 213-224.

[7] Frisse, M., Cousins, S. "Information retrieval from hypertext: update on the dynamic medical handbook project." In *Proceedings of the Hypertext '89 Conference*, (Pittsburgh, Penn., Nov 5-8, 1989), 199-212.

[8] Leggett, J., Schnase, J., Kacmar, C. "A short course on hypertext." *Texas A&M Technical Report*,

[9] Marshall, C., Irish, P. "Guided tours and on-line presentations: how authors make existing hypertext intelligible for readers." In *Proceedings of the Hypertext '89 Conference*, (Pittsburgh, Penn., Nov 5-8, 1989), 15-26.

[10] Nielsen, J. "The art of navigating through hypertext." *Communications of the ACM*, 33, 3 (March 1990), 296-310.

[11] Nielsen, J. "Hypertext and hypermedia." Academic Press, 1990.

[12] Tomek, I., Khan, S., Müldner, T., Nassar, M., Novak, G., Proszynski, P. "Hypermedia - introduction and survey." *Journal of Microcomputer Applications*, 14, 63-103.

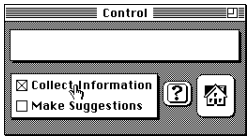
[13] Travers, M. "A visual representation for knowledge structures." In *Proceedings of the Hypertext '89 Conference*, (Pittsburgh, Penn., Nov 5-8, 1989), 147-158.

[14] Trigg, R. "Guided tours and tabletops: tools for communicating in a hypertext environment." *ACM TOOIS*, 6, 4 (Oct. 1988) 398-414.

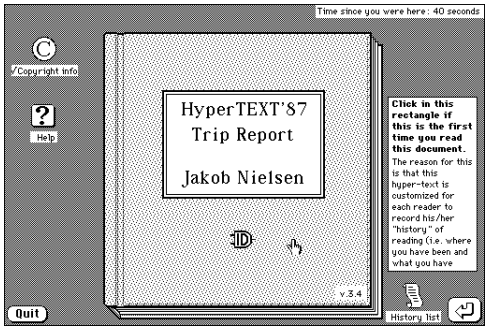
[15] Winkler, D., Kamins, S. "Hypertalk 2.0: the book." Bantan Books, 1990.

[16] Zellweger, P. "Scripted documents: a hypermedia path mechanism." In *Proceedings of the Hypertext '89 Conference*, (Pittsburgh, Penn., Nov 5-8, 1989), 1-14.

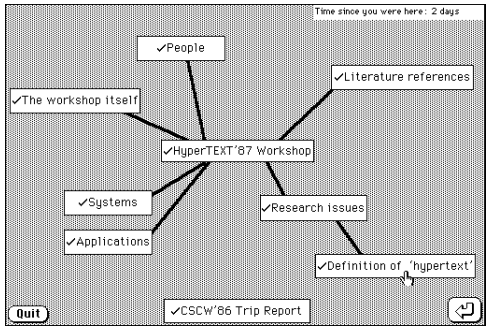
Appendix A



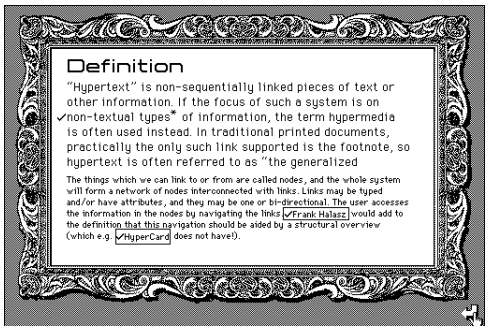
A1. Turn on information collection



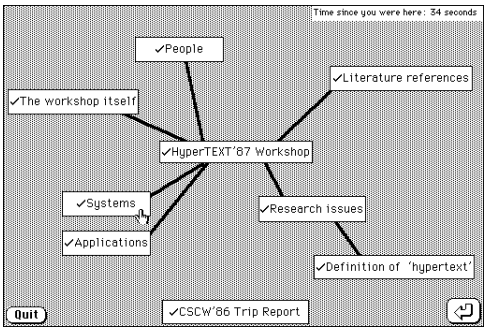
A2. Select the cover of the book



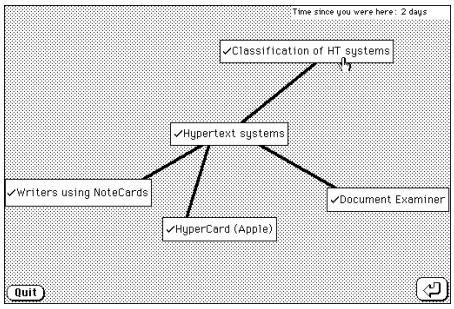
A3. Select "Definition of 'Hypertext' "



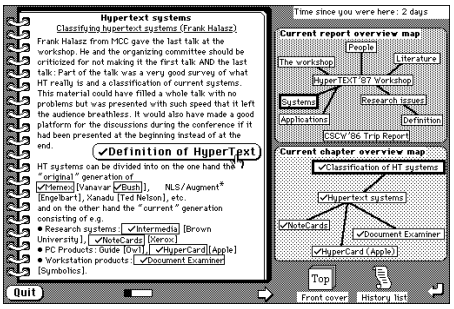
A4. Select the return arrow



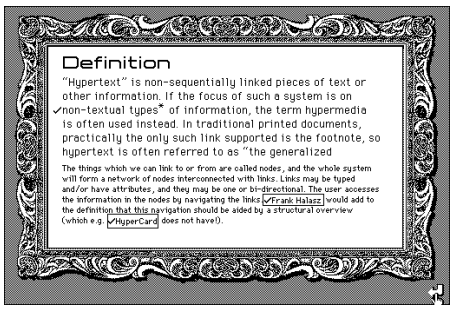
A5. Select "Systems"



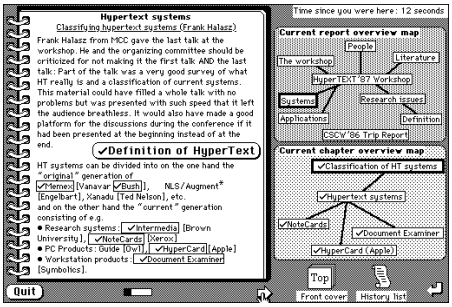
A6. Select "Classification of HT systems"



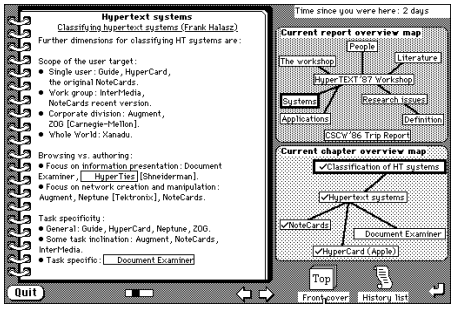
A7. Select "Definition of Hypertext"



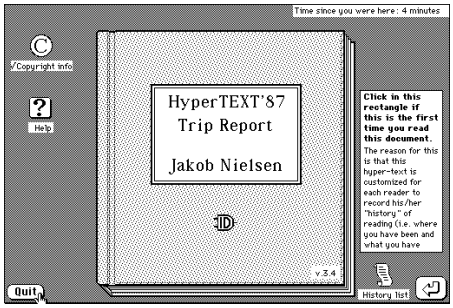
A8. Select the return arrow



A9. Select the next arrow



A10. Select "Front cover"

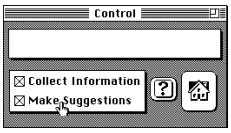


A11. Select "Quit"

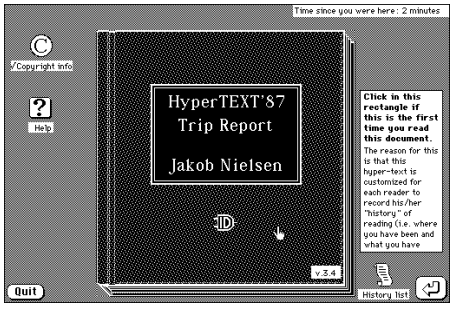
Appendix B

```
6440:S(2.1-6161#),
6161:6440(25.1-15324#),15324(15.1-31314#),
31314:6161(4.1-62480#),
15324:6161(23.1-6161#),62480(23.1-62480#),
62480:31314(14.1-15324#),15324(1.1-38994#),
38994:62480(15.1-6440#),
```

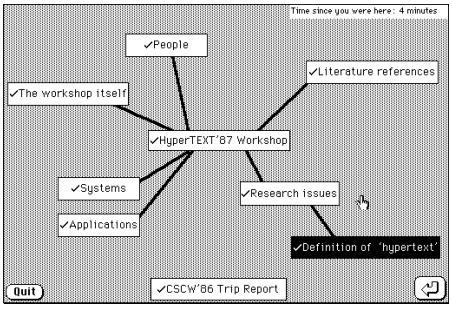
A12. The resulting list



B1. Turn on suggestions



B2. The cover of the book flashes. We follow the suggestion and select the cover of the book



B3. "Definition of Hypertext" flashes We ignore the suggestion and select "Quit"