HTGraph: A New Method for Information Access over the World Wide Web

Yee-Hsiang Chang
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304-1126
email: yhc@hpl.hp.com

Ellis Chi†
Massachusetts Institute of Technology
500 Memorial Drive
Cambridge, MA 02139
email: eyc@mit.edu

† This work was done when Ellis Chi worked at HP Labs in Palo Alto, CA

Abstract

HTGraph (hypertext graph) represents a new information accessing method based on our observations of the current World Wide Web structure. Our method extends the current Web navigation feature, which broadens its scope. The tool also couples database tools with the Web include both the navigation and database accessing paradigms. Further, the tool tries to associate Web information with real-life objects, such as a file directory structure in our case to improve usability.

1. Introduction

The World Wide Web is based on hypertext (or hypermedia). Its structure consists of nodes and links. Nodes can be Web special script files, documents, images, audio clips, and video clips; links connect those nodes over the network. When a user navigates in the Web, he/she views the content in the node. Then he/she can select one of the links for the next destination. Once the destination is reached, he can then view and pick the next one.

This kind of navigation is adequate if the purpose of the access is just looking around in a limited scope. However, it presents a problem when the number of nodes is huge. For example, assume the hypermedia nodes are arranged in a tree structure with layers, and there are ten choices in every node. Once the user selects his choice in the first layer, he/she sees only the ten choices in the selected node and misses 90 choices in other nodes. There will be 1,000 choices in layer three, ten thousand choices in layer four, and so on. In other words, the current Web navigation allows only one path out of many possible paths. The user normally loses the overall perspective once he/she is deep in the Web.

To solve the above problem, the solutions so far employ database technology through various robots [BOWM94; MACB94; MAUL94; DECE94], which automatically collect all information on the Web to build up the database. Database technology has proved its scalability in accessing a vast amount of information, so the solution is valid in this respect. However, database technology requires users to specify the search subject. If users are not exactly sure what the subject is called, database tools are less helpful. Furthermore, this approach takes no advantage of the inherent Web navigation structure, which uses links to associate (or cluster) information among nodes. The current solutions do not collect link information.

Our solution takes advantage of some previous hypertext work [NIEL90; RIVL94] and applies it to the Web environment. Our tool shows a larger scope of the Web (better than any single node can provide) through a graph with all the link information. Similar to the database tools, our tool also explores nodes on the Web and collects their information -- specifically, the title of each node. Unlike the database approach, our tool collects not just nodes but also links among nodes. Furthermore, our tool intends to associate information with real-world objects to facilitate usability, using a hierarchical graph structure similar to the file directory structure. This similarity helps people to browse in a larger scope.

The main contribution described in this paper is that the design utilitizes the Web's special features. Specifically, we take advantage of the home pages, which normally represent the starting point for a particular topic. Also, we explore nodes based on the way information is arranged around the home page. In other words, we collect nodes and links starting from these home pages to a specified degree to capture related information before the information diverges to other topics. Our tool, HTGraph (HyperText Graph), has all of the above features.

The rest of the paper is divided into four sections. Section 2 describes the concepts and operation of HTGraph. Section 3 surveys related work in this area. Section 4 shows the HTGraph's data structure and algorithms for node exploration. In section 5, the graph layout design is discussed.

2. HTGraph

2.1 Concept

We have three observations on the current hypermedia-based Web. First, information is normally organized or started from home pages, where one home page is equivalent to the root of a file directory. The home page serves as an entry point for a set of detailed nodes for a particular topic or institution that that home page represents. So, having as many home pages as possible at the first level of browsing is very useful. Here, we define a home page as being located at the highest layer. Any node that can be reached from the home page is considered as being at a lower layer; the exact layer depends on how many links are between the node and the home page.

Second, when a user follows the links on the hypermedia, he/she can take only one path at a time. The user loses more overall perspective the further he/she goes down through the layers. So, we would like to show all these choices at the same time.

Third, as the layer gets lower (*y*-axis in Figure 1), the topics get wider (*x*-axis in Figure 1).[†] The content and usefulness of information diverges and some is completely unrelated to the initial home page's topic. In fact, some of the lower layer nodes are the home pages of other topics. We would like to extract only the nodes in which most of the information is related to the topic of the initial home page.[††]

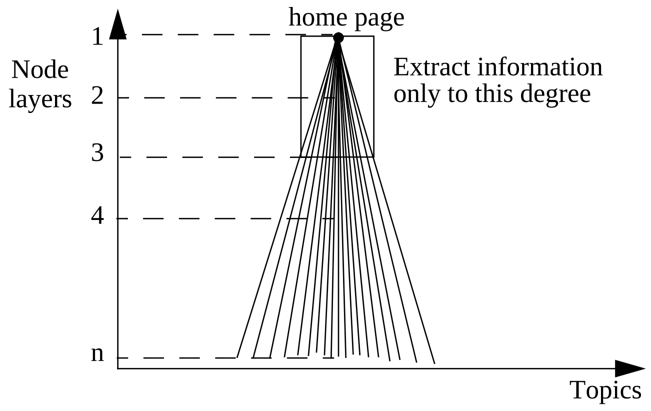


Figure 1. The Divergence of Topics Further Down the Node Layers

HTGraph is a tool to show the relationship among Web nodes as a directed graph by taking the above observations into consideration. The relationship among nodes is shown by displaying either all hypermedia references starting from the home page and its descendants, or the nodes that represent many physical linkages hiding inside the nodes -- that is, logical grouping. A good example of logical grouping would be a graph that hides linkages of a node if it has too many hypermedia nodes. Figure 2 shows an example of logical grouping.

Unlike the database approach, which retrieves only the nodes, we also collect and show the links among nodes. By doing so, we maintain the relationship among these hypermedia nodes and keep the Web's navigation feature. These links also represent natural groupings of similar nodes. For example, the MIT home page contains all the linkages to its related Web nodes. It currently contains some cultural events information in Boston for newcomers to MIT. Using the database technology, a user might not be able to specify the right query and retrieve such information. In our approach, these nodes are already linked together and are retrieved together.

† Figure 1 shows only a linear increase in the number of nodes as a user moves down the layers. In reality, this increase can be exponential, as stated earlier.

†† How much we should explore to capture the most information is unknown. Actually, each home page and its links are arranged differently depending on the creator of the page. We don't expect that a common number for the degree of detail will apply to all home pages. Currently, in HTGraph, a user can select the number of nodes he/she wants to explore.

Note that the solution further allows a “space jump” in the Web even without the exact address or Universal Resource Locator (URL) of a node. In other words, instead of following the existing node links by clicking at nodes one after another, users can “space jump” to a particular node by clicking the node on the graph.

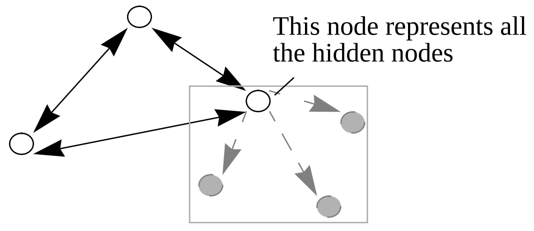


Figure 2. Logical Grouping

2.2 Operation

To run HTGraph, the steps are as follows:

Step 1: Use the existing database tool to generate the starting view.

We suggest to use either of two existing database tools, Harvest [BOWM94] or Lycos [MAUL94], to do the generation of a starting view. A user first uses the selected database tool to collect the home pages for the starting view. There are different starting views depending on the user’s preference. For example, if displaying all the possible “http://**www**.x.y.z” URLs as the starting point is a good idea, the user uses the tool (Lycos or Harvest) to retrieve the nodes that fit this pattern. Another preference could be all the “http://**www**.x.y.**edu**” URLs for the educational starting points or “http://**www**.x.y.**com**” URLs for the commercial case.

Step 2: Explore the selected home pages based on the specified degree.

The user runs the exploration part of HTGraph to access nodes from each home page to the specified degree. The tool automatically builds up the link and node information. This process should be done off-line (e.g., midnight every day) as is the case for the database and step 1. However, if the user has concerns about whether the information is current, step 2 can be skipped, and the HTGraph exploration will be done in step 4. The trade-off is performance, since the latter case requires doing both exploration and display at the same time.

Step 3: Generate the initial display with all the selected high level nodes.

The initial display of the starting view is generated when a user starts the display part of the HTGraph program with the selected starting view. We can associate the initial home pages with some physical objects. One example, shown in Figure 3, is to associate the nodes with a map. When a user starts to browse and wants to check on the Web sites in the San Francisco Bay Area, he/she double-clicks the node on the map, and a blow-up screen shows all the home pages in this region. This part is currently done manually and is still under development.

Step 4: View each individual home page graph by clicking on the page.

When a user clicks on http://www.hpl.hp.com in Figure 3, a graph is shown for this home page and associated links and nodes. Figure 4 shows the blow-up of http://www.hpl.hp.com using HTGraph for the first ten nodes, which is the degree we specified. Note that the nodes are shown similar to a directory structure. Also note that we display only the titles of the nodes instead of their addresses because the titles contain more meaning about the nodes. Since some of the titles are very long, users see the full title only when the cursor is on the specific node. Figure 4 shows an example: when a user accesses this home page, he/she can see right away that there is a node with the title “Management Profile” in layer three. He/she can then decide whether to access this node or not.

Step 5: View each node by clicking on the node.

If the user is interested in any node, he/she can click on the button and the document will be displayed. This is what we referred to as the “space jump” earlier, because a user can see further down through the layers from the HTGraph display and access the node directly.

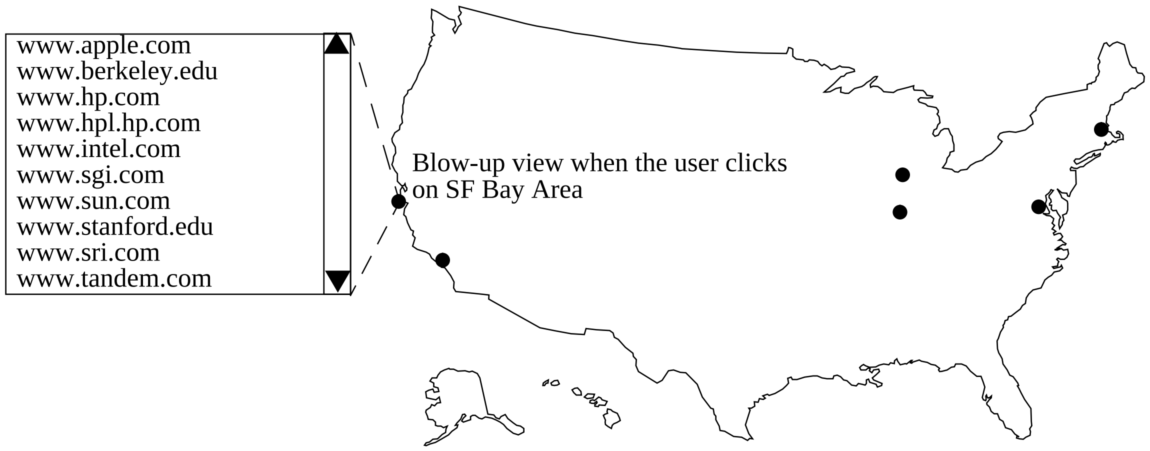


Figure 3. An Example of a Starting View from the HTGraph

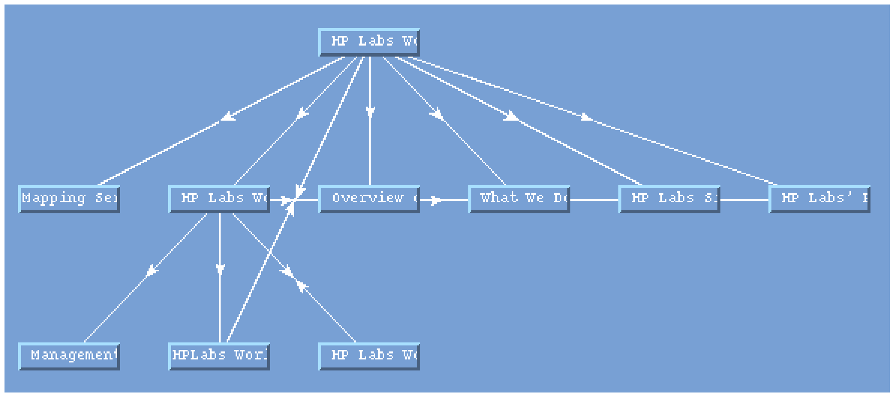


Figure 4. Display of a Wider View Starting from a Home Page

3. Related Work

3.1 Hypertext

Navigation over the hypertext technology has been studied extensively over the years [MARC88; MARS89; NIEL90; CREE91; RIVL94]. The fundamental issues have been identified and various solutions proposed. In [RIVL94], navigation over the hypertext is assisted by a structure point of view of the overall system. The user interface issue also has been addressed by [MARS88] using multiple windows, and by [NIEL90] using maps. We are taking advantage of the ideas from these efforts to apply to the Web environment.

3.2 Navigation vs. Database

One of the authors has investigated various information accessing methods in [CHAN94]. These methods fall into two categories: navigation and database. The database technique has been used extensively in the business environment. The technology is scalable in terms of the ability to access a vast amount of data. However, it is less helpful when a user has little idea about what he/she wants to see and wants only to look around, as is the case for most TV viewers. On the other hand, the navigation paradigm has proved to be powerful for the broadcast world. "Channel surfing" is a simple form of navigation within the broadcasting environment. The World Wide Web creates another form of navigation through the hypermedia links on the Internet.

3.3 Robots and Databases

Many robots have been developed on the Web [BOWM94; MACB94; MAUL94; DECE94]. Their primary purpose is to add a database function into the Web environment. The World Wide Web Worm [MACB94] and spiders [DECE94] represent tools to explore the Web and retrieve information. There are also *archie* to search the *ftp* space and *veronica* for the *gopher* space [DECE94].

The Lycos [MAUL94] and Harvest [BOWM94] tools built on the Worm's techniques, are the two most recent tools to couple Web information with current databases. They also improve the Worm technology by collecting information in a more efficient manner.

4. HTGraph Data Structure and Algorithms

Our implementation allows input of the degree of exploration and collects not just the node but also the link information. The data structure responsible for building HTGraph is called Node. Node has the following data structure:

```
struct _node {

        /* first part: info needed to build HTGraph */
        HTAnchor * anchor;
        struct _node * next;
        char * heading;
        C_list * FirstChild;

        /* second part: info for printing HTGraph */
        BOOL Printed;
        int XCoord;
        int YCoord;
};
```

A node contains two major parts. The first part keeps information for making HTGraph. The second part is responsible for printing the graph. Nodes are linked into a link list, headed by HTGraphLink (or FirstNode). The tool explores all the nodes by performing a breadth-first search. There are three major issues in building HTGraph:

- Node exploration
- Linkage to hypertext nodes
- Loop avoidance

4.1 Node Exploration

Exploring a node means to search through the whole hypermedia node, get all the accessible hypermedia references, and establish linkages. Since there is no particular goal (i.e., a particular hypermedia node) for the search, our consideration narrows down to depth-first searches and breadth-first searches. A depth-first search is out of the question, since the depth of the search may be infinite, in which case the exploration degenerates into merely retrieving the first hypermedia references in all explored hypermedia nodes. Therefore, a breadth-first search should be the most appropriate for HTGraph node exploration.

The breadth-first search algorithm presented here is slightly different from the one that is usually found in a textbook. Since the exploration here is not searching for a particular node, the way to stop the search would be either by setting up a time-out or by specifying a limit on the number of nodes explored. An ordinary breadth-first search does not care how nodes are related to one another, so it removes a node whenever it is explored. However, in HTGraph, all the explored and unexplored nodes are queued in HTGraphLink, and further linkage is implemented to relate parent and child nodes. Below is an ordinary BF algorithm and the BF algorithm for HTGraph; Figure 5 shows how HTGraphLink looks after node P is explored. Queuehead points to the node P that is currently under exploration. In the example, P is found to have three hypermedia references (called "children"), C1, C2, and C3. These children are appended at the end of the queue, since they have not been visited. After this, the queuehead moves to the next item on the link.

An ordinary BF search†

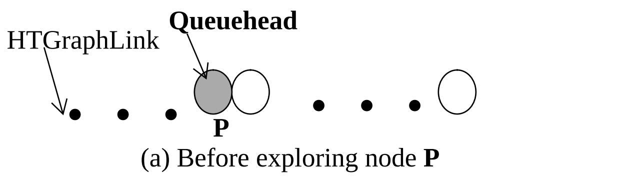
```
• Form a one-element queue consisting of a zero-length path that contains only the root node.
• Until the first path in the queue terminates at the goal node or the queue is empty,
    - Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
    - Reject all new paths with loops.
    - Add the new paths, if any, to the back of the queue.
• If the goal node is found, announce success; otherwise, announce failure.
```

A customized BF search

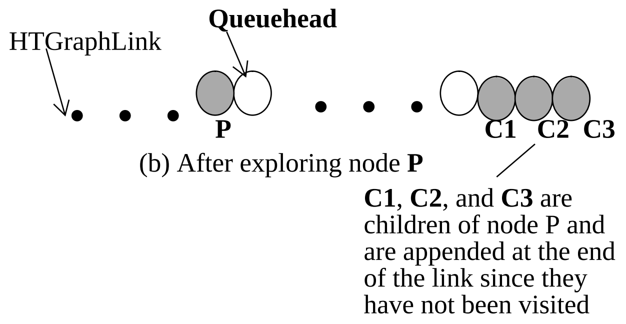
```
• Put a home hypermedia node in the BF queue.
• Until time-out or the queue is empty or specified degree is reached,
    - Explore the first unexplored node from the queue.
    - Check for loops (see whether the node has been visited (ChildVisited())).
    - Add the unexplored nodes, if any, to the
```

† Extracted from Patrick H. Winston, *Artificial Intelligence*, third edition.

back of the queue.



(a) Before exploring node **P**


HTGraphLink
Queuehead
P
C1 C2 C3
C1, C2, and C3 are children of node P and are appended at the end of the link since they have not been visited

(b) After exploring node **P**

Figure 5. Node Exploration of Node P

4.2 Linkage

We need a field that is responsible for keeping track of a node's hypermedia references or children. Since the number of children is not the same in each node, a structure with variable size is needed. A link list is chosen. By using the field FirstChild, which points to a link list that consists of a structure that points to the child, the parent-and-child relationship is established. Figure 6 shows how the linkage works using FirstChild. In the example, there are three children belonging to the first node. The first pointer on the child node points back to HTGraphLink. If the child has been visited and been recorded on the link, it points back to the location. Otherwise, it points to the new location that is created at the end of the link. The second pointer on the child node points to the next child node (NextChild) on the list.

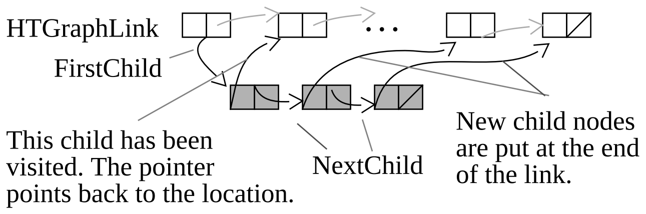

HTGraphLink
FirstChild
This child has been visited. The pointer points back to the location.
NextChild
New child nodes are put at the end of the link.

Figure 6. Child Linkage Using FirstChild

4.3 Loop Avoidance

Loop avoidance tackles the following problems:

[1] Looping.
This is a common phenomenon. An example would be a home page having its hypermedia reference referring back to itself (as shown in Figure 7). So when a graph is built, we have to make sure that a hypermedia node is not explored more than once. Otherwise, the exploration would be an endless loop between two nodes.

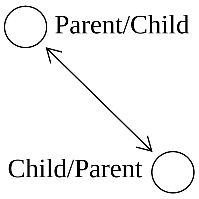


Figure 7. A Two-Node Case

[2] Multiple-parent nodes.
It is very common to find a hypermedia node being referred to in several hypertext nodes. We call this hypertext reference a multiple-parent node. A multiple-parent node should be printed only once in the graph; it should be displayed as a node being pointed to by a couple of nodes.

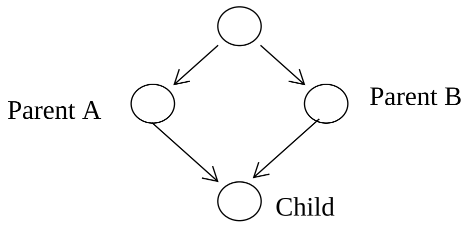


Figure 8. A Multi-Parent Case Where Parent A Shares a Child with Parent B

To ensure that a node is explored only once, the program makes sure that an explored hypermedia node will not be queued for a BF search again. To do that, a procedure named ChildVisited() is used to check if a hypermedia node has been explored (i.e., to check if the node is already in HTGraphLink). If the node has been explored, ChildVisited() returns a pointer to the node in HTGraphLink. If the node has not explored, ChildVisited() puts the node at the end of the BF queue and updates HistoryList. HistoryList is used to record the URLs of all explored hypermedia nodes and pointers to those nodes in HTGraphLink. Simply speaking, ChildVisited() gets the URL of the inspected node, compares the URL with the explored nodes' URLs, and decides if it should return the pointer to an existing node or create a new node and put it at the end of the queue. The structure of HistoryList is shown in Figure 9. HistoryList is an array that distinguishes addresses based on their length. Addresses that have the same length are linked together in a link list. The data structure in the link list consists of three fields. The first field is the address, used to determine whether this address has been visited. The second field is a pointer that points to the location of the node in HTGraphLink, and the third field is a pointer that points to the next record. Note that the data structure for HistoryList is not optimized.

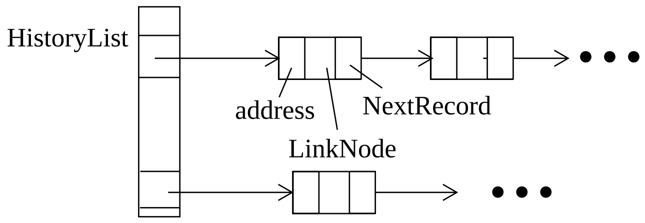


Figure 9. Data Structure of HistoryList

5. Node layout for HTGraph

To lay-out the nodes and links among one another is the key to the popularity of the tool. The difficulty in displaying a good graph is that there are many loops and links, which can occupy the same space. Currently, we are still seeking the best layout algorithm, one that can minimize the crossing of links and improve the legibility of HTGraph.

We have considered two ways to lay-out the graph. First, we can start the home page in the middle of the display, and then print all its descendants around the home page. Second, we can print the graph as a tree. The first method makes the display closer to a map, but it is hard to assign an empty spot for a hypermedia node. The second method, on the other hand, is easier to implement and shows a sense of hierarchy, which is chosen.

To make a printout of the graph, the HTGraph program generates *tcl/tk* [OUST94] command lines for each explored node in a script file that is concatenated to another script file containing the definition of the commands. The final script is invoked and the display is shown on the canvas in *tcl/tk*'s wish command. Figure 10 shows the printout of the two-node case and the multi-parent case.

6. Conclusion

In this paper, we identify the problem of accessing a vast amount of information in today's World Wide Web. We point out that the database solution takes no advantage of link information and is not very useful when the user has little idea about what to look for. We then propose a new method for Web navigation, which has resulted in a tool called HTGraph. This tool uses the features from the Web. It takes advantage of the home pages to collect nodes surrounding these home pages; it shows links among nodes in the graph, which offers natural indications about the relationship among nodes; and it also associates information with real-life objects, such as the file directory structure in our case, to improve usability. Moreover, our tool is scalable in terms of showing various levels of detail. It also allows the user to perform a "space jump" directly to the destination.

References

[BOWM94] Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U. and Schwartz, M. "The Harvest Information Discovery and Access System," *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois, October 1994, pp. 763-771.

[CHAN94] Chang, Y. H., "Wide Area Information

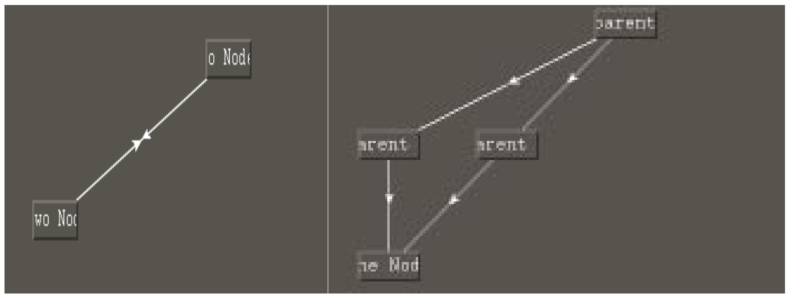


Figure 10. Two-Node Case and Multi-Parent Case

Accesses and the Information Gateways," *Proceedings of the 1994 1st International Workshop on Community Networking*, July 1994, pp. 21-27.

[CREE91] Creech, M. L., Freeze, D. F., and Griss, M. L., "Using Hypertext in Selecting Reusable Software Components," In *Proceedings of the Hypertext '91 Conference*, December 1991, pp. 25-38.

[MACB94] McBryan, O. A., "GENVL and WWWW: Tools for Taming the Web," *Proceedings of the First International World Wide Web Conference*, May 1994.

[MARC88] Marchionini, G., and Shneiderman, B., "Finding Facts and Browsing Knowledge in Hypertext Systems," *IEEE Computer*, 1988, pp. 70-80.

[MARS89] Marshall, C. C., "Guided Tours and Online Presentations: How Authors Make Existing Hypertext Intelligible for Readers," In *Proceedings of the Hypertext '89 Conference*, 1989, pp. 15-26.

[MAUL94] Mauldin, M. L., and Leavitt, J., "Web-Agent Related Research at the CMT," *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval*, August 1994.

[NIEL90] Nielsen, J., *Hypertext and Hypermedia*, Academic Press, San Diego, California, 1990.

[OUST94] Ousterhout, J. K., *Tcl and Tk Toolkit*, Addison-Wesley, 1994.

[RIVL94] Rivlin, E., Botafogo, R., and Shneiderman, B., "Navigating in Hyperspace: Designing a Structure-Based Toolbox," *Communications of the ACM*, vol. 37, no. 2, February 1994, pp. 87-96.

[DECE94] December, J., "New Spiders Roam the Web," *Computer-Mediated Communication Magazine*, 1(5), Sep. 1, 1994.