Working Paper Series
ISSN 1170-487X

Freezing Rich Fragments of the World Wide Web

by Geoffrey Holmes and William J Rogers

Working Paper 97/11
April 1997

Freezing Rich Fragments of the World Wide Web

by Geoffrey Holmes and William J. Rogers
Department of Computer Science
University of Waikato
Hamilton, New Zealand.

Abstract

While the World Wide Web (WWW) is an attractive option as a resource for teaching and research it does have some undesirable features. The cost of allowing students unlimited access can be high—both in money and time; students may become addicted to 'surfing' the web—exploring purely for entertainment—and jeopardise their studies. Students are likely to discover undesirable material because large scale search engines index sites regardless of their merit. Finally, the explosive growth of WWW usage means that servers and networks are often overloaded, to the extent that a student may gain a very negative view of the technology.

We have developed a piece of software which attempts to address these issues by capturing rich fragments of the WWW onto local storage media. It is possible to put a collection onto CD ROM, providing portability and inexpensive storage. This enables the presentation of the WWW to distance learning students, who do not have internet access. The software interfaces to standard, commonly available web browsers, acting as a proxy server to the files stored on the local media, and provides a search engine giving full text searching capability within the collection. The software gives educators the opportunity to provide the look and feel of WWW technology while limiting undesirable features of the WWW.

1. Introduction

Learning how to use the WWW is rapidly becoming as important as learning to read and write. Before long it will be the primary medium of publication for scientific and literary work. Within a decade or two it seems likely that the WWW will provide access to most if not all of the material currently obtained from libraries, television and radio broadcasts and newspapers. It is vitally important, therefore, to include such material in a Computer Literacy course.

At the University of Waikato we run a first year course on Computer Literacy that includes a unit of work on the internet. Students use a standard web browser to locate and download useful resources from the WWW. The same course is also offered throughout New Zealand at various Polytechnic sites where students can study a complete first year programme before coming into second year courses at the University. These sites do not have the same resources as the University, and while they may be connected to the internet, they cannot afford to allow students access, and could not therefore offer this unit as part of the course. In terms of quantifying the cost, last year at the University of Waikato full student access to the WWW cost the Department of Computer Science $50,000 in web traffic. This cost was incurred despite the cost-saving measure of providing a WWW proxy cache.

Even if the cost can be covered by either an institution or an individual, there are reasons why the internet is not ideal for educational purposes. First, it is possible with a perfectly innocent query to receive results that can take students to undesirable parts of the network, for example, the first hit from the Lycos search engine for the query term "games" on 25/3/97 gave:

1) !Adult CD-ROMS! Adult Video's! See Graphics! Free Catalog!
*!Adult CD-ROMS! Adult Video's! See Graphics! Free Catalog! Sex! ***** sex games adult games sexuality sexual intercourse sex games adult games sexuality sexual intercourse sex games adult games sexual*
http://www.romdezvous.com/sex1c.html (29k)
[100% relevant]

Second, there is some evidence to suggest that surfing the internet is addictive [1] and this addiction could have serious consequences for a student. Third, large latency times during peak loading on servers and networks, and the unavailability of web pages when a server is down, can lead to students gaining a very negative view of the technology. While this may be more realistic, it is not the experience that a newcomer to the technology should have.

We have observed that much of the educationally valuable content on the WWW is static. The web provides a general user interface which can be used to access a variety of information. However, the most common form is the hypertext document with associated pictures, movies and sounds. Such documents and their attachments are usually static, in the sense that they do not change between accesses. If two people request such a page, they receive exactly the same information. Such static data can easily be cached locally, and provided to several people without multiple communication costs. Not all documents are static. The WWW provides a form interface which can be used to query a database, or run some program on a server, dynamically generating appropriate response pages. The most important use of this interface is to access search engines.

In order to allow students at remote Polytechnic sites to complete the internet unit and to more generally avoid these negative views of the WWW we developed a system to freeze rich fragments of the WWW onto inexpensive local storage media. The system which indexes and serves the rich fragments can be put onto a large disk or onto CD ROM for distance education students.The software interfaces to standard, commonly available web browsers, acting as a proxy server to the files stored on the local media. We also provide a search engine giving full text searching capability to the collection. This gives students experience of a forms interface. The software offers educators the opportunity to provide the look and feel of WWW technology while limiting undesirable features of the WWW.

The rest of this paper is structured as follows: Section 2 explains how the software gathers, stores and indexes files from the internet, and how these files are served to a standard web browser. Section 3 demonstrates how users navigate through the collection. The issues involved in making good collections are briefly discussed in Section 4. Section 5 provides a discussion of the general applicability of the software followed by a conclusion.

2. System Architecture

The software we have developed consists of two major components. The gathering and indexing component is responsible for collecting a set of pages from the internet and building a full text index to those pages. The server component is responsible for providing files and processing search requests for a standard web browser such as Internet Explorer or Netscape Navigator.

2.1 Gathering and Indexing

There are four programs used in this process.

`gatherer`	Collects files from the internet.
`builder`	Constructs MG index of all text pages.
`indexer`	Extracts non-text pages and builds url index.
`addfiles`	Allows addition and removal of files.

Their operation, and the files used at each stage, are illustrated in Figure 1. The 'gatherer' program requires two data files. The first file (*url list*) is primarily a sequence of urls—addresses of files to retrieve. However, its format also allows url citations to provide an audit trail for locating faulty links. When starting a collection the *url list* will usually contain just a single entry. Later it may hold many thousands of urls. The second file (*limits*) lists the locations from which files may be taken. It is possible to specify the subnet, machine, port and/or subdirectory from which files must come. The basic operation of the *gatherer* program is to run through its *url list* retrieving each file, subject to the url satisfying an entry in the *limits* file. As each file is received, it is added to the *collection* file, and classified as text, html or something else. If it is an html file it is parsed for new links —written to the *links* file with citation. Any faults occurring—such as sites or files being unavailable, or problems with communication—are logged to the *faults* file. Entries in *url list* which are rejected for not satisfying the *limits* specification are written to the *rejects* file. To avoid loading the same file more than once the program maintains a table of the urls of collected files, which it saves in the *summary* file between runs—the *summary* file simply reflects the content of the collection file, in a form which can be quickly loaded. The *links* file is written in the appropriate form to be used as a *url list* for a

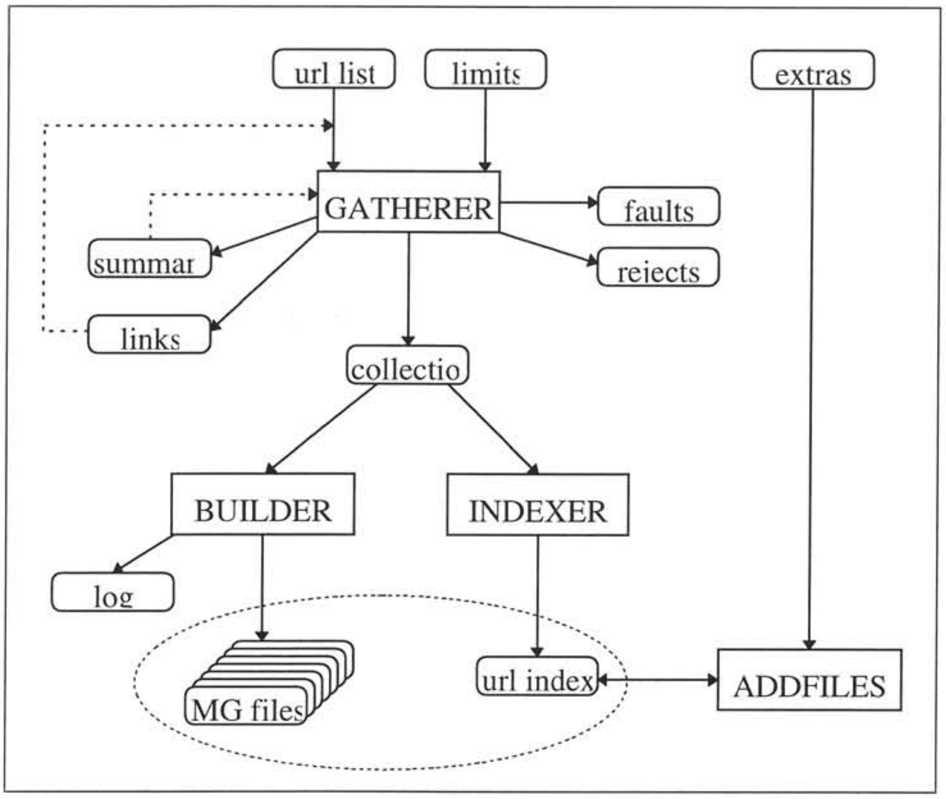


Figure 1 System Architecture

subsequent run of the program. Successive runs can therefore retrieve, in a breadth first search order, a tree of linked web pages with their associated graphics etc. The process can be illustrated by showing how files from the University of Waikato collection could be gathered. An initial *url list* (file name `CS1.DAT`) is:

```
! starting point
-http://www.cs.waikato.ac.nz/
```

In place of a citation was the text 'starting point' to indicate that the url was not obtained as a link. The second line is the starting url for the Computer Science Department web pages. A suitable *limits* file is the single line:

```
cs.waikato.ac.nz
```

Incoming urls will be accepted if their machine part ends in the manner shown —that is, the machine must be connected to the Computer Science Department subnet of the Waikato University system. Had the *limits* entry been just `waikato.ac.nz`, files from any machine at Waikato would have been accepted.

The *gatherer* program is run from a command line, and takes three parameters. The first is the prefix of the *url list* file name. The second and third are the first and last levels of search to run. The simplest case is:

```
gatherer cs 1 1
```

This will do a single run, fetching the file whose address is in `CS1.DAT` and writing any links found to `CS2.DAT`. The resulting `CS2.DAT` file starts with:

```
! From http://www.cs.waikato.ac.nz/
http://www.cs.waikato.ac.nz/icons/scms-title.gif
http://www.cs.waikato.ac.nz/welcome.html
http://www.waikato.ac.nz/
http://www.cs.waikato.ac.nz/whats-new.html
...
```

The new urls can be followed by:

```
gatherer cs 2 2
```

generating `CS3.DAT`, and so on. More conveniently, all links from start to the 10 th level can be pursued in a single operation, creating `CS2.DAT`, `CS3.DAT`, ... `CS11.DAT` on the way using the command:

```
gatherer cs 1 10
```

Collections from more than one starting point may be built in parallel by including both starting points in one *url list*, or they may be done separately using independent sequences of *url list* files. They may be built in single or multiple runs of the program. In each case all files retrieved are accumulated in the *collection* file. It holds url, type information, and retrieved data. It is the only file required to proceed to the next stage of the process, building a full text index to the collection.

The 'builder' program constructs a full text index. It is a version of the Managing Gigbytes (MG) software [2] which has been ported to the PC/Windows NT/95 system, and modified to read the collection file format. It reads through the collection file, extracting and processing only those documents which are either text or html source. MG was designed to process and compress very large text databases, creating indices which allow queries to be run, and documents retrieved on ordinary small workstations. Its index construction process however, is memory intensive. A (compressed) boolean matrix of words in documents must be built and inverted. We use a PC with 60Mbytes of main memory for index construction. Peak memory requirements in the range 40-50Mb have been observed when compressing 100MB of mixed html and plain English text.

The compression techniques used by MG treat documents as alternating sequences of words and non-words (spaces, newlines, etc.) and assume that the number of distinct words and non-words used are reasonably small. Erronenously including graphics or other non text material in the input to the indexing process leads to an explosion of dictionary size and a corresponding huge increase in memory requirements. For this reason it is necessary to be extremely careful when classifying documents collected by the browser. Each document retreived from the WWW comes with a MIME header [3], including a type field. In theory this type field specifies the form of the content: text/html, text/plain, graphics/gif, etc. In practice the type field is not reliable. It is common for graphics and sound files to be returned labelled as holding html text. There appears to be no easy solution to this problem. The 'gatherer' uses a heuristic classifier, making use of specified type, any file name suffix present, and frequency of non graphic ASCII codes. It is conservative in classifying files as text—that is, in cases of uncertainty files are taken to be non text. This protects the compression system from binary data. Mis-classification of a text file as non text has two consequences: the file is not indexed; and any html links in the file are not discovered. The frequency of problems arising from the current heuristic algorithm appear to be low.

The MG system provides text content searching, but it does not provide the primary document 'lookup' needed to allow retrieval of a document given its url. The first function of the 'indexer' program is to construct such an index. The anticipated number of documents in a collection could be quite large. For the University of Waikato site this number is approximately 30,000. Simply storing all the urls (without compression) takes nearly 2Mb of memory. For this reason, the 'indexer' program saves url data in a file (*url* index). The file holds url, type, and data reference for each document in a collection. The file is accessed using a perfect hash function [2]. Memory requirements are linear in the number of urls (approximately 0.25Mb of main memory for 30,000 urls). The hash function parameters are stored at the end of the *url index* file for quick loading by the 'server' program.

The 'indexer' program also processes the non text files of a collection. Most non text data from the web is already compressed. Whilst small improvements in size might be achieved by altering the compression method used (gif to jpeg, for example) there is a great advantage in not having to be able to recognise or process any of the data formats used. Accordingly we simply store non text data unmodified—it is all stored in *url index*. The data reference for text files is simply a number. MG numbers the documents it holds, and has the option of retrieval by document number.

The final program in the set is the 'addfiles' program. This provides a convenient mechanism for adding a header page and a search page to a collection. The program is also capable of removing added files in case some revision is necessary. It is driven by a parameter file which provides the name of the file to add, and the url by which it may be retrieved.

2.2 Serving

The basic problem to be solved in implementing the system was to persuade an unmodified web browser to direct requests to our collections of files instead of to the internet. We hoped to find a simple solution which would avoid the need to do any network programming. A number of possible methods were considered.

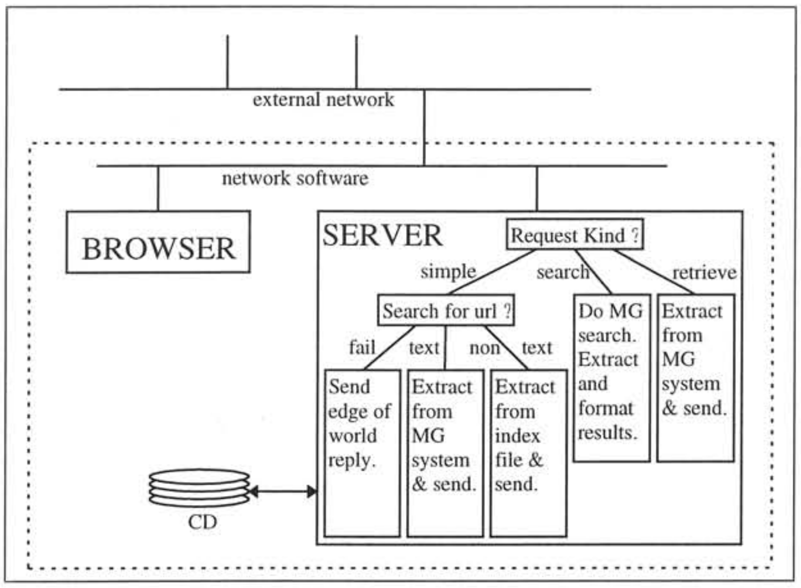
At first we tried to make use of the browsers' built-in cache capabilities. This proved impractical for two reasons. First, the cache systems would not function reliably on a read-only file system. Second, the system of storing pages as independent files was too wasteful of disk space.

The next idea was to edit the links in each html document of the collection so as to have them directly reference disk files (or a retrieval script to avoid storing pages in separate files). However, the primary purpose of our system was to provide a facility for teaching students to use an internet browser, without the problems inherent in providing full internet access. It was therefore important that we did not alter the look and feel of the browser in any significant manner. Altering links would lose the identification of document source, and make it impossible for a student to directly enter a url.

Finally, we accepted the necessity to do some network programming. A well established mechanism in WWW implementation is the idea of a 'proxy'. Web browsers can be configured to send all requests to a proxy instead of directly to the machines specified in each url. The communication protocol is just the same. The only difference is that the proxy is sent an entire url, not just the file identification bit. The usual purpose of a proxy is to maintain a cache of recently used files, but they can also be responsible for some authentication and charging activity. Of course, under normal circumstances it will pass a request on to the addressed machine if the request cannot be satisfied from cache. Presence of the proxy is completely transparent to a person using a browser. We decided to build server software which would function as a proxy from the viewpoint of a browser. To minimise programming effort we looked for existing software to modify. A web server called FNORD [4] which ran on Windows NT/95 was found on the internet with source code under the GNU General Public Licence. FNORD was particularly well suited to our application because it was capable of running on the same machine as a browser, without a network connection.

The browsing software configuation for the system is shown in Figure 2. The dotted lines represent the boundary of a single machine. Thus, our modified version of FNORD, here called 'server' runs on the same machine as 'browser' which can be any web browser. As indicated in the diagram it is possible to retain an external network connection, over which browsers running on several machines can share a single copy of a collection and the server.

The 'server' program classifies incoming requests into two classes: special and simple. Special requests have urls starting `http://cd-net/server`. At present 'server' responds to two kinds of special request: MG search request, and MG retrieval of a search result. This processing is done within 'server' to avoid the installation complexity and performance penalty associated with cgi scripting. Simple requests are just that: requests for web pages or other files, addressed by url. In response 'server' checks its collection *url index*. If the file is not in the collection an error page is generated and returned. If the file is available, the file type is used to determine whether it must be decompressed by MG, or simply retrieved from *url index*. In either case it is returned to the browser.



external network
network software
BROWSER
SERVER
Request Kind ?
simple
search
retrieve
Search for url ?
fail
text
non text
Send edge of world reply.
Extract from MG system & send.
Extract from index file & send.
Do MG search. Extract and format results.
Extract from MG system & send.
CD

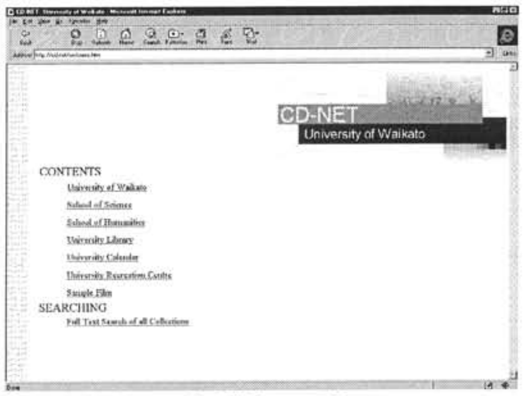
Figure 2 Browser-Server Interface

3. Navigating a Collection

The welcome page of the system is shown below in Figure 3a. The user is supplied with a number of broad category urls which represent a collection overview and which can be treated as starting points for browsing that particular collection of material. This style of presentation is only really feasible with small collections. The page also contains a link to the search page.

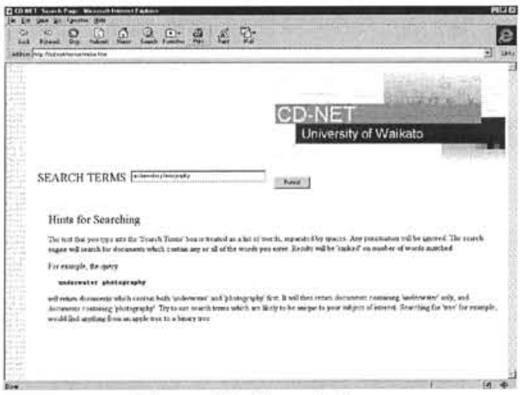
More complete access to the collection is achieved via the search page and is accessed in the usual way from the browser, or from the welcome page. This page, shown in Figure 3b, has been modelled on the Digital Library under development in the Computer Science Department here at the University [5]. Information is accessed using the same keyword and ranked boolean systems that are commonly available on the WWW. Results of queries are returned in ranked order, see Figure 4a, and represent word matches against all the documents in the collection that contain the word or words in the query. Each result contains a hyperlink simply labelled LINK to the document, plus a description of the match. This description can either be the title field of the html page or the first 100 non-tagged characters in the matched html document.

When a user requests a page that is not in the collection a special page Figure 4b is displayed. As can be seen, the page informs the user that they have reached the 'edge' of the collection and that the information they seek can only be obtained by direct access to the indicated url.



CONTENTS
University of Waikato
School of Science
School of Humanities
University Library
University Calendar
University Recreation Centre
Sample Film
SEARCHING
Full Text Search of all Collections

Figure 3a Welcome Page



SEARCH TERMS underwater photography

Hints for Searching

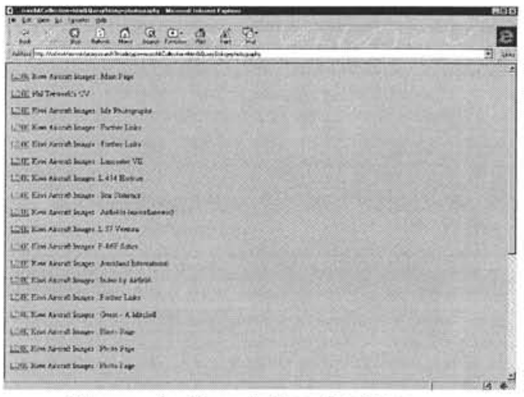
The text that you type into the 'Search Terms' box is treated as a list of words, separated by spaces. Any punctuation will be ignored. The search engine will search for documents which contain any or all of the words you enter. Results will be 'ranked' on number of words matched.

For example, the query

`underwater photography`

will return documents which contain both 'underwater' and 'photography' first. It will then return documents containing 'underwater' only, and documents containing 'photography'. Try to use search terms which are likely to be unique to your subject of interest. Searching for 'tree' for example, would find anything from an apple tree to a binary tree.

Figure 3b Search Page


[illegible] Kiwi Aircraft Images : Main Page

[illegible] Phil Treweek's CV

[illegible] Kiwi Aircraft Images : My Photographs

[illegible] Kiwi Aircraft Images : Further Links

[illegible] Kiwi Aircraft Images : Further Links

[illegible] Kiwi Aircraft Images : Lancaster VII

[illegible] Kiwi Aircraft Images : L.414 Hudson

[illegible] Kiwi Aircraft Images : [illegible]

[illegible] Kiwi Aircraft Images : Airfields (miscellaneous)

[illegible] Kiwi Aircraft Images : L.37 Ventura

[illegible] Kiwi Aircraft Images : [illegible]

[illegible] Kiwi Aircraft Images : Auckland International

[illegible] Kiwi Aircraft Images : Index by Airfield

[illegible] Kiwi Aircraft Images : Further Links

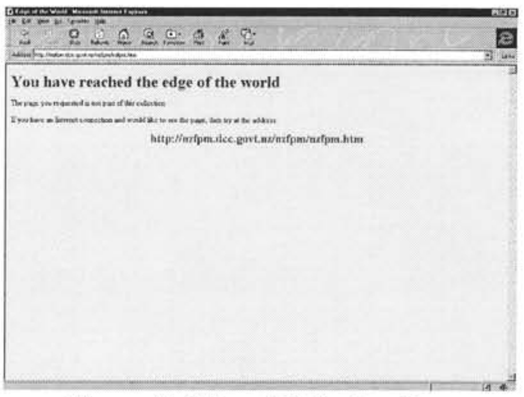
[illegible] Kiwi Aircraft Images : [illegible]

[illegible] Kiwi Aircraft Images : Photo Page

[illegible] Kiwi Aircraft Images : Photo Page

[illegible] Kiwi Aircraft Images : Photo Page

Figure 4a Search Results Page


You have reached the edge of the world

The page you requested is not part of this collection

If you have an Internet connection and would like to see the page, then try at the address

http://nzfpm.dcc.govt.nz/nzfpm/nzfpm.htm

Figure 4b Edge of Collection Page

4. Making Collections

The practical success of the system depends on gathering interesting and usefully complete collections of web documents within reasonable storage constraints. If we are to save a collection on CD ROM the upper size limit is 650Mb including the index files. To preserve the flavour of the web, it is important that a collection does not have too many unconnected links—a user should be able to browse, following reasonable lines of inquiry, without continually reaching 'edges'. The challenge, therefore, is to find suitable sets of files.

We have observed some patterns in web site structure. Firstly, we were surprised by the sheer size of web sites. At the University of Waikato, for example, there is at the time of writing approximately 600Mb of data in nearly 30,000 files. This has all been put in place over a period of three or four years by sections and individuals, with no special corporate commitment to web development. The files are quite well organised. Different schools of study and administrative sections have their own machines. Individuals have files under personal directories, usually structured in subdirectories if they have more than one area of interest. 'Foreign' links are usually isolated and fairly obvious to the reader. For example, an individual may have a page with links to sites they find interesting, separate from pages of their own original material. Waikato's organisation seems typical of University sites. Our 'gatherer' program is designed around this observed structure. Beginning with a small number of starting urls—usually 'home' pages of a site as a whole, or of an individual's collection on a site—the program pursues all links in a breadth first search, subject to scope restrictions limiting machines and/or directories to access. It can therefore collect all linked files on a given machine, or from a given directory tree.

Setting up a collection on a given topic is reasonably straightforward. We start by using a search engine to find promising documents. After a small amount of manual exploration it is usually possible to decide on the best starting points, and on the machines and directories which hold useful information. The 'gatherer' program can then collect a complete set of linked files. There are two ways to check this for completeness. One is to try browsing the collection, looking for major omissions. The second is to scan the list of rejected links. It is best to begin with conservative search restrictions, and gradually widen the scope of search until a satisfactory collection is developed, or until storage limits are reached.

5. Discussion

A number of issues arise when the system is written to CD ROM. Not least is the issue of copyright. We take the view that the system acts as a WWW proxy cache like thousands of others around the world. The only difference is that a WWW proxy cache usually has a replacement policy for web pages and so could be viewed as an amorphous collection rather than something more solid. Aside from the complex legal issues, the CD ROM solution has great educational potential. Not only can it be used in distance learning, but students can take it all home and carry on with their laboratory work.

The cost to educators with this system is that they must make an effort to provide good starting places for students. This cost is a real one for both full and limited internet access. Letting students loose on either system would not be profitable although somewhat easy for the teacher. Considerable effort has to go into finding good material and there is no escaping this fact. This process is being made easier, however, by site reviews which many search engine producers are now providing.

Finally, it should be noted that the system provides a general framework for other educational applications, for example, the provision of tightly focussed Digital Libraries; packaged collections of html resource materials for distance learning; and advertising an Institution by collecting its web site onto CD ROM and sending it to potential customers.

6. Conclusion

We have presented a system that is capable of capturing, storing and indexing rich fragments of the WWW for use in a Computer Literacy class. The system can be accessed from a standard web browser and gives the user the look and feel of the WWW with low cost to the Institution. Undesirable material is avoided by careful selection of sites, and the size and scope of the collection draws students more gradually into the technology, limiting the cost in student time. The system is predictable and reliable, and the speed is acceptable, giving students a positive first view of the technology. When written to CD ROM the system provides a small low-cost internet education facility for distance learning purposes.

It is clear that the WWW is simultaneously a useful resource and a trap for the unwary student. Whether the system presented here encourages good habits ahead of full internet access remains an open question and one we intend to pursue in the near future.

7. Bibliography

[1] Paxton, J. T. "Webucation: Using the Web as a Classroom Tool", *Proceedings of the twenty-seventh SIGCSE Technical Symposium on Computer Science Education* , Philadelphia, Pennsylvania, Feb 1996.

[2] Witten, I.H., Moffat, A. and Bell, T.C. *Managing Gigabytes* Van Nostrand Reinhold, 1994.

[3] Franks, M. *Internet Publishing Handbook* Addison Wesley, 1995.

[4] Morin, B. *Fnord Server* site http://www.wpi.edu/~bmorin/fnord/index.html

[5] Witten, I.H., Cunningham S.J. and Apperley M.D. "The New Zealand Digital Library Project" *New Zealand Libraries* 48(8): 146-152, 1996.