To appear in the 1997 Usenix Technical Conference, Jan 6-10, 1997.

WebGlimpse — Combining Browsing and Searching

Udi Manber,[1] Mike Smith[2], and Burra Gopal

Department of Computer Science
University of Arizona
Tucson, AZ 85721
{udi | msmith | bgopal}@cs.arizona.edu

ABSTRACT

The two paradigms of searching and browsing are currently almost always used separately. One can either look at the library card catalog, or browse the shelves; one can either search large WWW sites (or the whole web), or browse page by page. In this paper we describe a software tool we developed, called WebGlimpse, that combines the two paradigms. It allows the search to be limited to a *neighborhood* of the current document. WebGlimpse automatically analyzes collections of web pages and computes those neighborhoods (at indexing time). With WebGlimpse users can browse at will, using the same pages; they can also jump from each page, through a search, to ‘‘close-by’’ pages related to their needs. In a sense, our combined paradigm allows users to browse using hypertext links that are constructed on the fly through a neighborhood search. The design of WebGlimpse concentrated on four goals: fast search, efficient indexing (both in terms of time and space), flexible facilities for defining neighborhoods, and non-wasteful use of Internet resources. Our implementation was geared towards the World-Wide Web, but the general design is applicable to any large-scale information bases. We believe that the concept of combining browsing and searching is very powerful, and deserves much more attention. Further information about WebGlimpse, including the complete source code, documentations, demos, and examples of use, can be found at http://glimpse.cs.arizona.edu/webglimpse/.

[1] Supported in part by NSF grant CCR-9301129, and by the Advanced Research Projects Agency under contract number DABT63-93-C-0052.

The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

[2] Current address: Qualcomm, Inc., San Diego, CA 92121

1. Introduction

Browsing and searching are the two main paradigms for finding information on line. The search paradigm has a long history; search facilities of different kinds are available in all computing environments. The browsing paradigm is newer and less ubiquitous, but it is gaining enormous (and unexpected) popularity through the World-Wide Web. Both paradigms have their limitations. Search is sometimes hard for users who do not know how to form a search query so that it is limited to relevant information. Search is also often seen by users as an intimidating ‘‘black box’’ whose content is hidden and whose actions are mysterious. Browsing can make the content come alive, and it is therefore more satisfying to users who get positive reinforcement as they proceed. However, browsing is time-consuming and users tend to get disoriented and lose their train of thoughts and their original goals.

These two paradigms are used separately by most systems. A site may give you a link to a search page that will allow you to search the whole site. But this search will not generally take into account where you are coming from, therefore it will not incorporate any knowledge you already gained while browsing the site.

We argue that by combining browsing and searching users will be given a much more powerful tool to find their way. We envision a system where both paradigms will be offered *all the time*. You will be able to browse freely — the usual hypertext model — and you will also be able to search from any point. The search will cover only material *related* in some way to the current document. (Of course, global search may be offered too.)

Suppose that you are looking for information on network research at a certain institution. You may first go to their home page. If you search for ‘network’ throughout the institution, you’ll get too many unrelated hits. If you search for ‘network research’ you may get too few hits, because the word research may not appear in research documents (some keywords are often missing because they are implied by context). On the other hand, if you could search only the pages of the research department, or only pages related to research, then a query for ‘network’ may be quite relevant. You will be getting the context for your query from the pages where you initiated it. Such search is much better than the current most common way to get that information, which is to browse back and forth through dozens or hundreds of pages. After all, the computer is the one that is supposed to do most of the work, not you.

Some simple facilities of focusing the search while browsing have been employed. We have attempted limited browsing and searching facilities in our GlimpseHTTP package [1]. Since the work on GlimpseHTTP has not been published (aside from the code), we describe here some of the features that WebGlimpse borrowed. GlimpseHTTP indexes a UNIX file system, and provides search that can be focused to any part of the directory tree. For example, if you are looking for a Computer Science citation (our most popular demo [2] of GlimpseHTTP), you can browse one of 16 different categories and perform the search only for the current category. Only one index is needed for the whole archive; the CGI program identifies the page from which the search originates, and limits the search accordingly. The search pages are created automatically by GlimpseHTTP. If the information is hierarchical, and it is organized in a corresponding hierarchical file system, GlimpseHTTP works well. But its browsing and searching capabilities do not apply to arbitrary links (or to sites with information that is not neatly organized in a tree structure). GlimpseHTTP has been very successful; it is used in more than 500 sites [3], some quite major. WebGlimpse is a natural extension of GlimpseHTTP, borrowing some of its features but taking them much further.

The idea of searching only parts of a hierarchy is now used by Yahoo [4] to search only one category at a time. This is done in a very similar fashion to GlimpseHTTP, and it suffers from the same problem. For example, if you search for ‘‘mentally ill’’ under the ‘‘Arts’’ category, you will miss a site that features a ‘‘collection of art by mentally ill people’’. The reason for that miss is that this site is listed under ‘‘Arts Therapy’’ which is listed under Arts, but only as a symbolic link (it is located in the ‘‘Mental Health’’ category). Yahoo probably made a good design decision not to follow symbolic links in the search, because following several such links may result in unrelated material. But following only one symbolic link from any subcategory would probably have been better. This is the kind of feature that WebGlimpse easily provides.

Other web search servers take a different approach. Both InfoSeek [5] and Lycos [6] offer ‘‘Find Related Sites.’’ The relationship is computed by searching the whole database for keywords similar to the major keywords of the given document. This is a traditional information retrieval approach, and it can be quite effective, but it can also lead to unusual results. (For example, we tried this feature in Lycos and looked for related sites to ‘‘Accumulated Accordion Annotations’’ in the ‘‘Entertainment & Leisure: Music: Instruments: Individual Instruments’’ category. The second match was ‘‘Comprehensive Epidemiologic Data Resource WWW Home Page’’ in the ‘‘Health & Medicine: Medical Research: Libraries, Databases & Indices’’ category. The similarity was the word ‘‘accumulated.’’ The third match was in the ‘‘Space & Astronomy: Image Files & Archives’’ category and it apparently resulted from sharing the word ‘‘annotations.’’) The idea behind WebGlimpse is not only to utilize neighborhoods, but to allow search with additional keywords on these neighborhoods. In the ‘‘accordion’’ example above, we may want to search pages related to accordion annotations and also related to jazz. With WebGlimpse we would have to input only ‘jazz.’ Such a query would probably have filtered out the unrelated documents that somehow ended in those neighborhoods.

Another example of focusing search to limited domain based on current browsing is the WWW-Entrez system [7] from the National Center for Biotechnology Information, which precomputes neighborhoods within the nucleotide database and within the Medline database and presents fixed links from all pages to their neighborhoods. The determination of sequence and text neighborhoods for the millions of records in the database is computationally intensive requiring weeks of CPU time. Like the Lycos and InfoSeek examples listed above, Entrez allows one to quickly explore a neighborhood from any given document, but it does not provide search within neighborhoods. We are told that they are considering adding this feature.

The next section describes the design and implementation of WebGlimpse. Section 3 presents applications of WebGlimpse, and Section 4 ends with conclusions and further work.

2. WebGlimpse Design and Implementation

2.1. Overall Architecture

In a nutshell, WebGlimpse works as follows. At indexing time, it analyzes a given WWW archive (e.g., a whole site, a collection of specific documents, or a private history cache), computes neighborhoods (according to user specifications), adds search boxes to selected pages, collects remote pages when relevant, and caches those pages locally. Once indexing is done, users who browse that site can search from any of the added search boxes and limit their search to the neighborhood of that page (or search the whole archive). In a sense, WebGlimpse transforms the archive into an easier-to-navigate hypertext. As its name suggests, WebGlimpse uses our Glimpse search engine, which was modified slightly to add a few features useful for WebGlimpse. Only one index is needed for each archive (e.g., for each site). The focus to neighborhoods and the collection of remote pages are done in an efficient way at indexing time, so search is always fast.

WebGlimpse consists of several programs which perform five main steps. The first four are performed by the server (or publisher) administrator to set up an archive, and the last one is the actual user search of an existing archive. Their main features are given below, followed by more detailed descriptions.

Analysis of a given archive

Starting with a set of *root* URLs, this stage traverses local and remote pages[2] reachable by a path of links of a given maximum distance from the initial set. The links contained in each page are extracted, and their corresponding pages are then followed. The end result of this stage is a full graph of the whole archive, where the edges of the graph are the HTML links. The limit on the length of the traversed path can be set differently for local and remote pages. For example, one can allow unlimited distance on the local pages, but only a distance of 2 at any remote site.

Collection of remote documents

Non-local URLs are fetched and saved in a mirror file system. This is an optional step. Sometimes, local archives can be nicely complemented with data from remote sources. For example, with WebGlimpse one can collect in an archive a list of ‘‘favorite pages,’’ or simply one’s bookmarked pages. The links from these pages, and in general their structure, are preserved. This mirror file system can serve as a ‘‘hypertext book’’ collected from the web.

Neighborhoods computations

Depending on how neighborhoods are defined, this step (which in practice is interleaved with the first step) builds all the neighborhood files to help with the search. We will discuss neighborhoods in detail in Section 3.

Addition of search boxes to selected documents

Selected documents with non-empty neighborhoods are identified and modified by an addition of an HTML code which provides the search facilities. It is possible to define which pages will be selected in a flexible manner.

Search

Glimpse is used for all the search routines.

We’ll start with a description of Glimpse, because it serves as the basis for the whole design.

2.2. Glimpse

The search engine for WebGlimpse is *glimpse* [8] (which also serves as the default search engine in the Harvest system [9]). For completeness, we’ll mention here the features of glimpse that are relevant to the searching/browsing problem. Glimpse is an indexing and search software, designed slightly different than most other indexing systems. Glimpse’s index consists essentially of two parts, the word file and the pointers file. The word file is simply a list of all the words in all documents, each followed by an offset into the pointers file. The pointers file contains for each word a list of pointers into the original text where that word appears. A search typically consists of two stages (some searches can do with only one): First, the word file is searched and all relevant offsets into the pointers file are found. The relevant pointers (to the source text) in the pointers file are collected. The second stage is another search, using agrep [10], in the corresponding places in the original text. This is similar in principle to the

[2] We will use the term *page* to denote an HTML document corresponding to one URL.

usual inverted indexes, except that the word file, being one relatively small file, can be searched sequentially. This allows glimpse to support very flexible queries, including approximate matching, matching to parts of words, and regular expressions. These flexible queries are implemented by running agrep directly on the word file. The fact that the files are searched directly allows the user to decide on-the-fly how much of the match to see. Glimpse's default is to show one line per match (as in grep), but it can also show one paragraph or any user-defined record. This gives context to every match.

The second advantage of this design is that the pointers file can be built in many different ways. In particular, the granularity of the pointers — the precision of where they point to — can be set arbitrarily. The pointers can be to the exact locations of the words, which is similar to regular inverted indexes, to the documents (files) containing them, to whole directories, etc. The larger the granularity the more work will need to be done in the second stage (where the source is searched directly), but the smaller the index will be. Glimpse supports three types of indexes: a tiny one (2-3% of the size of all files), a small one (7-9%), and a medium one (20-30%). In the medium-size index the pointers point to exact locations, in the small one the pointers are just file names, and in the tiny one the pointers are to blocks of files.

The pointers file has another feature. Its pointers are indirect. They are indexes to yet another file, called the *filenames* file, which contains the list of all indexed files. To support WebGlimpse better, we added an option to glimpse to store with each HTML file name its title and, if relevant, its URL. WebGlimpse obtains this information directly from the result of a glimpse search. A typical search first gets offsets into the pointer file, from there it gets the indexes to the filename file, from there it collects the file names (and possibly the titles and URLs), and then in the final stage it searches those files directly. The performance of glimpse depends on the complexity of the queries and the size of the index. Simple queries with words as patterns and Boolean operators is optimized by using hashing into the word file. Such queries generally take less than a second. More complex searches obviously take more time, but it is worth the extra flexibility. GlimpseHTTP was compared with WAIS [11] (independently of us), with the conclusions that ‘‘system requirements, administration and maintenance and, most importantly, user functionality is dramatically better than WAIS for both searching and browsing.’’

Glimpse is particularly suitable for our purposes because it supports very flexible ways to limit the search to only parts of the information base. Since the search is divided cleanly into two stages, searching for the words, and then going to the appropriate files, we can filter files in the second stage. This is done through two options in the search: –f and –F (the former added specifically for WebGlimpse). The first one reads a list of file names from a given file and uses only those files. The second one uses the full power of agrep to filter file names by matching. We'll describe later how WebGlimpse uses these options to limit the search.

2.3. Scanning the Archive

A WebGlimpse archive is built with a script called *confarc*. The script first sets up the right paths to the archive, the URL of the cgi-bin programs, the title of the archive, and other administrative details. More important parameters to confarc are the initial URLs from which the traversal of the archive will begin, whether or not non-local URLs should be collected, and the definition of neighborhoods. The traversal of the archive is done in a straightforward way. Each HTML document is analyzed to extract all its outgoing links. All links are checked for locality (using IP numbers), and traversed using depth-first search (we may change that to breadth-first-search in later versions). WebGlimpse provides flexible facilities, through the use of regular expressions, to define which URLs/files should be included or excluded from the archive. For example, one may decide to exclude from the archive any files in a directory called /private/, except for one file there called ‘‘told-ya.html’’. The syntax for exclude/include follows the one used by the Harvest system [9].

2.4. Collecting Remote Pages

WebGlimpse provides facilities to collect remote URLs and include them in the indexing and in the search. A neighborhood can therefore include remote URLs, and users may jump through a neighborhood search to remote sites. We believe that such a facility is very important, and is sorely missing from local search facilities. Including remote pages in the search can help users make connections that would have been much harder otherwise. If you are looking for ‘network research’ you may find that someone is working on, say, stochastic analysis and has a remote link to references about network research. Or if you are looking for military accidents involving computers, you may find a reference to a helicopter incident with remote links to discussions on the computers aboard. This feature makes the search a global hypertext search.

The remote URLs that WebGlimpse collects are saved in separate files with a mapping mechanism from URLs to file names so the indexing and search can be done transparently. The original content of these URLs is not discarded. If space is a problem, however, then these mirror files can be removed, and the search will still be possible, because they have been incorporated into the (small) index. Glimpse has been modified to take this into account and provide a limited search (e.g., without showing the matching lines, which are not there anymore) when the original content is not available; it will still show the original URL (which is actually the typical way to show results on the web). We originally did not allow a recursive collection of remote URLs, so that WebGlimpse would not be used as a robot. However, based on user feedback, and the fact that a multitude of robots already exist and are very easy to deploy, we now allow arbitrary remote collection. WebGlimpse abides by the robot exclusion rules.

2.5. Neighborhoods

When WebGlimpse traverses an archive, it also computes neighborhoods. The current version supports two types of neighborhoods. The first consists of all pages within a certain distance (the default is 2) of the current page. The idea is that WWW pages are often written with links to related pages, so the neighborhood concept is implicit. The second type consists of all subdirectories (recursively), similarly to the way GlimpseHTTP operates. The implementation of neighborhood search is quite simple. The list of all file names (recall that even remote URLs are mapped to local file names) that constitute a neighborhood of a given page is kept in one file (whose name is mapped to that page).

When a search is triggered, glimpse first consults the main index, finds the list of files with relevant matches, then *intersects* that list with the neighborhood of the given page. The neighborhood list is fetched *at query time*, and the index does not depend on it in any way, which allows easy access and easy modification if needed. For example, if a certain file or directory becomes irrelevant for some reason, all one has to do is delete its name from all neighborhood files (or from some).

In the current working version of WebGlimpse we added a compression of the neighborhood files to save additional space. This compression, which is especially designed to work with the glimpse index, is computed at indexing time through an extra program. The compressed neighborhood files are used directly with the glimpse search, so their decompression on the fly during the search is very quick. WebGlimpse includes the compression and decompression programs, so if one wants to change any of the neighborhoods or generate them through another program, it can be easily done. We expect people to write scripts for generating neighborhoods, and this design makes such scripts completely independent of WebGlimpse itself. They can be run as post-processing steps. The neighborhood lists can further be compressed (and decompressed on the fly) to save space if that becomes a concern.

We are working on other types of neighborhoods we well as on general tools to allow people more flexibility to define their own types. One idea is to allow several neighborhoods for one page so that users can decide which one to use at query time. There will still be one neighborhood file per page, plus sets of ranges into it (e.g., smallest neighborhood contains files 1-23, second one contains files 1-39). Using classification or clustering tools, one will also be able to define a neighborhood as ‘‘all pages in this site that are in the same area.’’

2.6. Search Boxes

To further integrate browsing and searching, WebGlimpse automatically adds small search ‘‘boxes’’ to selected pages. An example of such a search box is shown in Figure 1. The HTML code for including the box is given in a template file; it can be easily changed to fit the preferences of the archive maintainer. The boxes are added at indexing time by adding HTML code (essentially a FORM field) to the original pages. The boxes are added at the bottom of pages, and special markers (implemented as HTML comments) are also added so that the boxes can be easily removed (WebGlimpse includes a program to do that). It is possible to customize where the boxes are added by simply moving these markers. The default is to add the boxes to all HTML pages with non-empty neighborhoods. The same kind of exclude/include facilities are available for selecting these pages. (Obviously, if a page is excluded from indexing no search box is added to it, but the opposite may not be true in some cases.) The typical approach on the web is to provide a link to a separate search page. We decided to add boxes to all pages so that users do not need to go anywhere else for search. This minimizes the ‘‘context switching’’ and keeps users focused. They can see the content of the page while they compose the query.

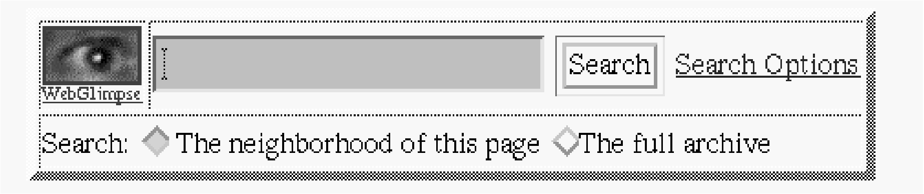
Each box also includes a dynamic link to advanced search options. It’s dynamic in the sense that it is generated on the fly and can also display the neighborhood (also generated on the fly). The search options interface is shown in Figure 2. The advanced options include old glimpse options like ‘‘case sensitivity,’’ ‘‘partial match,’’ and ‘‘number of spelling errors,’’ as well an option to jump directly to the line of the match (more on that in the next section). A very nice new option allows users to search only files updated within the last X days. The downside of adding boxes is that pages are modified, which some users may object (especially in a shared environment), and if pages are printed there is a little extra to print. We believe that in most cases this is still worthwhile. (Of course, it is easy to modify the box to make it just one link to the advanced search.)

2.7. The Output

The output of a query is a set of records, one for each matching file. WebGlimpse formats the results in four ways:

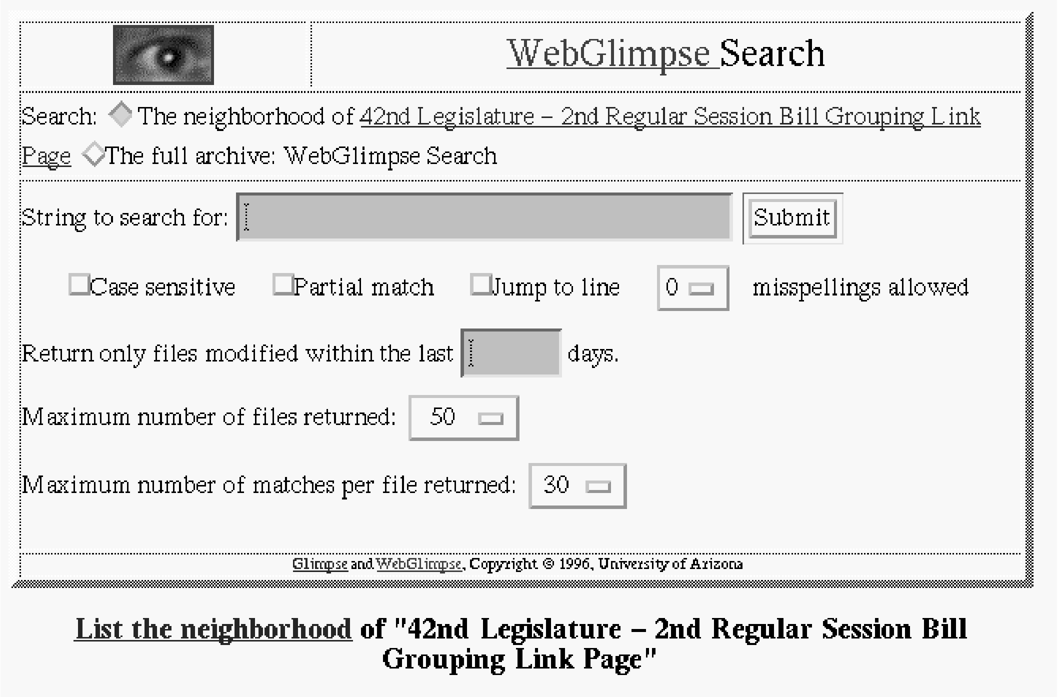
Context for each match

WebGlimpse outputs the title of each matching URL with a link to it (if the file name corresponds to an HTML document), and, since glimpse provides the matching lines or records and not only the file names, all the matching lines or records are output too. In general, the records provide quick-and-dirty context for matches.



Search: ◆ The neighborhood of this page ◇The full archive

Figure 1: WebGlimpse's search box



Search: ◆ The neighborhood of 42nd Legislature – 2nd Regular Session Bill Grouping Link Page ◇The full archive: WebGlimpse Search

String to search for: Submit

□Case sensitive □Partial match □Jump to line 0 misspellings allowed

Return only files modified within the last days.

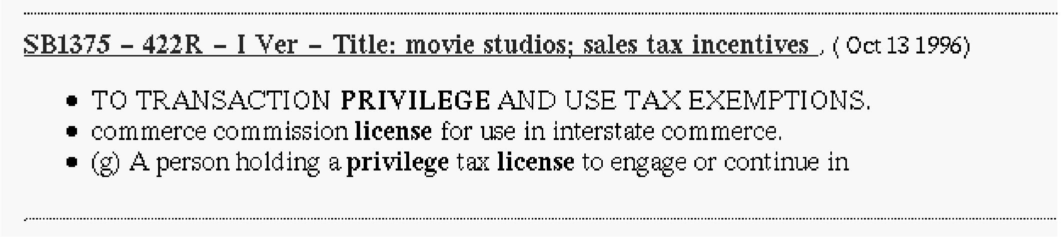
Maximum number of files returned: 50

Maximum number of matches per file returned: 30

Glimpse and WebGlimpse, Copyright © 1996, University of Arizona

List the neighborhood of "42nd Legislature – 2nd Regular Session Bill Grouping Link Page"

Figure 2: WebGlimpse's search options



SB1375 – 422R – I Ver – Title: movie studios; sales tax incentives , (Oct 13 1996)

- TO TRANSACTION **PRIVILEGE** AND USE TAX EXEMPTIONS.
- commerce commission **license** for use in interstate commerce.
- (g) A person holding a **privilege** tax **license** to engage or continue in

Figure 3: An example of WebGlimpse's output (searching for ‘‘privilege and license’’)

Providing line numbers

Glimpse can compute the right line number for each match, and WebGlimpse has an option (borrowed again from GlimpseHTTP) to bring the documents automatically to that line number. This is done by modifying the HTML document on the fly to insert the corresponding anchor, which is not trivial because some links may need to be recomputed as well.

Highlighting keywords

All the matched keywords are highlighted (formating them to bold in HTML), both in the output records and, in case line numbers are used, in the files themselves if the links are followed.

Showing dates of modification

Starting at version 3.5, glimpse can provide dates for each file, filter by dates, and show them. WebGlimpse uses these features.

An example of an output (one match of a search for ‘‘privilege AND license’’ from our demo of the Arizona legislature pages) is given in Figure 3. (The date refers to our copy, not the original document.)

2.8. Experiments and Experience

Our experience with WebGlimpse is still quite limited. We give here some conservative numbers that we obtained, using a very complex archive. All experiments were run on a DEC Alpha 233 MHz with 64MB of RAM.

The archive for this experiment was the pages of the Arizona Legislative Information System, which includes information about budget, committees, statutes, the constitution, floor calendars, agendas, and (the most space consuming) full text of bills. This archive occupies 152.5 MB and about 20,000 files. The number of links was quite high, and we selected the neighborhoods to be within 3 hops of the given page. (In hindsight, this is too much. The neighborhood files became too big and search was not sufficiently restricted; the indexing stage was also too slow as a result. But it gives a worst-case scenario.) We selected this archive for its complexity and the multitude of links.

The complete indexing process took 3 hours and 5 minutes. It took about an hour to traverse the whole archive and compute all the neighborhoods, an hour to analyze all HTML files and add the search boxes (we added search boxes to all files that had non-empty neighborhoods — about 85% of all files), 22 minutes to index the whole thing, and 41 minutes to compress all the neighborhood files. (We should mention that the glimpse indexing and compression is done with C, and everything else is done with Perl.) We believe that this is indeed close to the worst case, because the neighborhood structure was so complex (some neighborhoods, especially ones from the floor calendar files, contained thousands of links). The archive after indexing occupied 205.8 MB, a 34% increase, which was divided about equally between the index, the (compressed) neighborhood files, and the added search boxes.

Query times depend heavily on the type of queries. We run a few experiments with typical queries. As expected, we found no significant difference between a whole archive search and neighborhood searches. The running times, measured from the time a user clicks on the search box to the time the results page appears on the browser, range from 2 seconds for a query that matches less than 10 hits, to 12 seconds for a complex query with many hits. The corresponding times for pure glimpse search are less than half that; the rest of the time is taken to compose the HTML result page.

These numbers are taken from an early version of WebGlimpse. We put most of our efforts into making WebGlimpse work in a flexible manner. We expect the performance numbers to improve as we tune the code. More information will be posted to the WebGlimpse home pages (http://glimpse.cs.arizona.edu/webglimpse/).

2.9. Design Decisions

We discuss here briefly the rationale behind the major design decisions we made. As we said in the abstract, our main four goals were fast search, efficient indexing, flexible facilities for defining neighborhoods, and non-wasteful use of Internet resources.

Our first design decision was to have fixed neighborhoods, computed at indexing time. There has been a lot of discussion lately of search *agents* that will traverse the web looking for specific information. While the ability to explore in real time is very attractive, we believe that at the moment the web is too slow and the bandwidth is too narrow to support wide deployment of a large number of such wandering agents. WebGlimpse can be thought of as an agent for servers, but users still search fixed indexes residing in one place. This makes the search much faster, but it limits its flexibility. In particular, neighborhoods in WebGlimpse cannot be defined by the users. It is possible, however, for the server maintainer to provide several different neighborhood definitions and to let the users choose between them. We have not yet implemented such an option.

The ability to define any neighborhood one wants is important. We put emphasis in the design to unlink the neighborhood definition and use from the rest of the system as much as possible. The neighborhoods files are consulted only at the end of the search process, and they are not integrated into the index in any way.

Fast search always conflicts with efficient indexing. The more you spend on indexing, especially more space, the faster the search. WebGlimpse was designed for small to medium-sized archives, and, like glimpse, it puts indexing efficiency slightly above search speed. Nevertheless, the search is quite fast, and indexing can sometimes take quite a long time. Indexing can be slowed down by two features: 1) having to fetch many remote documents (nothing we can do about that), and 2) having to compute and manipulate complex large neighborhoods.

User interface is very important to any search application. We believe that our design of the output of queries — with the inclusion of matched lines, highlighted keywords, and dates of modification — will be very helpful. Being able to quickly judge the relevance of the output to one's query is a problem in many search services today. One often finds oneself spending considerable time following up on the multitude of links that are returned as results of queries.

We believe that making the search box available all the time, rather than the more typical link to a different search page, is a good idea. A search box does not take much ‘‘real-estate,’’ and it allows users to compose their search query *while* looking at the page. We received comments from web administrators who do not want to modify existing pages to add the search boxes. This is a valid concern, especially in sites where pages are owned by many people. But many sites strive for coherent design and adding boxes is not much different than requiring a certain format or adding uniform links to the main search page. We provide easy tools to add and remove those search boxes at any time.

3. Applications

The main application of WebGlimpse is, of course, to provide search for collections of hypertext files. We foresee several other related applications.

3.1. Building Personal or Topical Collections

In a sense, WebGlimpse is a ‘‘light’’ version of Harvest [9]. It does not have the full power of Harvest to automatically collect massive information from given sites and extract the indexed text from different formats. But it does allow one to collect and organize specific documents that are relevant. In contrast with Harvest, the link structure of that information is kept. (Harvest, like most global Internet search facilities moves everything it collects into a flat structure.) Hypertext ‘‘books’’ can be written much more easily with WebGlimpse, which will collect and index all links (citations) and allow flexible browsing through the whole book. Lists of ‘‘interesting sites’’ are commonly kept by many people, some are quite substantial. WebGlimpse will allow the maintainers of such lists to easily make them searchable with full-text search.

3.2. History and Cache Files

Most web browsers cache the pages they fetch. Like typical caches, this is done completely transparent to the user, and is done for performance purposes only. But having this large set of mostly relevant pages can be very helpful. For example, two years ago we designed a system called *Warmlist* [12] that cached pages per user demand, indexed them, organized them, and provided full text search. It was meant to be a natural extension of the ‘‘hot list’’ concept. There are now similar commercial products, which also allow to cache everything automatically and to view and navigate based on this cache. This becomes a history feature more than just a cache. With WebGlimpse, you can easily construct an *archive* of your history list. Not only can you browse and search it, but you may also discover relationships between pages — by viewing the neighborhoods — or other context information.

3.3. Visualization and Customization

Combining WebGlimpse with graph drawing packages (such as [13] and [14]) will allow for better visualization of the hypertext structure. Imagine adding to the results of queries some summaries of the documents, icons of them, or other useful information of the kind you find in static pages with links to related documents. When you visit a page and perform a query (and queries with WebGlimpse can be simpler because the context is kept and the domain of the search is smaller) you get a customized view of the ‘‘way ahead’’. This view may be just as good in terms of information as the static view, but much more relevant to you. In other words, you as the navigator can build your own hypertext part of your way, customized to the areas of interest to you. In particular, it would be very interesting to see how to combine our fixed neighborhood search with the ideas of scatter/gather [15], or an automatic classification system like AIM [16].

4. Conclusions

Searching the web is growing quickly from an infancy stage. The current facilities are quite amazing and very useful, but they are far from being the last word or even a good word. Finding useful information on the web is still a very frustrating process. We believe that more methods need to be attempted, more prototypes employed and experimented with, and more paradigms need to be explored. WebGlimpse presents one such attempt. It is simple, easy to build, natural to follow, and flexible so it can be extended. WebGlimpse is part of the FUSE (Find and USE) project at the University of Arizona (http://www.cs.arizona.edu/fuse/), where we are working on other methods for the same problems.

References

[1] The Home Page for GlimpseHTTP, http://glimpse.cs.arizona.edu/ghttp/

[2] Glimpse Search of Computer Science Bibliography, http://glimpse.cs.arizona.edu/bib/

[3] A partial list of sites using GlimpseHTTP, http://glimpse.cs.arizona.edu/ghttp/sites.html

[4] Yahoo search, http://www.yahoo.com/

[5] InfoSeek Search, http://www2.infoseek.com/

[6] Lycos Internet Directory, http://a2z.lycos.com/

[7] Epstein J. A., J. A. Kans, and G. D. Schuler, ‘‘WWW Entrez: A Hypertext Retrieval Tool for Molecular Biology,’’ *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois (October 1994).

[8] Manber U. and S. Wu, ‘‘GLIMPSE: A Tool to Search Through Entire File Systems,’’ *Usenix Winter 1994 Technical Conference*, San Francisco (January 1994), pp. 23–32. See also Glimpse Home Pages at http://glimpse.cs.arizona.edu/

[9] Bowman C. M., P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, ‘‘The Harvest Information Discovery and Access System,’’ *Computer Networks and ISDN Systems*. **28** (1995), pp. 119–125.

[10] Wu S., and U. Manber, ‘‘Fast Text Searching Allowing Errors,’’ *Communications of the ACM* **35** (October 1992), pp. 83–91.

[11] Morton D., and S. Silcot, ‘‘Systems for providing searchable access to collections of HTML documents,’’ *First Australian WWW conference,* July 1995. (http://www.scu.edu.au/ausweb95/papers/ indexing/morton/)

[12] Klark P., and U. Manber, ‘‘Developing a Personal Internet Assistant,’’ *Proceedings of ED-Media 95, World Conf. on Multimedia and Hypermedia*, Graz, Austria (June 1995), pp. 372–377.

[13] Ayers, E. Z. and J. T. Stasko, ‘‘Using Graphic History in Browsing the World Wide Web,’’ Technical Report GIT-GVU-95-12, Georgia Inst. of Technology (May 1995).

[14] Domel, P., ‘‘WebMap - A Graphical Hypertext Navigation Tool,’’ *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois (October 1994).

[15] Cutting D. R., D. R. Karger and J. O. Pedersen, ‘‘Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections,’’ *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1993), pp. 126–134.

[16] Barrett R. and T. Selker, ‘‘AIM: A New Approach for Meeting Information Needs," IBM Technical Report RJ 9983 (October 1995).