Information Retrieval in the World−Wide Web: Making Client−based searching feasible

dr. P.M.E. De Bra, debra@win.tue.nl
drs. R.D.J. Post, reinpost@win.tue.nl
tel. +31 40 472733
Eindhoven University of Technology
Dept. of Computing Science
PO Box 513
5600 MB Eindhoven
the Netherlands

Abstract

Finding specific information in the World−Wide Web (WWW, or Web for short) is becoming increasingly difficult, because of the rapid growth of the Web and because of the diversity of the information offered through the Web. Hypertext in general is ill−suited for information retrieval as it is designed for stepwise exploration. To help readers find specific information quickly, specific overview documents are often included into the hypertext.

Hypertext systems often provide simple searching tools such as full text search or title search, that mostly ignore the 'hyper−structure' formed by the links.

In the WWW, finding information is further complicated by its distributed nature. Navigation, often via overview documents, still is the predominant method of finding one's way around the Web.

Several searching tools have been developed, basically in two types:

- A gateway, offering (limited) search operations on small or large parts of the WWW, using a pre−compiled database. The database is often built by an automated Web scanner (a "robot").
- A client−based search tool that does automated navigation, thereby working more or less like a browsing user, but much faster and following an optimized strategy.

This paper highlights the properties and implementation of a client−based search tool called the "fish−search" algorithm, and compares it to other approaches. The fish−search, implemented on top of Mosaic for X, offers an open−ended selection of search criteria.

Client−based searching has some definite drawbacks: slow speed and high network resource consumption. The paper shows how combining the fish search with a cache greatly reduces these problems. The "Lagoon" cache program is presented. Caches can call each other, currently only to further reduce network traffic. By moving the algorithm into the cache program, the calculation of the answer to a search request can be distributed among the caching servers.

Introduction

Before turning to the subject of the matter, we would like to apologize for the structure of this document. As it is intended to be a conference paper we have concentrated this hyperdocument into a single document. This makes it easy to print. Hyperlinks are present for online reading.

The World−Wide Web (WWW or Web for short) is a fast growing wide−area hypermedia database. It contains information on many related and unrelated topics. Finding specific information on a certain topic by simply browsing through the Web seems almost impossible. The navigation metaphor, typical for hypertext, is ill−suited for information retrieval. Hyperdocuments located on a single site can often be searched by ignoring the link structure and applying full text search like with plain text databases. However, a hyperdocument like the WWW, distributed over hundreds or even thousands of loosely coupled sites, can only be searched by retrieving documents and scanning them for relevant information, and also scanning them for extracting links pointing to other documents.

In order to make it possible to find information in the WWW several types of solutions have been proposed and implemented:

1. specially designed meta–hyperdocuments, dividing the WWW (and possibly other information bases) into sections, grouped by topic, and providing different types of search operations on selected databases; some examples:
   The Global Network Navigator, EINet Galaxy, CERN's Virtual Library, The Mother–of–all BBS, ALIWEB, the GNA Meta–Library index;
2. robot–based searchable indexes on (almost) the entire WWW, such as: the World Wide Web Worm, the JumpStation, NorthStar and CUI's W3 Catalog);
3. client–based search tools, most notably our fish search, added to Mosaic for X.

As these tools are constantly changing, the difference between them often fades. Some of the meta–hyperdocuments are beginning to look more and more like the robot–based searchable indexes and vice versa. However, the first and second category differ significantly from the third, in that they use a (database) server which performs the computation of the answer to an information request, while the client–based tools perform the computation on the end–user's machine. The kinds of requests that are handled by the information servers are completely determined by the designers and implementers of these service providers. Also, the completeness, correctness and performance of this calculation depends solely on the providers. While the use of the Internet is generally based on a system of flat fees, it is unlikely that large and frequently used database servers such as the ones mentioned above will remain operational for a long time on a voluntary basis and without charging the users.

The fish–search algorithm (and its implementation)

Searching for information "on the fly" requires three types of actions:

1. Perhaps the most difficult part is the first step: finding a good starting point for the search. Two possibilities are to either use a "well–connected" document or to use the answer from a (robot–based) index database.
2. Documents are retrieved (over the Internet) and scanned for relevant information at the receiving (client) end.
3. The retrieved documents are scanned to find links to (URL's of) other WWW–documents.

The effectivity of such a search–algorithm depends largely on the ability to find relevant documents and to avoid downloading irrelevant documents. Also, the algorithm must penetrate a significant part of the WWW as quickly as possible. The best possible result of the search–algorithm is a subset of the documents in the WWW that contain the requested information. In order to ensure a complete answer a scan of the entire WWW would be necessary, which would take a week or more. Typically, a few relevant documents can be found within minutes, provided a "good" starting point is chosen.

The navigation strategy

The heuristics for navigation, used in the search algorithm, are based on the experimental research by Joep De Vocht [DV94]. Several navigation strategies and aids were tried for browsing a number of hyperdocuments, including the structure of [SK89] and [VAJ92]. The two most important conclusions from that research are:

- The *depth–first navigation* strategy is optimal for discovering the structure of a hyperdocument (meaning discovering the largest number of cross–reference links), regardless what the structure of that hyperdocument is. This optimum is "stable", meaning that strategies that deviate a bit from a strict depth–first navigation still provide near–optimal results. Also, the more the strategy deviates from depth–first navigation, the less effective it becomes.
- From the different navigation aids suggested in literature, and implemented in systems, the *highlighting links* facility, which means marking links leading to nodes that were visited before, provides the largest improvement in browsing efficiency.

The schools of fish metaphor

The algorithm simulates a school of fish, breeding and searching for food. A school of fish generally moves in the direction of food. How long the fish live and how many children they produce depends on the amount of food they find. When fish deviate from the school they may survive if they find food. Their offspring will form a new school. However, if they do not find food they die. Fish who enter polluted water also produce few and weak children and die quickly.

In the fish−search each URL corresponds to a fish. When a document is retrieved the fish breeds a number of children, depending on whether the document is relevant (contains food), and also depending on the number of links (URL's) embedded in the document. Each time a relevant document is found the fish and its children become stronger. If a document is not relevant the fish become weaker and fewer children are produced. After following a number of links in a direction without finding a relevant document the search stops investigating that direction (the fish die). When retrieving a document takes a long time (when the transfer rate is low) this corresponds to a fish swimming in polluted water. Few documents from the same site will be retrieved in order not to waste too much time.

The implementation of the search

While the schools of fish metaphor describes many parallel processes the actual implementation is sequential. Parallel access over the Internet would improve the efficiency of the search for a single user, but the overall network resource consumption would be unacceptable. The fish are simulated by means of a list of URL's. The average number of children produced each time is called *width*. The number of steps that can be made (by a fish or its offspring) without finding a relevant document is called *depth*. Each time a relevant document is found a selection of the embedded links are added to the front of this list. This leads to depth−first navigation. When a document is not relevant its embedded links are added to the list after all the links marked as embedded in relevant documents. This deviation from depth−first navigation leads to a concentration of the attention on the neighborhood of relevant documents. Also, links to documents on different sites are prefered by the algorithm, thus ensuring a better spreading of the attention over the entire Web.

When adding URL's to the list, the algorithm checks whether the URL already occurs in the list. This has the same effect as the *highlighting links* navigation aid. An URL already in the list is not added a second time. However, it may be moved to the beginning of the list, if the new node it occurs in contains relevant information.

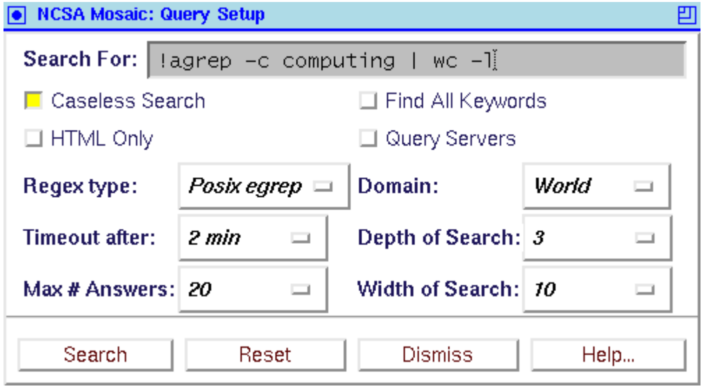
The fish−search implementation offers three types of search criteria:

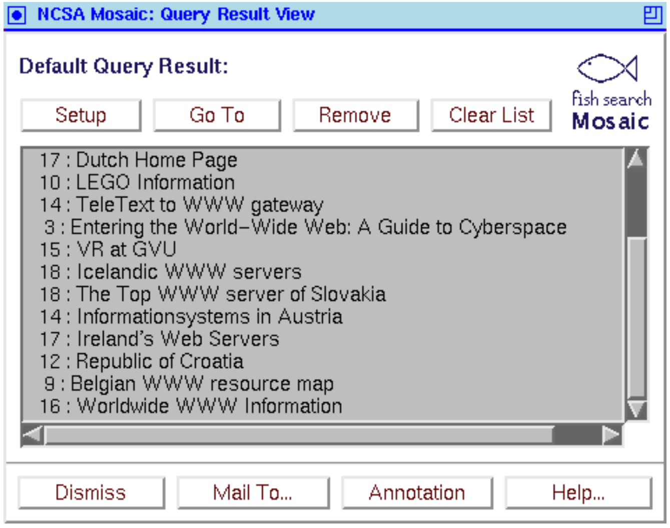
- **Keyword search:** the user specifies a number of (key)words, and indicates whether they must all appear in a document or not. The relevance of a document is measured by counting the occurences of the keywords, giving a bonus for words occuring together, and dividing the result by a factor that takes into account the size of the document.
- **Regular expression search:** the user specifies a regular expression (using one of many different syntaxes). The number of matches in a document is divided by a factor that takes into account the size of the document.
- **External filters:** the documents are passed to an external filter, which must return a relevance indicator (a number). Any standard or user−defined program can be used as a filter.

The result of a search is a list of (relevant) documents, with the "relevance score" of each document. Since the relevant documents may be totally unrelated, and far away from each other in the Web, alternative ways of presenting the result, such as fisheye views [GF86] or guided tours [GS92] were not considered.

The fish−search is implemented by integrating it into the most popular WWW browser: Mosaic for X (currently version 2.4). The latest version is always available from `ftp.win.tue.nl`. Apart from boolean combinations of (key)words, one can search for regular expressions, and one may also invoke external filters to determine the relevance of documents. This selection mechanism is much more flexible than the selections offered by index databases available on the WWW.

The figures that follow show the widget used for initiating a search (using an external filter in this case) and the widget with a possible answer to this query.





Limitations of the search

The fish–search has a few unfortunate properties:

- Retrieving documents through the Internet can be time consuming. Even when avoiding slow links, retrieving the first document through such a link may take an unacceptably long time.
- The use of network resources is sometimes considered unacceptably high. Compared to a (human) user retrieving documents and reading them the fish–search places a much higher load on both the network and the WWW servers. Some administrators have already announced that they wish to block access by "robots".
- The fish–search can only search documents for which links (URL's) are found in other documents. An increasing part of the WWW is reachable through forms or through clickable images (maps). Trying all possible inputs (like all possible coordinates in the image) would be too time–consuming. The information servers that are available on the WWW also do not contain information of documents "hidden" behind forms or clickable images.
- Because of its implementation as part of Mosaic for X, the fish–search is not available to users of other WWW browsers. Part of this problem can (and will) be solved by supplying a separate fish–search program, accessible through a form. However, separating the search from Mosaic makes it difficult or impossible to offer facilities like using one's hotlist as the starting point for the search.

In order to (partially) alleviate the performance problems we developed the cache described in the next section.

The Lagoon cache

A major success factor of WWW is the fact that a very wide range of information sources and services distributed all over the Internet are available in a very user–friendly way: there is a user–friendly, uniform interface, in which distribution is completely transparent.

Still, distribution affects the user–friendliness just because getting remote information can take so much time. If the same information is requested more than once, it makes sense to save a local copy. A **cache** makes copies automatically.

Another application of copying is to prepare for situations in which the remote documents are not available at all. For example, local copies can be used to give demonstrations of WWW, or WWW applications, in places where no live Internet access is available. The Lagoon program is regularly used for this purpose.

In this section, we first outline several ways of providing local document copies, outlining why the design used for Lagoon, a cache running with a local document server, is often more effective than alternative approaches. We point out that Lagoon is merely one of many similar programs developed for this purpose. We review update rules to decide what documents to copy and when to do it. Finally, we briefly mention the implications the use of a cache such as Lagoon has for its direct environment and for the Web in general.

Various ways of using local copies

The idea of using local copies can be realized in various ways. A spectrum of possibilities is presented by the following questions:

- when is the copy made;
- what documents are to be copied;
- who has access to the copies;
- in what way are the copies accessible.

When to make a copy

The decision when to make a copy can be made either by the end user or by a local WWW maintenance person. End users can make private copies of documents the moment they are viewing them. Maintainers can regularly browse the Web more or less haphazardly and copy some often requested documents.

Alternatively, the decision can be automated. Automated copiers basically exist in two flavors:

- the **mirror**: copying proceeds independently of usage, usually with a regular schedule;
- the **cache**: a document is copied (initially, or to refresh it) only when requested by a user.

The Web is much too large to be mirrored in its entirety. However, it is quite likely for a document, once requested, to be revisited soon. Consequently, we can say that mirroring is appropriate for small, selected parts of the Web, while a cache is much more effective overall. The two can be mixed, by prefetching often requested documents in a cache.

A mirror or cache program needs rules for when to refresh documents. A short review of refreshing policies for WWW, and the approach presently used in the Lagoon cache, is given later in this paper.

What documents to copy

When using automatic document copiers (mirrors or caches), it is of great importance to carefully select the documents to be copied. Especially so if network bandwidth, processing time or disk space are limited. The issue is less pressing for a cache, because the overhead involved is limited to documents actually requested by users, and is spread amongst individual document requests. We return to this matter when discussing refreshment policies for the WWW.

Who are the intended users of the copies

This is strongly related to the question who is responsible for making the copies. End users often copy isolated documents or document trees for personal use. Their WWW clients keep a cache of the documents retrieved in a session, but the cache is lost as soon as the session is terminated. Clearly, if a user often visits the same remote documents in different sessions, making these client caches persistent across sessions is worthwhile.

We can do better still if many users on a site access WWW regularly, by setting up a cache for the whole site. The pattern of WWW usage will be more varied amongst different users, but we may still expect a large amount of similarity between users of the same site. In this case, the cache (or mirror) will be maintained centrally by local WWW maintainers, and it needs to work on remote documents only.

In the WWW, this is easy to accomplish by setting up a cache as part of a WWW server. This is in fact the way most recent WWW caches have been implemented. Maximal portability of the cache software comes as an additional benefit: the cache is independent of the client software used. By designing it as a server script, it can even be made independent of the server software used. Our own Lagoon is an example of this setup.

Another consequence is the fact that the cache will automatically be available to remote users. This is especially useful for the purpose of concentrating network traffic on large, dedicated machines. In fact, access need not be limited to end users; network load can be further optimized if caches call each other. Existing Lagoon caches have already been doing this for some time.

From this we conclude that caching is best implemented as a server capability, for reasons of effectiveness, maintainability, and general flexibility. For mirrors, roughly the same considerations hold.

In what way are the copies served

Finally, there is an issue of how to present the copies. One approach is to make caching (or mirroring) fully transparent: users don't realize they are looking at copies. For most information in the Web, getting the most recent version is not critical, or cannot be guaranteed anyway, and the extra risk of serving documents that are out of date, that comes with a cache or mirror, isn't much of a problem.

A second approach is to install the copied documents on a separate space in local document space. This way, the copies constitute a separate part of the Web, that can be jumped into and be left again in the course of a browsing session. When caching a set of interlinked documents, the links must be translated to point to the cached copies. This approach is popular for use on small, well–defined document trees (such as online help for software). For a general cache, it is less common.

The Lagoon program can be used in this way. Its name was derived from the fact that, in a metaphor where documents are seen as fish, the cache forms a "lagoon" of documents "trapped" in local Web space.

Another way of describing the two approaches is by saying that document copies can be served either with the same identity (URL) or with a different one. However, the identity is not all that can be changed in a document copy. If it is important for users to be aware of the fact that they are using a copy, or if users even need to have control over the copying process, it is useful to insert a header into the copy, giving details about the copying process, possibly with a refresh button to retrieve the latest version, or other functionality. (Lagoon presently supports such a header.)

Lagoon as compared to other copying programs and techniques

In the previous section, several properties of the Lagoon program were mentioned that distinguish it from other approaches to copying:

- Lagoon is a cache: it is not filled by hand or by a robot, but a document is copied only if users make a request for it.
- Lagoon is intended to be used with all documents in principle, not for specific small document trees. To be more specific, it can be used for all documents served through the HTTP protocol; other documents, served with USENET News, Gopher, FTP or local files cannot be cached with Lagoon.
- Lagoon is a server script: it is a separate piece of software, to be used with a HTTP server. Therefore, it is independent of WWW clients, and reasonably independent of the server it is used with. To be

precise, Lagoon adheres to the Common Gateway Interface for server scripts.
- Lagoon can be used in two ways: as a "proxy gateway", serving copies as if they were the original, or as a device to create a local Web space of copied documents. Both methods have their pros and cons, which we shall not describe here. Copies can be served with headers to distinguish them from originals and provide user control over the copying process.

We are aware of several other programs to automate the task of making local document copies. Some, such as Oscar Nierstrasz's htget program, simply are command–line scripts to copy document trees, intended for use by end users or server maintainers. Some efforts provide caching as a built–in server capability: Guenther Fischer's server patch and the built–in caching support of CERN's HTTP server by Tim Berners–Lee and Ari Luotonen. Another server script to provide caching was written by Gert–Jan van Oosten.

Differences between these programs are quite small, and more programs could be added to this list. Our description is targeted at Lagoon simply because we are most familiar with it.

Deciding what and when to copy

The most imminent problem with caching, or other forms of keeping copies, is the decision on what to copy, and when to refresh it. For the World Wide Web, this certainly is a nontrivial problem. As existing caches are restricted to HTTP–served documents, let us consider the problem for those.

Serving a document copy instead of the original is acceptible if

- they are identical, or
- neither the user, nor the document provider relies on the latest version being served.

In the Web, a document is whatever is returned in response to a request (basically consisting of the document's URL, its identity). Many URLs identify immutable documents: their location and contents are unlikely to change for years. Many other URLs identify throw–away documents, generated on the fly, changing from request to request: a document giving the current time, a randomly chosen picture, a response from a search engine, an acknowledgement from an email composition form, and so on.

It is generally for the latter type that serving the latest version is of most importance, as the examples clearly show. But even for highly immutable documents, serving the latest version can be important: sometimes, a reader visits a document only to see if any changes have been made.

Generally speaking, we can take three approaches for a given document:

- not to cache at all;
- to cache 'safely', always returning the latest version;
- to take for granted the risk of serving an outdated copy.

How not to cache at all

The choice of retrieving a document directly from its source, or retrieving it through a caching server, generally is made in a Web client. The most popular approach is to simply redirect all HTTP requests to a HTTP server in which caching is installed. This eliminates the possibility of bypassing the cache for certain documents. If the client doesn't redirect requests to a cache automatically, and cached documents form a separate Web space, it is left to the document structure formed by the server and its cache program what documents will be referred to via the cache. In Lagoon, all links within a cached document point back to the cache space, so once inside the cache, the user will stay there.

How to cache safely

In a safe policy, we require that the cache always provides the latest versions of documents. Clearly, this can only done by contacting the original server every time a request for a document is made. Simply refetching the original document defeats the purpose of caching. To find a way out, extra information on documents must be available, for example, the time it was last modified, or a time limit during which the document is guaranteed not to change. Several schemes in this direction have been proposed and implemented in the Web. Some of these involve special, short requests designed to transmit information on a document instead of the document itself. Not all document servers presently provide this information or support these schemes.

Rules for ’unsafe’ caching

The Lagoon cache program was designed as a plug−in Web component, requiring no changes to existing software. As a consequence, it does not provide safe caching, but instead, uses heuristics and manual configuration to guess whether or not a cached document is out of date.

The cache maintainer manually provides some rules to decide on the moment a document is out of date. The rules presently refer only to document URLs, assigning to each of them a refreshment rate; other document information, such as the textual content, cannot be used. However, URLs provide much information that is of value in heuristics: the site at which the original document resides, often, the type of document (text, image, etc.), and whether or not the document is a response to a query. Sets of documents can often be identified by carefully selecting on the basis of the document ’path’ given in the URL. For ’uncacheable’ documents, the retrieval rate is set to 0.

Recently, special rules have been added, that attach to URLs not refreshment rates, but alternative sources from which to obtain the original documents. This is the way Lagoon caches can be configured to call each other and form cooperative networks of caches.

In all, we consider this approach to be acceptable, even though careful manual tuning of the rules is required, and the risk of presenting outdated copies of documents is not fully eliminated.

The implications of Lagoon for the Web

It has been mentioned that Lagoon was designed to work with existing client and server software, ready to be plugged in as they run. In order to achieve this, we decided to implement Lagoon as a server script. It was shown, however, that caching is necessarily ’unsafe’ if no specific provisions to the servers are made. Obviously, Lagoon can be taught to take advantages of changes in clients and servers designed to support caching as they are developed in the Web community. But it will remain to be usable without imposing special requirements on the Web surrounding it.

Generally speaking, the use of caching in the Web is likely to increase with its size. Without caching, network traffic would have to increase much faster than the number of Internet users and information providers, in order to keep up the same performance. Lagoon is already equipped with the possibility of forming networks of cooperating caches. However, present software such as Lagoon is unlikely to be sufficient for the huge future demands for caching.

Fish searching in a Lagoon: automated browsing and caching combined

As their name suggests, the "fish search" automated Web browser and the "Lagoon" cache work best in combination.

Benefits of a cache for automated browsing

The automated browser retrieves only text documents, many of which are small. For small documents, most of the effort retrieving them is spent establishing the remote connection, and caching is maximally effective.

At our own site, which has excellent Internet connections, speed improvements of a factor 10 are common. Of course, improvements are only realized if the automated browser is used on a part of Web space that has been browsed previously, either manually or automatically.

While caching may substantially increase browsing performance, the "fish" algorithm will always remain a tool for selective exploration of small parts of the Web. It will never compete in speed or exhaustiveness to any of the index−based search engines.

Benefits of automated browsing for a cache

An automated browser can be a useful tool in maintaining a cache. The browser can be used by cache maintainers, or even end users, to prefetch parts of Web space into the cache in a semi−automatic way. Typically, the browser is started on an initial document, then restarted on some of the resulting documents.

This is a much faster method of filling a cache than manual browsing. Compared to fully automatic robots on the other hand, the automatic browser allows human monitoring and intervention, which avoids spending time on irrelevant retrievals. The lazy user is free to turn on a long search and review the results after lunch; apart from the fact that it clogs the network, this method is not likely to produce good results.

Tighter integration

Currently, the fish algorithm and the Lagoon cache are completely separate programs. Their performance can be further improved by closer integration.

A long−awaited step, still to be made, is to build in the fish algorithm into the cache program. In recent months, it has become possible to dispose of client modifications altogether, by using a forms based interface for the specification of the fish algorithm parameters.

An additional possibility created by integration is to distribute automated browsing sessions over Lagoon servers, thus bringing the fish algorithm closer to its original idea of parallel execution.

Finally, by maintaining indexes over the contents of Lagoon caches, our system can provide the fast search capabilities of existing index−based search engines (World Wide Web Worm, JumpStation, NorthStar).

Authorization matters

Especially with the advent of commercial activity on the Internet, authorization issues are starting to matter. Many documents are supposed to be available only to certain users, or under certain conditions. The WWW is beginning to support several authorization protocols designed to deal with this situation. Caching can complicate this situation.

First of all, a cache must not break authorization, by making freely available copies of restricted documents, for example. Some caching initiatives do pay special attention to providing support for authorization protocols.

Secondly, it turns out that some document providers are happy to make their documents freely available to casual human browsing, while at the same time strongly objecting to retrieval by automatic browsing or copying programs. Fortunately, simple technical means exist to stop automatons from visiting such documents, and the fish/Lagoon system will be made to support them.

Conclusions and Future Work

Client−based search mechanisms can be very useful for finding a (limited) number of documents in the World−Wide Web, satisfying a condition which can be completely specified by the user. The "fish−search", integrated into Mosaic for X (release 2.4 at the moment), finds relevant documents by browsing (automatically) through the Web, using a navigation strategy which has been verified to be efficient.

Usually a user, or a number of users on the same site, will initiate a search from a common home page, or from answers provided by an index database (such as the JumpStation or the World Wide Web Worm). These searches involve a large number of common documents. By caching these documents on the local site the performance of the fish−search can be improved a lot. Caching also reduces network traffic and the number of accesses to remote servers. The Lagoon cache has the ability to redirect accesses to a part of the Web to a common server (also running Lagoon). By letting the caches call each other, the network traffic can be concentrated into a limited number of well−connected sites.

In the (near) future we intend to incorporate the fish−search facility into Lagoon, thereby enabling users of other browsers than Mosaic for X to also benefit from the search, and thereby also enabling the caches to distribute the search process among each other.

References

References to paper documents

[MB85]
Bärtschi, M., *An overview of information retrieval subjects*, IEEE Computer, 18:5, pp 67–84, May 1985.

[GF86]
Furnas G.W., *Generalized fisheye views*, CHI'86 Conference, Boston, April 1986, pp 16–23.

[GS92]
Guinan, C., Smeaton A.F., *Information Retrieval from Hypertext Using Dynamically Planned Guided Tours*, 4th ACM Conference on Hypertext, Milan, December 1992, pp 122–130.

[DV94]
De Vocht, J., *Experiments for the Characterization of Hypertext Structures, Masters Thesis*, Eindhoven Univ. of Technology, april 1994.

[GS89]
Salton, G., *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison–Wesley, 1989.

[SK89]
Shneiderman, B., Kearsley G., *Hypertext Hands–On!: An Introduction to a New Way of Organizing and Accessing Information*, Addison Wesley, 1989.

[VAJ92]
De Vries, E., Van Andel, J., De Jong, T., *Computer–Aided Transfer of Information on Environment and Behavior: Network versus Hierarchical Structures*, 23rd Conference of the Environmental Design Research Association, April 1992.

Web locations

Some Web references follow; more appear within the paper. They can only be used directly in the online version.

The following is available on the Web regarding Mosaic with fish algorithm and the Lagoon cache program:

- sources and binaries;
- online help for the fish algorithm;
- a short paper on the fish algorithm;
- information about caching and Lagoon.