Information Retrieval in Distributed Hypertexts

Paul De Bra[1], Geert-Jan Houben[1], Yoram Kornatzky[2], Reinier Post[1]

1 : Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, the Netherlands,

debra@win.tue.nl, houben@win.tue.nl, reinpost@win.tue.nl

2 : Computer Systems Research Institute, Univ. of Toronto, 6 King's College Road,

Toronto, Ontario M5S 1A4, Canada,yoramk@db.toronto.edu

Abstract

Hypertext is a generalization of the conventional linear text into a non-linear text formed by adding cross-reference and structural links between different pieces of text. A hypertext can be regarded as an extension of a textual database by adding a link structure among the different text objects it stores. We present a tool for finding information in a distributed hypertext such as the World-Wide Web (WWW). Such a hypertext is a distributed textual database in which text objects residing at (the same and) different sites have links to each other. In such a database retrieval is limited to the transfer of documents with a known name. Names of documents serve as links between different documents, and finding such references names is only possible by parsing documents that have embedded links to other documents.

Full-text search in such hypertexts is not feasible because of the discrepancy between the large size of the hypertext and the relatively low bandwidth of the network. We present an information retrieval algorithm for distributed hypertexts, which does an incomplete search through a part of the hypertext. Heuristics determine the selection of the documents that are to be retrieved and searched. A prototype implementation for the WWW, on top of Mosaic for X, is being used by an increasingly large user base.

1 Introduction

A simple definition of hypertext (mostly taken from [9]) is *a database of (textual) information fragments (nodes) that has active cross-references (links) and allows the reader to "jump" toother parts of the database as desired*. Most hypertext systems have a limited view on the jumps the reader may *desire*. Links are hard-wired, and the only jumps that are allowed are following links, backtracking, and maybe also jumping to the first node or to a node that appears in a history list or a hotlist.

Full-text search and/or the creation of indexes are only feasible when the hypertext system controls the entire hypertext, e. g. when the hypertext is contained in a (local) database or in files in a specific directory (tree). Intermedia [11], HyperNews [1], Superbook [6] and many others offer querying and information retrieval on the entire hypertext. The answers can be a list of relevant nodes, possibly annotated by a relevance score, or a graphical overview of the hypertext structure in which the relevant nodes are highlighted. An interesting variation in [7] presents the result of a search as a guided tour, generated by using the relevance scores and the link types.

In this paper we concentrate on distributed hypertexts such as the World-Wide Web. Unlike the distributed hypermedia storage system described in [8], the computer systems carrying the parts of the World-Wide Web are only loosely coupled, somewhat like heterogeneous multidatabase systems (but without a database scheme and without a DBMS). Each site is completely autonomous, and offers a simple interface (protocol) to the other sites. Hypertext nodes (often called documents) have a unique name, which combines information on the site, the protocol to be used to access the node, and the pathname within the site. Such a name is called a *Universal Resource Locator* (URL). In general the only service one may expect from a site is that when it is given a URL of an existing document it will return that document.

Finding information in such a distributed hypertext can only be done by means of (automated) browsing. Assuming that the hypertext is connected and that a good starting point can be found it is theoretically possible to simply retrieve all the static nodes and determine their relevance to a given expression, set of keywords, or other query. However, the size of a distributed hypertext such as the World-Wide Web is such that, given the current wide-area network infrastructure, such a search would take days, if not weeks, depending on the network load and on the availability of all the sites. Also, since these large hypertexts contain information on a wide variety of subjects, most of the time would be spent retrieving irrelevant documents. Furthermore, the Web contains computed nodes, that change very often, possibly depending on arguments given in the command to retrieve them. Some attempts have been made to create databases that can be used for retrieving information in the World-Wide Web. However, these "indexes" do not contain all information of the World-Wide Web. They can be used to retrieve documents based on title, citations and other preprocessed material. Furthermore, the creation and updates to these databases take a *very* long time, hence they are always out of date.

We present an automated browsing algorithm that tries to find some of the nodes that are relevant for a given query. The heuristics of this algorithm are based on a number of experiments related to finding the optimal browsing strategy, depending on the time or number of nodes one can visit, on the browsing facilities offered by the hypertext system, and of course on the structure of the hypertext itself. Several databases and facilities exist to find *all* nodes that satisfy a certain *limited* condition. Our algorithm tries to find *some* nodes *quickly*, satisfying any kind of condition the user may want (and provides a filter for). The answer of a query is always incomplete and non-deterministic, and it depends on the starting point (node) given to the browser. The algorithm has been implemented on top of the popular WWW browser "Mosaic for X". An early prototype of this implementation, before the current heuristics were developed, was demonstrated at the 1993 ACM Conference on Hypertext.

2 Browsing Techniques

Browsing is mostly influenced by the following three factors :

- the structure of the hypertext;
- navigational aids (hotlist, history, bread crumbs, fish-eye views, etc.);
- the navigational strategy of the user (e. g. breadth first or depth first).

In [2] a method is given to reduce the structure of a hypertext to a hierarchy (or set of hierarchies) and a set of cross-reference links. The basic idea is that from the user's point of view a hypertext appears to have a hierarchical structure which is "disturbed" by some cross-reference links. In a (strict) hierarchy the reader is unlikely to get lost. The "amount" of disturbance generated by the cross-reference links is measured using two metrics, *compactness* and *stratum*. Compactness measures how many links one must follow (on average) to move between two arbitrary nodes. Stratum measures the amount of "reading order" in a hypertext.

In [3] navigation efficiency (or difficulty) is measured when taking into account different navigation aids offered by hypertext systems. Getting from one node to another becomes easier when history lists and/or hotlists are available. Marking nodes that have been visited before with so-called "bread crumbs" (from the fairy tale. . .), or marking links leading to them, helps avoiding going in the same direction twice. These facilities cannot simply be modeled by adding extra links to the hypertext in order to calculate their effect on the metrics or to count the number of cross-reference links they would generate. In [3] a large number of experiments (with simulated users) were conducted in order to find the influence of navigation aids on browsing.

The order in which the user visits nodes determines the number of links that must be followed in order to visit a given number of nodes, because links may have to be followed again or backwards in order to reach the desired nodes. Also, the order determines which links the user perceives as being structural (hierarchical) links and which links appear to be cross-reference links. In [3] different browsing strategies, ranging from breadth-first to depth-first have been evaluated. The hypertexts that were used in the experiments included those of the work of Erica de Vries [4, 5] and the book [9] by Shneiderman and Kearsley, as well as some artificially generated structures. The conclusion in [3], which is crucial for the algorithm we present is :

For all hypertext structures, and for all kinds of navigational aids or combinations thereof, the depth-first navigation strategy is most effective in finding cross-reference links and in traversing a hypertext in as few steps as possible.

The only exception to the above conclusion was that in very short browsing sessions, breadth-first navigation may be more effective in finding cross-reference links than depth-first. However, in this paper we assume that the (automated) browsing session is always long enough to benefit most from a depth-first strategy. In the next section we shall explain why finding a large number of cross-reference links is important.

The algorithm we present in section 3 uses a depth-first search because of the above conclusion. A breadth-first search tries to span the whole hierarchy, thus spending as much time scanning whole parts of the hypertext that contain no relevant information. A depth-first search finds a "sparse" subset of the hypertext. Assuming that relevant nodes occur in clusters, one is more likely to encounter nodes from relevant clusters by means of the sparse search than by an exhaustive search that never gets far away from the starting point in a reasonable amount of time.

3 A Navigational Algorithm : the "Fish-Search"

The main problems in performing information retrieval on a distributed hypertext are :

- finding a good starting point for the search;
- finding the "optimal" order in which to retrieve nodes.

Our algorithm only deals with the second point. Databases such as "The JumpStation"[1] or "The World-Wide Web Worm"[2] can be used to find documents that seem to be relevant based on a header, title or citations. But even more important than a starting point which is relevant to the search one wants to perform is a starting point that is "well connected", meaning that from the starting point many sites that participate in the distributed hypertext are easily reachable. Existing tools do not support finding such well connected starting points.

Our search algorithm is based on the **schools of fish** metaphor : A school of fish moves in the direction of food. While swimming, the fish also breed. The number of children and their strength depends largely on the amount of food that is found. The school regularly splits into parts, when food is found in different directions. Individual fish or parts of the school that go into a direction in which no food is to be found will die of starvation. Fish or parts of the school that enter polluted waters also die.

When explaining the algorithm we will frequently refer to the metaphor.

Our Search Algorithm makes the following assumptions :

1. It is impossible to perform a complete search in a reasonable amount of time.
2. The relative cost (time/size) for retrieving a document is more or less constant for each site, but may differ significantly from site to site; the retrieval time is always an order of magnitude longer than the time to actually scan the document for information, regardless whether it is a keyword (or full-text) search, (approximate) regular expression search or a search using any kind of information filter.
3. Nodes that are relevant are usually clustered (meaning there is food for a number of fish, not just a single one).

[1] The JumpStation can be found at `http://www.stir.ac.uk/jsbin/js`
[2] The Worm can be found at `http://www.cs.colorado.edu/home/mcbryan/WWWW.html`

4. Depth-first search is generally better than breadth-first, except when there is very little time [3]. (This means the school moves on, rather than staying in the same neighborhood for too long and risking to die of starvation.)

5. All nodes have a unique identification, called *Universal Resource Locator* (URL), which can be used to tell the hypertext system to retrieve the node with a given URL.

6. Repetitive access to the same remote site may overload or break the connection, so access has to be spread among sites (this is not really an assumption but an observation).

7. Accessing sites in parallel does not increase overall throughput, (or if it does it places an unacceptable load on the network).

8. There is always a "current" node from where the search starts.

Note that the assumptions 3, 4, 5 and 8 relate to the hypertext structure; assumptions 2, 6 and 7 relate to the distributed system and network; assumption 1 is simply a matter of size vs. network and computation speed.

A particularly important aspect is the choice of the navigation strategy. One could imagine that because of assumption 5 the navigation strategy would not be important in the search-algorithm because we assume that there is no overhead in navigating from one node to another node to which there is no direct link. However, since we cannot retrieve all nodes in the distributed hypertext, we need a strategy that selects an order of retrieval which provides a reasonable distribution of the nodes within the hypertext. In [3] the distribution of nodes is not measured directly but the number of cross-reference links that are discovered is measured instead. Each time a cross-reference link is found a "gateway" is discovered that leads to another part of the hypertext. Cross-reference links can exist between nodes on the same site and between nodes on different sites. Our algorithm favors links to different sites because of the penetration they provide into different parts of the hypertext, and also because successive network accesses should preferably not be concentrated on the same part of the network. The experiments in [3] show that depth-first navigation results in the discovery of the largest number of cross-reference links, regardless the structure of the hypertext or the navigation aids offered by the system.

The algorithm keeps a sorted list (of URL's of nodes to retrieve). Sorting is based on the relevance of nodes and on the last-in first-out principle that generates depth-first navigation behavior. (This *list* represents the school of fish.) Each time a node is retrieved, the URL's of the nodes to which the outgoing links point are added to this list, in a way we shall explain later. (This is the way the fish produce offspring.) When we say a "link" is added to the list we actually mean the URL of the node the link points to.

The algorithm can be influenced by the following parameters:

width: Some nodes have a large number of outgoing links. Checking all these links would hold up the search in the immediate neighborhood of that node. Therefore the number of links that will

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.