Journal of Information Science
http://jis.sagepub.com/

A web crawler design for data mining

Mike Thelwall

Journal of Information Science 2001 27: 319
DOI: 10.1177/016555150102700503

A web crawler design for data mining

Mike Thelwall

University of Wolverhampton, Wolverhampton, UK

Received 7 February 2001
Revised 5 June 2001

Abstract.

The content of the web has increasingly become a focus for academic research. Computer programs are needed in order to conduct any large-scale processing of web pages, requiring the use of a web crawler at some stage in order to fetch the pages to be analysed. The processing of the text of web pages in order to extract information can be expensive in terms of processor time. Consequently a distributed design is proposed in order to effectively use idle computing resources and to help information scientists avoid the need to employ dedicated equipment. A system developed using the model is examined and the advantages and limitations of the approach are discussed.

1. Introduction

The economic and cultural importance of the web has guaranteed considerable academic interest in it, not only for affiliated technologies, but also for its content. Research into web pages themselves has been motivated by attempts to provide improved information retrieval tools such as search engines [1–3], but also by the desire to know what is available, how it is structured and to determine its relationship with other meaningful human activities [4–14]. One strand has been the counting of links between academic web sites in order to construct an analogy to the journal Impact Factor [15–19]. The figures required for this calculation can be obtained using the advanced facilities available in search engines such as AltaVista and Infoseek, but their use has raised questions of reliability that have led to the creation of a specialist web spider/analyser to produce the raw data by direct crawling and analysis of the sites concerned. Information scientists and others wishing to perform data mining on large numbers of web pages will require the services of a web crawler or web-crawler-based tool, either individually or collaboratively [20]. The potential power of web mining is illustrated by one study that used a computationally expensive technique in order to extract patterns from the web and was powerful enough to find information in individual web pages that the authors would not have been aware of [21]. A second illustration is given by the search engine Google, which uses mathematical calculations on a huge matrix in order to extract meaning from the link structure of the web [1]. The development of an effective paradigm for a web mining crawler is, therefore, a task of some importance.

A web crawler, robot or spider is a program or suite of programs that is capable of iteratively and automatically downloading web pages, extracting URLs from their HTML and fetching them [22]. A web crawler, for example, could be fed with the home page of a site and left to download the rest of it. A sophisticated web crawler may also perform additional calculations during the crawl in order to identify pages judged relevant to the crawl or to reject pages as duplicates of ones previously visited. One important role for crawlers is to support the action of search engines, normally assisted by other programs that construct the searchable index after the crawler has obtained the pages. If data mining is required, then it is possible to either build this into the crawler or into a separate program. In the normal course of operation, a simple crawler will spend most of its time awaiting data from remote computers as part of the process of requesting and receiving a web page. For this reason, crawlers are normally multi-threaded, so that hundreds of web pages may be requested simultaneously by one process [23, 24]. If, however, the crawling task requires more complex processing, then

Correspondence to: M. Thelwall, School of Computing and Information Technology, University of Wolverhampton, Wulfruna Street, Wolverhampton WV1 1SB, UK. E-mail: m.thelwall@wlv.ac.uk

clearly the maximum number of threads that can be efficiently handled will be reduced, and consequently the speed of the crawler too. This paper presents a distributed approach for crawlers including computationally expensive additional functionality, such as data mining. It works by operating as a distributed system, with a central control unit allocating tasks to idle computers connected to a network.

Distributed systems are not new. The idea of using idle computers connected to the internet or other network as a means of gaining extra processing power has been used many times before [25–27]. A large-scale distributed system for web crawling has also been proposed, but one aimed at general web searching [28] rather than for specific tasks. Large search engines can also widely distribute processing power, including Google, which uses 'thousands of PCs linked together' [29]. Much smaller-scale crawlers have also been in use for some time, for example personal site-specific spiders [30]. These run on single personal computers and their performance, in terms of the number of pages that can be processed in any given time period, is limited by that fact. For applications requiring extensive processing by the crawler, a single personal computer solution may be fast enough. An alternative is a distributed model in which a central control unit allocates tasks to many crawlers operating on individual personal computers. This is a practical solution in any organization in which many computers are left unused for periods of time, for example weekends. Many universities would also make ideal candidates, with whole laboratories of computers left idle during holiday periods. An architecture for such a system will be discussed, one that is easy to create, install and run, even through firewalls.

2. Architecture

The proposed design must be capable of operating in a networked environment with minimal disruption to its normal functioning. To this end, four constraints were placed upon the implementation of the general concept.

1. Almost all processing should be conducted on idle computers.
2. The distributed architecture should not significantly increase network traffic.
3. The system must be able to operate through a firewall.
4. The components must be easy to install and remove.

2.1. The crawler/analyser units

The crawler is the program that will crawl a site or set of sites, analyse the pages fetched and then report its results. It will need to be written so that it can execute on the type of computers on which there will be spare time, normally personal computers running a version of Windows, and so could be written in C, Java, Visual Basic or Delphi, for example. The crawler will need to access permanent storage space to save the web pages downloaded before analysing them. This can be achieved by linking to a database, if the target computers are known to all have a common one capable of handling enough data. In other situations a simple solution is to use the normal file storage system, saving the pages as individual files. In either case, in order to make the program easily removable, it must provide a facility for the user to delete all saved data. Pages must be saved on each host computer, rather than transmitted to the control unit, in order to minimize network traffic. This precludes the use of a large-scale server-based database to store web pages. The interface must include an immediate stop button visible when it is running, and a button to clear all data from the computer. It is also recommended to include a button to stop the crawler once it has completed the current task. This is useful for cases when it is known in advance that a computer, or set of computers, will be needed for other purposes at a given time in the future.

2.2. The control unit

The control unit will live on a web server and will be triggered by it when a crawler unit requests a job or sends some data. The ability to start a program in response to a particular web page request is standard to the common web servers such as Apache and Internet Information Server as part of their implementation of the Common Gateway Interface. The program can be written in any language capable of running on the server computer, with common choices for CGI applications being Perl, ASP, C and Python. It will need to store the commands that the owner wishes to be executed, together with their status, whether they have been completed, have been sent to a crawler, or are as yet unallocated. A simple way of implementing this is as a text file saved on the server. In order to make the system robust in an environment where individual components can be stopped without warning, a feature must be included to automatically identify incomplete jobs, and then to reallocate them either with or without human intervention. This allows, for example, the owner of the computer on which any crawler unit is operating to stop it without letting it complete its current job.

2.3. The messaging system

In order to satisfy the first and second requirement above, the crawler units must be almost completely self-contained, and able to download and process entire websites with little or no communication with the control unit. This can be achieved by reducing the communication between the two to that concerning the nature of the task to be executed, results reporting and status checks. Interaction is therefore limited to the following messages.

- the crawler unit, upon starting or completing a crawl, sending a message to the control unit indicating that it is ready to execute a new request;
- the control unit sending an instruction to the crawler unit indicating a site to crawl, and the type of processing to be performed on the downloaded site;
- the crawler unit reporting its results once the crawl has been completed.

This will only work, however, if the crawl results to be reported are significantly smaller in size than the web site. If, for example, the entire uncompressed web site were to be ‘reported’, then this would represent an unacceptable doubling of network use.

Firewalls can cause problems to distributed systems because they are designed to stop access to specified information and resources to all those with computers running outside the protected network. A firewall, for example, may protect a web server by only allowing computers outside of a given network to send information into it that can be identified as part of the process of requesting and downloading a web page. The distributed crawler design described here can operate through this kind of firewall by hosting the control unit on a web server and coding all communications between the crawler and control units as web page requests to the control unit and web pages returned. A firewall can also be configured to stop all computers outside a network from requesting information from the computers inside it, whilst allowing those inside to request any external information. As an example of this, networks of open access student computers may be configured so that the computers can use all the resources of the internet, but outside users would be prohibited from sending in any unrequested information. The distributed design circumvents this problem by coding information sent to the crawler units from the control unit as web pages. The particular problem that cannot be avoided is that a control unit outside the network could not initiate communication with computers inside, but only send information in response to a request for it.

In summary, the architecture will work for any system where the control unit is on a public access web server and the client units are on computers with permission to access the web. This is possible because all the information communicated is encoded in web page requests and responses. It does not allow individual crawler units to communicate with each other, nor the control unit to initiate communication with any crawler, but this, in fact, is not a problem. The overall design is shown in Fig. 1.

3. A distributed web analyser

The architecture described above was employed to create a system for the task of analysing the link structure of university web sites. The need for this was rooted in previous experience in running a single crawler/analyser program. This program had not run quickly enough to cover the necessary number of web sites, and so it had been individually set up and run on a number of computers simultaneously. The inefficiency of this approach, in terms of both human time and processor use, led to the specification of the distributed system as a logical progression from the existing technology. In this new system, the existing standalone crawler was used as the basis for the new crawler

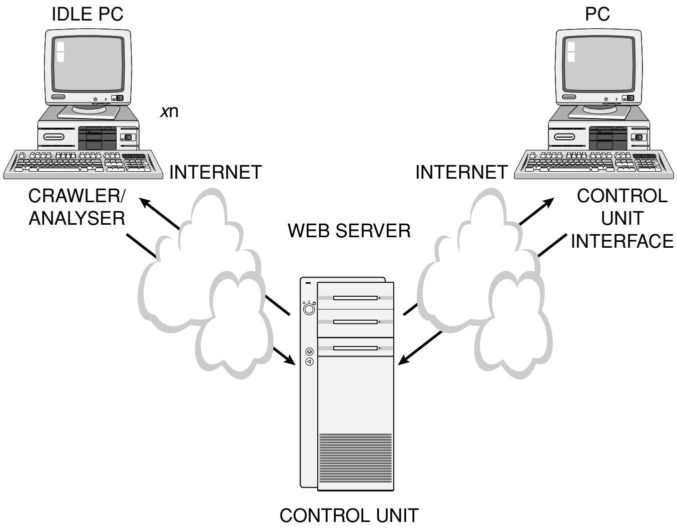

Fig. 1. The distributed data mining crawler design.

unit component, greatly saving development time. The main modifications made were in adding the communication mechanisms and features, allowing it to be easily installed and removed. The latter changes resulted in the new program being available as a single file small enough to fit on a floppy disk and having buttons to instantly close the program and remove any saved data. The program was written in a rapid application development language in order to be able to quickly design and modify components of the analyser. It was then processed by an executable compressor in order to get it and all supporting library files to fit on one disk as a single file.

One feature that was built into the crawler component was a choice of the types of checking for duplicate pages to be used in a web site crawl. There were three options:

- use no page checking – assume that two pages with different URLs are different pages;
- use HTML page checking – reject any new page (except a frameset page) which has identical HTML to a previously retrieved page;
- use weak HTML page checking – reject any new page (except a frameset page) which has at least (say) 95% identical HTML to a previously retrieved page. This option was designed to pick up pages which contained a text counter, or similar incidental changeable page component, so that if the same page is downloaded from two different URLs, the HTML would be slightly different the second time.

The inclusion of three different crawling methods was necessary, because one of the tasks to be given to the system was comparing the output produced with the different versions, and other tasks to be processed later would require very precise results. This activity did, in fact, add considerably to the workload of the program for the larger sites. In order to minimize the work, instead of comparing each page against all of the others, various numbers were calculated from the text of each page, and only if enough of the numbers matched between two pages were they fully compared to see if the new one was to be rejected as the same, or essentially the same, as the other one. A version of this technique has been used in another search engine, Mercator [23], and may well be used by others. The simplest of the numbers was the length of the page, with the others calculated from the page content. This technique speeded up the process, essentially because the derived numbers were kept in main memory, and so could be checked very quickly, whereas comparing the actual pages would require them to be fetched from the hard disk first, slowing down the process considerably.

The analyser component inherited most of the analytical capabilities of its predecessor, in addition to some new ones. The three main types of tasks that it could be commanded to do were as follows.

- summarise the page and link structure of a given web site;
- process individual pages or framesets to extract information such as meta tag, Java and JavaScript use;
- use page-scraping techniques in order to report on the presence or absence of web sites in a set of search engines.

In addition to the main analyses, there was a default 'idle' task which was sent when all control unit commands had been allocated. This instigated a half-hour wait before repeating the request for a new instruction. This was essential to the smooth running of the system.

The control unit was entirely new and consisted of a simple Perl program. In addition to the functionality described in the previous section, it was given a reporting facility so that it could deliver to the owner a summary of which computers had crawler units allocated to tasks and which tasks were completed, allocated and unallocated. This provided a useful mechanism for checking that all was proceeding as expected, and allowed manual identification and checking for computers that had been switched off or which were malfunctioning.

Table 1
A sample of the controller output

Computer ID	Code of task	Site/ page	Job start time	Total jobs finished by this computer
bn-45.wlv.ac.uk	01n	www.uts.edu.au	16-00 on 30/7/00	172
bo-45.wlv.ac.uk	03s	www.gu.edu.au	9-21 on 28/7/00	134
bq-45.wlv.ac.uk	03n	www.uwa.edu.au	17-12 on 31/7/00	156

4. System testing

The system was given a series of tasks to perform in June and July of 2000. Each consisted of a set of sites or web pages to download, plus an analysis to perform on the downloaded sites. The total number of crawler/analyser units in use at one time varied during the time period as additional rooms of computers became available and had to be phased out. The number peaked at just over 100 with three rooms of computers operating as part of the system. This is still a small number compared to the number of threads that a commercial search engine crawler can run simultaneously on one machine, but it represents a significant use of idle resources, which would otherwise have been switched off and left in locked rooms. The total number of tasks completed by the system was 9112, ranging from a request to download and process a single web page to web sites of over 100,000 pages. An additional benefit from the crawler was in the availability of temporary storage space for the downloaded sites. Each computer used had approximately 1 Gb of hard disk space free, and so at its peak the system had become a virtual computer with over 100 Gb of disk space and over 100 processors.

The system was not, however, able to run fully automatically on all of the sites because the nature of one type of crawl meant that there could be no automatic checks for looping. The first task set, a crawl of the UK academic web, will be discussed here, including a more detailed coverage of this problem and the strategy employed to solve it. The system was given the task of analysing the link structure of the UK academic web in June 2000. First, a list of home page URLs was obtained, converted into instructions and uploaded to the control unit. Then the crawler unit was installed onto each of 40 machines in an idle student computer laboratory. The task required that the whole of each web site, as far as was possible by following links, should be downloaded and analysed. On some websites this was not a problem, but on others the control unit reporting mechanism showed jobs taking much longer than expected. The problem was that these sites contained large numbers of unnecessary files, or a virtually infinite number. The unnecessary files included, for example, a huge set of web pages containing usage statistics for electronic equipment with one page per device per day. The virtually infinite collections included one set of apparently randomly generated web pages, each containing a list of words and links to other similar pages. The solution to this problem was to manually check which files were the root cause of the problem and then add their URLs to a banned list operated by the control unit. This was a labour-intensive task but necessary since the alternative of designing a heuristic to avoid such problems, for example a maximum crawl depth, had been rejected as too arbitrary for the purposes of the analysis being conducted. The manual modification procedure that was adopted to deal with this issue was as follows:

1. Identify unexpectedly long jobs through the control interface.
2. Go to the computer executing the job and verify that there is a genuine looping problem.
3. Stop the individual crawler unit.
4. Identify the cause of the looping by examining the crawler log and inspecting the individual pages.
5. Add the page or pages to a ‘banned list’ kept on the server and automatically checked by crawler units when starting new jobs.
6. Restart the crawler unit and manually configure it to resume the previous task.

It was possible to operate the crawler manually because it retained a full interface from its previous incarnation as a stand-alone crawler. The reason it was restarted on the same machine, rather than allowing the control unit to reallocate the job to another computer at random, was to allow the program to restart its crawl with local copies of web pages previously fetched, rather than having to download them all again.

As part of the testing, separate tasks were run with different duplicate checking procedures. The additional work did slow down the crawling process considerably on the larger sites when tens of thousands of pages had been downloaded and needed to be checked for each new page retrieved. The total number of pages downloaded using each method for most sites varied by only a few percent and so the net effect was an increase in the total time taken to complete the crawl. It is difficult to quantify exactly the crawl length difference because of the natural periodic and random variability of internet download times, but, in very general terms, the difference was less than 1% for sites with under 1000 pages, but was approximately double for sites of about 100,000 pages. It is likely that more sophisticated checking algorithms could considerably reduce this additional time, even for larger sites. Most duplicate pages were identified by both checking methods described above, with very few ‘duplicates’ having non-identical HTML. On a few of the sites, however, checking did prevent infinite looping and so did lessen the need for human intervention. The end result of using duplicate checking, then, was to slow down the crawl without producing significantly different results on most sites and reducing the need for human intervention only slightly. Despite these findings, checking is still essential if very precise results are required.

The system proved itself capable of executing the task, albeit with manual intervention for the problem sites. The big advantage over the previous approach of loading and operating copies of an autonomous crawler/analyser was a reduction in the human effort required to operate it. A second advantage was much more efficient use of resources as, once a crawler had completed one job, a new one would be automatically allocated. One feature that was not available, but which is planned for the next version, is the ability for the crawler units to automatically update themselves. This facility would have been useful because at two stages the whole system was shut down and an updated crawler was installed on all workstations. This process was not too time-consuming since the installation process was simply copying the new program file from a floppy disk to each computer, but it could be avoided by including the ability to download upgrades from the internet into the crawler units.

5. Conclusion

The distributed architecture has shown itself capable of crawling a large collection of web sites by using idle processing power and disk space. The testing of the system has shown that it cannot operate fully automatically for tasks that involve crawling entire web sites without an effective heuristic for identifying duplicate pages. This is a problem that would also be met by any crawler executing the same task, but is possibly more of a problem in a distributed system where the tasks can be spread around computers in different geographical locations. When the computers have to be individually visited to get details of the crawl, this adds to the human time element, even when computers with potential problems are identified through the central control unit. The system's ability to run fully automatically was demonstrated following the main test period when it executed thousands of requests for sets of pages to be fetched and processed without requiring any human intervention. It is considered, then, that the extra time spent on the test crawl was a relatively minor problem that does not undermine the validity of the concept.

The architecture is particularly suited to situations where a task can be decomposed into a collection of disconnected crawling based tasks so that their execution on different machines would not create a problem. It would be unsuitable if the crawls had to cross-reference each other in any way, for example to check whether a page from one crawl had already been already found in another. It would also be unsuitable if the data mining had to be performed upon the whole data set in an integrated way. It has been shown to be effective here, however, when the task was to crawl and summarise data from a collection of different web sites for comparison and subsequent further analysis.

The tests took place in a university environment, which simplified the task of finding available computers because they were all co-located in three workshop rooms. In a different workplace, the idle computers may well be spread over individual offices in one or more buildings, creating problems of identifying when they were available. Two logical solutions would be to make the software activate itself when the computers were idle or to request the owners to run the program instead of turning their PCs off when they would be unused for a useful period of time. The practicality of such an approach remains untested.

Experience with the system tested pointed to the need for a facility to automatically upgrade the crawler/analyser units as new features were added, but, apart from this, the overall design did not seem to need major modification. It would have been desirable, however, to be able to request additional information from the crawler/analyser units via the control unit without actually visiting them, and also to send a command to terminate a particular job that needed restarting. The impossibility of this is a by-product of the need for the system to operate through firewalls. A major task of a firewall is often to stop computers on the outside from being able to request information from those inside. The system bypasses this by disguising the data as a request for information from the external control unit, something that is normally allowed. With this technique, however, the control unit sitting on the web server, which must be outside the firewall in order to be allowed to respond to requests for information, cannot initiate the request. A possible workaround for this would be to code the crawler/ analyser units to periodically poll the control unit, perhaps hourly, in order to see if there were any additional requests for information about the crawl, or commands to be processed immediately. This would provide a solution, albeit with a time delay, to the central control problem.

In summary, it is contended that the architecture presented is an effective way to use idle computing resources within an organization in order to create the capacity to execute certain types of large-scale web data mining tasks. The importance to information scientists of having access to this kind of functionality lies in the value of the web as an information resource [20] and increasing evidence that meaningful patterns can be extracted from web data [31].

References

[1] S. Brin and L. Page, The anatomy of a large scale hypertextual web search engine, *Computer Networks and ISDN Systems* 30(1–7) (1998) 107–117.

[2] J. Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM* 46(5) (1999) 604–632.

[3] J. Choo, and H. Garcia-Molina, The evolution of the web and implications for an incremental crawler. In: *Proceedings of the 26th VLDB Conference*, Cairo, Egypt (2000) 200–209.

[4] C. Chen, Structuring and visualising the world-wide web with generalised similarity analysis. In: *Proceedings of the 8th ACM Conference on Hypertext (Hypertext '97)*, Southampton, UK (April 1997). Available from: www.brunel.ac.uk/~cssrccc2/papers/ht97.pdf

[5] C. Chen, J. Newman, R. Newnam and R. Rada, How did university departments interweave the web: a study of connectivity and underlying factors, *Interacting with Computers* 10 (1998) 353–373.

[6] D. Gibson, J. Kleinberg, and P. Raghavan, Inferring web communities from link topology. In: *Hypertext 98: Ninth ACM Conference on Hypertext and Hypermedia* (ACM, New York, 1998).

[7] A. Lubansky, Socio-economic impact of the internet in the academic research environment. In: *IRISS 98 International Conference*, Bristol (25–27 March 1998). Available from: www.sosig.ac.uk/iriss/papers/paper18.htm

[8] H. Miller and R. Mather, The presentation of self in WWW home pages. In: *IRISS 98 International Conference*, Bristol (25–27 March 1998). Available from: http://www.sosig.ac.uk/iriss/papers/paper21.htm

[9] I. Middleton, M. McConnell and G. Davidson, Presenting a model for the structure and content of a university world wide web site, *Journal of Information Science* 25(3) (1999) 219–227.

[10] J. Bar-Ilan, The web as an information source on informetrics? A content analysis, *Journal of the American Society for Information Science* 51(5) (2000) 432–443.

[11] A. Broder, R. Kumar, F. Maghoull, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins and J. Wiener, Graph structure in the web, *Computer Networks* 33(1–6) (2000) 309–320.

[12] S.W. Haas and E. S. Grams, Readers, authors and page structure: a discussion of four questions arising from a content analysis of web pages, *Journal of the American Society for Information Science* 51(2) (2000) 181–192.

[13] B. Kelly, WebWatch: a aurvey of links to UK university web sites. *Ariadne* 23 (2000). Available from: http://www.ariadne.ac.uk/issue23/web-watch/

[14] M. Thelwall, Commercial Web Site Links, *Internet Research* 11(2) (2001) 114–124.

[15] P. Ingwersen, Web impact factors, *Journal of Documentation* 54(2) (1998) 236–243.

[16] H. Snyder and H. Rosenbaum, Can search engines be used for web-link analysis? A critical review, *Journal of Documentation* 55(4) (1999) 375–384.

[17] A.G. Smith, A tale of two web spaces: comparing sites using web impact factors, *Journal of Documentation* 55(5) (1999) 577–592.

[18] M. Thelwall, Web impact factors and search engine coverage, *Journal of Documentation* 56(2) (2000) 185–189.

[19] M. Thelwall, Results from a web impact factor crawler, *Journal of Documentation* 57(2) (2001) 177–191.

[20] J. Bar-Ilan, Data collection on the web for informetric purposes – a review and analysis, *Scientometrics* 50(1) (2001) 7–32.

[21] N. Sundaresan and J. Yi, Mining the Web for relations, *Computer Networks* 33 (2000) 699–711.

[22] T.Y. Chun, World wide web robots: an overview, *Online & CD-ROM Review* 23 (3) (1999) 135–142.

[23] A. Heydon, and M. Najork, Mercator: a scalable, extensible web crawler, *World Wide Web* 2 (1999) 219–229.

[24] M.S. Jackson and J.P.H. Burden, WWLib-TNG – new directions in search engine technology. In: *IEE Informatics Colloquium: Lost in the Web – Navigation on the Internet* (1999) 10/1–10/8.

[25] M.M. Theimer and K.A. Lantz, Finding idle machines in a workstation-based distributed system, *IEEE Transactions on Software Engineering* 15(11) (1989) 1444–1458.

[26] F. Monrose, P. Wyckoff, and A.D. Rubin, Distributed execution with remote audit. In: *Proceedings 1999 Network and Distributed System Security Symposium*, Reston, VA, USA (1999) 103–116.

[27] M.C. Ferris, M.P. Mesnier and J.J. More, NEOS and Condor: solving optimization problems over the internet, *ACM Transactions on Mathematical Software* 26(1) (2000) 1–18.

[28] M.A.C.J. Overmeer, My personal search engine, *Computer Networks* 31 (1999) 2271–2279.

[29] Google. http://www.google.com/corporate/facts.html. Accessed 21 January 2001.

[30] R.C. Miller and K. Bharat, SPHINX: a framework for creating personal, site-specific web crawlers, *Computer Networks and ISDN Systems* 30(1–7) (1998) 119–130.

[31] M. Thelwall, Extracting macroscopic information from web links, *Journal of the American Society for Information Science and Technology* (2002) (in press).