Panorama: Extending Digital Libraries with Topical Crawlers*

Gautam Pant
Department of Management Sciences
The University of Iowa
Iowa City, IA 52242
gautam-pant@uiowa.edu

Kostas Tsioutsiouliklis
Judy Johnson
NEC Laboratories America, Inc.
4 Independence Way
Princeton, NJ 08540
{kt,jaj}@nec-labs.com

C. Lee Giles
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802
giles@ist.psu.edu

Abstract

A large amount of research, technical and professional documents are available today in digital formats. Digital libraries are created to facilitate search and retrieval of information supplied by the documents. These libraries may span an entire area of interest (e.g., computer science) or be limited to documents within a small organization. While tools that index, classify, rank and retrieve documents from such libraries are important, it would be worthwhile to complement these tools with information available on the Web. We propose one such technique that uses a topical crawler driven by the information extracted from a research document. The goal of the crawler is to harvest a collection of Web pages that are focused on the Web communities associated with the given document. The collection created through Web crawling is further processed through lexical and linkage analysis. The entire process is automated and uses machine learning techniques to both guide the crawler as well as analyze the collection it fetches. A report is generated at the end that provides visual cues and information to the researcher.

Keywords: digital libraries, topical crawling, Web mining

1 Introduction

The World Wide Web is a very large and mostly unedited medium of publication. In contrast, a number of digital libraries index a more controlled set of quality documents that are of interest to an organization or a community of researchers. However, the Web can be expected to have a lot of information related to the documents in a digital library that would be of interest to the users of the library. Bridging the gap between the high "quality" information available within the digital libraries and the high "quantity" of information on the Web is an important research problem. The high quality information can be used to bootstrap a process for deriving a larger pool of related information from the Web.

A simple idea along these lines would be to query a Web search engine with the keywords (or query) supplied by the user in addition to retrieving relevant documents from the digital library. The results from the search engine can then be displayed along with the documents from the library. Our approach is somewhat different. We would like to provide a more panoramic view about the Web communities represented by a document or a set of documents. The focus of the current work is on providing a researcher with a tool to advance her knowledge and understanding associated with a research document. The tool must make it easier to identify key Web communities of interest, their associations, properties and examples. Our long term goal is to be able to provide similar information for any kind of well formatted document.

In the next section we discuss our general approach. This is followed by a description of our current imple-

*Technical Report 2003-L087, NEC Labs, Princeton, NJ.

mentation of the approach (Section 3). We then detail the creation of the test bed (Section 4). In Section 5 we present the results of our study and Section 6 describes some of the related work.

2 The General Approach

Figure 1 illustrates our general approach. Given a document or a set of documents from a digital library, we first parse them and extract information. The extracted information may include the title, author names, affiliations, abstract, keywords (if provided) and references. We then use this information to characterize the document or the set of documents. In the case where we have more than one document we may correlate the information from different documents to find common or overlapping characteristics. For example, we can identify words or phrases that are commonly used, or the references that are popularly cited in the given document set. In the current implementation we restrict ourselves to information derived from single documents.

The information extracted through parsing is used to query a search engine. For example, we can use the title and the author names as individual queries to a search engine. The pages corresponding to the top results of the search engine are then treated as positive examples of the desired information (there may be a filtering step to avoid bad examples). The positive examples are used to train a classifier. The negative examples for the classifier may be picked at random from a large database of Web pages. Once the classifier is trained, it is used to guide a *topical* crawler. A topical or *focused* crawler is a program that follows hyperlinks to automatically retrieve pages from the Web while biasing its search towards topically relevant portions of the Web. The trained classifier will provide the crawler with the needed bias.

Once a collection of Web pages has been downloaded by the crawler, we analyze them to find more structured information such as potential Web communities and their descriptions. The analysis process includes both lexical as well as link (graph) based analysis. The final result of the analysis is then shown as an interactive graphical report that describes various clusters (potential communities) found through the crawl, their examples, as well as authorities and hubs within each cluster.

3 Implementation

The general approach leaves room for a number of different implementations. For example, we may use various components of the information extracted through parsing to query a search engine. One of the many classifier algorithms, such as the Naive Bayes, the Support Vector Machines or the Neural Networks, may be used to guide the topical crawler. Similarly, we have a choice of clustering algorithms (lexical and link based) to use for identifying key Web communities. Finally, what properties we identify for each of the clusters is an open area for experimentation as well. We call our current implementation *Panorama* because it uses the Web to provide a wider, more complete view of information surrounding a research paper.

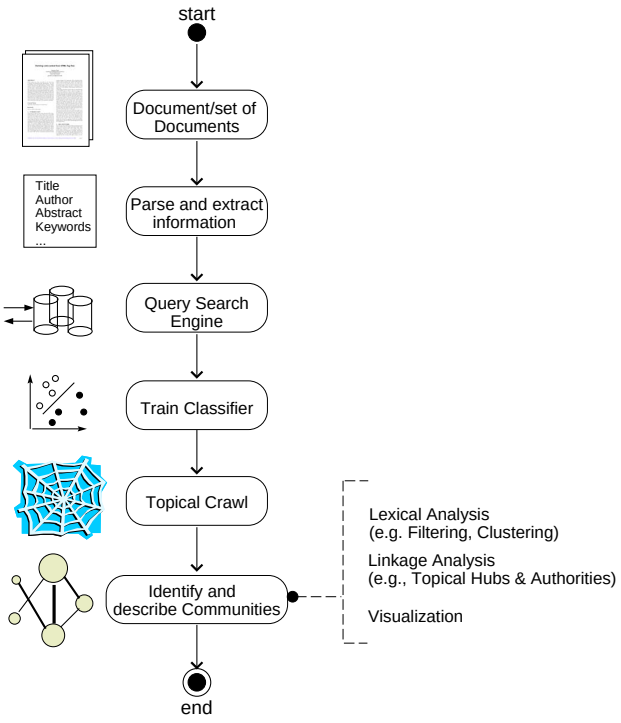

Figure 1: General Approach

3.1 Parsing The parsing code is an adaptation of that used by CiteSeer [5].[1] The adapted code takes in a research document in `pdf` or `ps` format and converts it into an XML file that contains the extracted information within appropriate tags. Even though we parse and extract a number of attributes of a research paper, currently we use just the *main* title of the paper and the titles of references (*reference* titles) within the paper. We assume that through the main title and a large enough sample of the reference titles, we capture various aspects of the topic or topics represented by the given paper.

[1] `http://citeseer.nj.nec.com/`

3.2 Examples from Search Engine Once we have extracted the main title and the reference titles we use them as exact phrase queries to Google using the Google Web APIs.[2] The search engine results returned for each of the titles are URLs of pages that contain the corresponding title (phrase) in them. Since, the titles of papers often tend to be quite suggestive as well as unique, the results returned by the search engine tend to be very relevant. The Web API provides us with 10 or fewer results for each of the queries. We accumulate the results (URLs) for each of the titles and the corresponding pages make up a positive example (training) set. Table 1 shows an instance of titles extracted from a paper and the corresponding positive examples.

3.3 Classifier The negative examples for the classifier are a random subset of positive examples found for other unrelated papers. The other papers may be a set of papers that are used just to provide negative examples. We keep the number of negative examples to twice the number of positive examples. All the pages in the positive as well as the negative set are parsed and tokenized to identify the words within them. The stop-words are removed and the remaining words are stemmed using the Porter stemming algorithm [16]. These stemmed words or *terms* from all the negative and positive examples form our vocabulary $\mathcal{V}$. Next, we represent the positive and the negative examples as feature vectors where each element in the vectors corresponds to a term in $\mathcal{V}$. Thus, we represent a Web page p as a vector $\vec{v_p} = (w_{1p}, w_{2p}, ..., w_{np})$, where w_{sp} is the weight of the term s in page p and n is the size of $\mathcal{V}$. The weights are TF-IDF (Term Frequency-Inverse Document Frequency) weights [19] that are computed as:

$$w_{sp} = \underbrace{\left(0.5 + \frac{0.5 \cdot tf_{sp}}{\max_{s' \in T_p} tf_{s'p}}\right)}_{TF} \cdot \underbrace{\ln\left(\frac{|C|}{df_s}\right)}_{IDF} \quad (3.1)$$

where tf_{sp} is the frequency of the term s in page p, T_p is the set of all terms in page p, C is the collection containing the positive and the negative examples, and df_s (document frequency or DF) is the number of pages in C that contain the term s. Equation 3.1 shows the standard *at* components of *atc* weighting in the SMART system [18].

Both the positive and the negative example pages are represented as TF-IDF based feature vectors as described above and used for training a Naive Bayes classifier. The implementation of the Naive Bayes classifier that we use is the one available as part of the Weka machine learning software.[3] Once the classifier has been trained it can be used to estimate the $\Pr(c|q)$, where c is the class (positive or negative) and q is an arbitrary page. The space in which the page q is represented remains confined to the vocabulary $\mathcal{V}$. Also, the page q is represented in the same way as the positive and the negative examples with term weights being computed as given in Equation 3.1. Note that we will continue to use the df_k computed based on the collection C. We call the trained classifier as the *crawling classifier* since we will use it to guide the topical crawler.

3.4 Topical Crawler A crawler is a program that automatically fetches pages from the Web while following the hyperlinks. Crawling can be viewed as a graph search problem. The Web is seen as a large graph with pages as its nodes and hyperlinks as its edges. A crawler starts at a few of the nodes (seeds) and then follows the edges to reach other nodes. The process of fetching a page and extracting the links within it is analogous to expanding a node in graph search. A topical crawler tries to follow edges that are expected to lead to portions of the graph that are relevant to a topic. In other words, it biases the node expansion order (crawl path) to harvest topically relevant pages.

Current work uses a crawler that is implemented as multi-threaded objects in Java. The crawler can have many (possibly hundreds) threads of execution sharing a single synchronized *frontier* that lists the unvisited URLs. Each thread of the crawler follows a *crawling loop* that involves picking the next best URL to crawl from the frontier, fetching the page corresponding to the URL through HTTP, parsing the retrieved page, and finally adding the unvisited URLs to the frontier. Before the URLs are added to the frontier they may be assigned a score that represents the estimated benefit of visiting the page corresponding to the URL. Using the previously trained crawling classifier (Section 3.3), we assign each unvisited URL a score that is equal to $\Pr(c^+|p)$ where p is the parent page from where the URL was extracted and c^+ is the positive class. The starting pages or seeds for the crawler are the positive examples for the given paper and some of its in-links. We get the in-links again by using the Google Web API.

The crawler implementation uses 75 threads of execution and limits the maximum size of the frontier to 50,000. Only the first 10KB of a Web page are downloaded. We typically crawl 5,000 pages for a given research paper. In a crawl of 5,000 pages a crawler may encounter more than 50,000 unvisited URLs. However,

[2] http://www.google.com/apis/

[3] http://www.cs.waikato.ac.nz/ml/weka/

Table 1: A sample paper - main title, reference titles and positive examples

main title	*reference titles*	*positive examples*
Combinatorial algorithms for DNA sequence assembly	1. A greedy approximation algorithm for constructing shortest common superstrings 2. Identification of common molecular subsequences 3. An upper bound technique for lengths of common subsequences 4. Minimal mutation trees of sequences 5. An algorithm for reconstructing protein and RNA sequences 6. Linear approximation of shortest superstrings ...	http://www.cs.helsinki.fi/u/ukkonen/ http://www.cs.brown.edu/courses/cs250/bibliography.html http://www.csd.uwo.ca/faculty/bma/teaching/approx/resources.html http://ljsavage.wharton.upenn.edu/%7Esteele/Papers/HTML/Lcsatp.html http://theory.lcs.mit.edu/%7Edmjones/STOC/stoc91.html http://www.cse.psu.edu/%7Efurer/597/bib-with-links.html http://www.cs.arizona.edu/people/kece/Research/publications.html http://www-users.cs.umn.edu/%7Echi/papers/vis95/html/node7.html http://www.psc.edu/research/biomed/homologous/bib.html ...

given that the average number of outlinks on Web pages is 7 [17], the maximum frontier size is not very restrictive for a crawl of 5,000 pages. In case the frontier size is exceeded only the best 50,000 URLs are kept. Being a shared resource for all the threads, the frontier is also responsible for enforcing certain ethics that prevent the threads from accessing the same server too frequently. In particular, the frontier tries to enforce the constraint that every batch of N URLs picked from it are from N different server host names ($N = 100$). The crawlers also respect the Robot Exclusion Protocol.[4]

3.5 Clustering Once the crawler has downloaded a collection of Web pages corresponding to a given research paper, we move to the analysis step. We first filter out the worst 25% of the pages in the collection based on the scores given to them by the crawling classifier. Then, we use the spherical k-means clustering to split the collection into meaningful sub-parts. Spherical k-means clustering is a k-means clustering algorithm where the proximity between a pair of feature vectors is measured through their cosine similarity. The cosine similarity is computed as:

$$sim(v_p, v_q) = \frac{v_p \cdot v_q}{\|v_p\| \cdot \|v_q\|} \tag{3.2}$$

where v_p and v_q are feature vectors, $v_p \cdot v_q$ is the dot product between the vectors, and $\|v\|$ is the Euclidean norm of a vector v. We set the k (number of clusters) to a value of 5. The choice of k was based on some preliminary experimentation with different papers as well as considerations relating to visualization. For the clustering we represent each Web page in the downloaded collection as TF-IDF based vectors similar to the ones used for the crawling classifier. However, the IDF part of the term weights (see Equation 3.1) is now computed based on the entire collection that is created by the crawler. The vocabulary and hence the feature space is based on all of the terms in the collection. The terms are the word stems after removing the stop-words. DF-thresholding [20] is used to remove the terms that appear in less than 1% of the collection pages. Also, the terms that appear in more than 80% of the pages are treated as stop-words for the given topic and hence removed from the vocabulary. Finally, the feature vectors representing the pages have one attribute corresponding to each term in the filtered vocabulary.

After the clustering is performed over a collection, we store for each URL the cluster (number) that it is assigned to. We also measure the cosine similarity of the pages corresponding to each URL in the collection to each of the *cluster centers*. A cluster center is the centroid of all the feature vectors (pages) that are assigned to the cluster. Hence a cluster center Q_c for a cluster c is computed as:

$$Q_c = \frac{1}{|c|} \cdot \sum_{p \in c} v_p \tag{3.3}$$

where v_p is the feature vector corresponding to page p that is in cluster c. The pages closest to their assigned cluster center are treated as good examples of the cluster.

3.6 Cluster based Hubs and Authorities We view each cluster as a potential Web community and would like to characterize it by some resourceful pages. One way to identify resourceful pages is to find the top hubs and authorities. A *hub* is a page that links to many good authorities, and an *authority* is a page that is linked to by many good hubs. This recursive definition forms the basis behind Kleinberg's algorithm [13] that gives a hub score and an authority score to each page in a collection. Initially, we applied Kleinberg's algorithm to a directed graph defined by all the pages

[4] http://www.robotstxt.org/wc/norobots.html

in a collection downloaded for a given research paper. A node in the graph is a page in the collection and an edge is a hyperlink from one page to another within the collection. We can represent the graph for a collection of m pages using an m by m adjacency matrix A such that element A_{ij} of the matrix is 1 if there is a hyperlink from page i to page j and 0 otherwise. Also, let $\vec{a}$ be a vector of authority scores where each score corresponds to a page in the collection. Similarly, $\vec{h}$ is a vector of hub scores. Then an iteration of Kleinberg's algorithm consists of the following steps:

$$\begin{aligned} \vec{a} &\leftarrow A^T \cdot \vec{h} \\ \vec{h} &\leftarrow A \cdot \vec{a} \\ \vec{a} &\leftarrow \frac{\vec{a}}{\|\vec{a}\|} \\ \vec{h} &\leftarrow \frac{\vec{h}}{\|\vec{h}\|} \end{aligned} \tag{3.4}$$

The algorithm terminates after l iterations or when some convergence criterion is met for the scores. In our implementation we terminated the algorithm after $l = 100$ iterations. We identified the top hubs and authorities that lie within each cluster of the collection. The algorithm often gave us pages that were not very typical of the cluster but were generic hubs or authorities such as `http://www.gnu.org`. We then decided to bias Kleinberg's algorithm and compute it separately for each of the clusters. We still apply the algorithm over pages in all the clusters but we apply it multiple times—each time for one cluster. During the clustering process, described in Section 3.5, we had measured the distance of each page (feature vector) from each of the k cluster centers. Based on this information we perform a modified version of Kleinberg's algorithm for cluster c where each iteration has the following steps:

$$\begin{aligned} \vec{a} &\leftarrow A^T \cdot D_c \cdot \vec{h} \\ \vec{h} &\leftarrow A \cdot D_c \cdot \vec{a} \\ \vec{a} &\leftarrow \frac{\vec{a}}{\|\vec{a}\|} \\ \vec{h} &\leftarrow \frac{\vec{h}}{\|\vec{h}\|} \end{aligned} \tag{3.5}$$

where D_c is a diagonal matrix whose ith diagonal element is the cosine similarity of page i to the cluster center of c. We again terminate the algorithm after 100 iterations. This modified algorithm is a specific case of a general form suggested by Bharat and Henzinger [4] to bias Kleinberg's algorithm. They also showed that such an algorithm converges. After running the algorithm for each cluster we identify the top hubs and authorities in the cluster. We found the results to be more cluster specific which validated the previous user studies on topically biased versions of Kleinberg's algorithms [4].

3.7 Cluster Labels It is important to label the clusters with descriptive words or phrases. The potential communities represented by the clusters can be easily characterized by such labels. In the current implementation we only look for keywords (and not phrases) to label the clusters. Again we represent all the documents in a cluster as vectors of TF-IDF term weights. However, now the terms are actual words rather than word stems. We still remove the stop-words as well as perform DF based filtering similar to that used during clustering. We compute a *profile vector* V_c [9] for a cluster c as follows:

$$V_c = \sum_{p \in c} v_p \tag{3.6}$$

where v_p is the TF-IDF based vector representing page p in cluster c. The words corresponding to the top 5 largest components of the profile vector V_c are then used as labels for the cluster c.

4 Test Bed

We used CiteSeer [5] to create a test bed in order to understand the merits of the proposed approach. We first downloaded 150 random research papers (in `pdf` or `ps` format) from CiteSeer. Next, we parsed each paper in the downloaded set to extract some pre-defined meta-information such as the main title, author names, affiliations, abstract, keywords and references. Since we depend primarily on the main title and the reference titles in the current implementation, we filtered out the papers that had less than 8 references. Finally, our test bed had 94 papers representing a wide spectrum of research papers available on CiteSeer. For each of the papers we picked half of all the references randomly and used them along with the main title to query using the Google Web API. The top results from the search engine are then used as positive examples to train a classifier that guides the crawler as explained in Section 3.3. The negative examples are a random subset of positive examples found for the other 93 papers in the test bed following the same process.

For the evaluation of the crawler performance, we use all of the reference titles (instead of a random subset), as well as the main title, as the queries. We then use the top results returned by the Web API to form a large set of positive examples. The negative examples for evaluation are a random subset of the positive examples for the other papers. Note that the positive examples for each of the papers are obtained using all of the reference titles. Hence, we can expect the classifier built using the larger and more informed pool of positive and negative examples to be a better judge of the performance of the crawler when compared to the crawling classifier. We call such a classifier the *evaluation classifier*. We build the evaluation classifier using the Naive Bayes scheme as was used for building

the crawling classifier. We use the trained classifier to then score ($\Pr(c^+|p)$) each of the pages fetched by the crawler for a given research paper. Based on these scores we compute the *harvest rate* at different points during the crawl. The harvest rate H_t after crawling t pages is computed as:

$$H_t = \frac{1}{t} \cdot \sum_{i=1}^{i=t} \Pr(c^+|p_i) \tag{4.7}$$

where p_i is the page i in the t crawled pages. This metric has been used to evaluate precision of the crawlers in the past [7, 11, 1, 6]. In most of the previous work the classifier that was used to crawl was also used for measuring the harvest rate. We deviate from that trend by using a more informed classifier for the evaluation than that used for the crawling.

5 Results

The goal of the current work is not to find the best topical crawler or for that matter the best clustering or visualization techniques. We want to simply demonstrate the feasibility of our general approach that aims to connect the Web based information with that in high quality documents of a digital library. However, we do want to make sure that our topical crawling technique gives a focused collection that is better than that produced by a simple Breadth-First crawl around the seeds. Hence, we also run a Breadth-First crawler for each of the 94 papers in our test bed. A Breadth-First crawler treats the frontier as a FIFO queue putting new unvisited URLs at the back of the queue and picking the next URL to crawl from the head.

5.1 Crawler Performance Figure 2(a) shows a graph of the average harvest rate at different points during the crawl for both the Naive Bayes crawler as well as the Breadth-First crawler. The horizontal axis approximates time by the number of pages crawled. The harvest rate is averaged over the 94 papers of the test bed and the error bars show ± 1 standard error. Hence, the figure shows average trajectories of crawler performance with time. Such graphs are often used in crawling literature to capture the temporal nature of the crawlers [7, 8, 15]. From Figure 2(a) we conclude that the Naive Bayes crawler produces better collections at almost all points during the crawl. In fact a one-tailed t-test to check the same at various points during the crawl gives us a graph as shown in Figure 2(b). With the level of significance set to $\alpha = 0.05$ (shown as dotted line), the Naive Bayes crawler produces significantly better collections than the Breadth-First crawler for any crawl that is larger than a few hundred pages. Hence, there is a strong reason to use the topical crawler for building collections for our current application.

5.2 Information Discovery After crawling, clustering, linkage and lexical analysis the system generates an HTML report. The report contains a visual map of potential communities associated with a research paper and their characteristics. Examples of the graphical reports are shown in Figure 3. Each circle in the report corresponds to a cluster (potential community) and its size is proportional to the number of pages in the cluster. The lines between clusters show the strength of their relationship. The thicker the line the stronger the connection. The strength of the connection is measured by the cosine similarity between the cluster centers. When any of the circles is clicked a separate window reveals the details of the cluster such as the example pages (pages most similar to cluster center), cluster based hubs and authorities (see Figure 4). The user also has access to the paper for which the report was generated through a link on the corresponding report.

The visual map in Figure 3(a) corresponds to a paper titled "Combinatorial algorithms for DNA sequence assembly." We find that we have four well connected clusters (cluster 1,2,4 and 5). The largest one (cluster 5) represents a community that is primarily related to biotechnology. On the other hand, a closely connected cluster (cluster 1) concentrates on algorithms and data structures. Hence, we find a clear separation of related communities that would contribute to research associated with the paper. Cluster 2 is about more general computing literature and organizations. Even though cluster 2 is not so well connected to the biotechnology cluster (cluster 5), it is well connected to the algorithms and data structures cluster (cluster 1). Similarly we have a cluster representing related research primarily in university science departments (cluster 4). Figure 3(b) is about a paper titled "InterWeave: A middleware system for distributed shared states." Here we find a community that concentrates on architectures, operating systems and distributed computing (cluster 2). Another community (cluster 3) relates to Web sites that concentrate on reviewing technologies such as java and jini that could be used for distributed computing. Yet another (cluster 5) is associated with standards and technology companies such as CORBA, HP (Hewlett Packard) and OMG (Object Management Group). Figure 3(c) shows the map for a paper titled "Scalable session messages in SRM." Here again we find an interesting breakup of communities, where one concentrates on routing issues (cluster 3), while another concentrates on protocol documentation and RFCs (cluster 2), and a third that is associated with related software development and de-

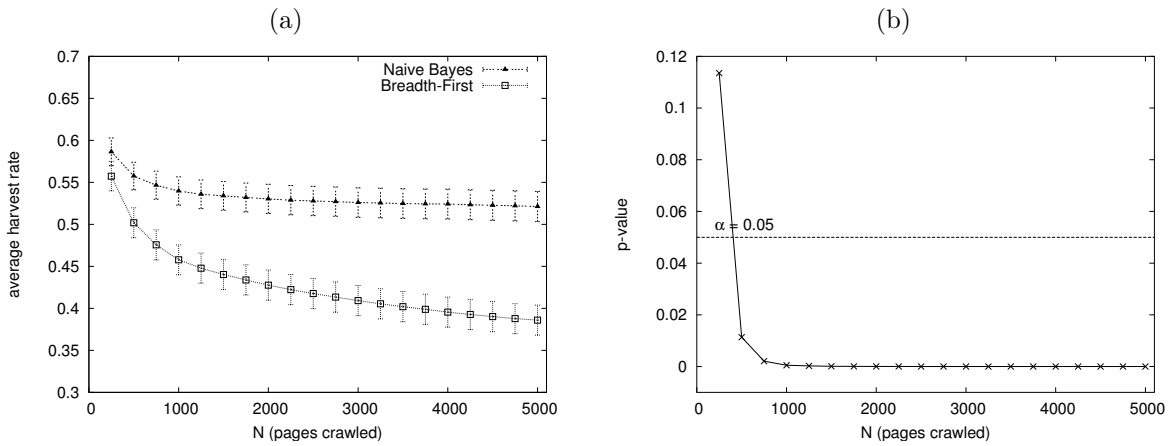

Figure 2: (a) Average harvest rates with pages crawled: error bars show ±1 standard error (b) p-values with pages crawled: Naive Bayes topical crawler significantly outperforms after crawling a few hundred pages

livery (cluster 1). The final example (Figure 3(d)) is for a paper titled "Extracting provably correct rules from artificial neural networks." Here we find a community that pursues abstract intelligence issues such as mind, philosophy and cognition (cluster 4). In contrast, cluster 3 is a community that applies intelligent algorithms for game playing and robotics. Another cluster that is very related to cluster 4 and also well connected to cluster 3 is one that concentrates on neural network based machine learning (cluster 5).

6 Related Work

With an ever increasing amount of documents available in digital format new ways to build and use digital libraries are being actively researched. CiteSeer [5] was one of the early systems that demonstrated the use of the Web to feed a digital library. In this paper we looked at a reverse problem of feeding a Web based resource discovery system using documents from a digital library.

Topical crawlers have been studied in great detail during the past decade [10, 7, 14, 11, 1, 12]. However, it is still a young area of research and innovative ways to apply topical crawlers are being invented. Chakrabarti *et al.* [7] were one on the earliest to apply bayesian classifiers built through examples for guiding a topical crawler. Topical crawlers have been applied to build collections for large scale digital libraries [3, 2]. Evaluating crawler performance, as well as the performance of the end systems that rely on it, is in itself an important research challenge [15].

7 Conclusion

We have described an approach that aims to connect the high quality information available in digital library documents with the high quantity of information on the Web. We hope that such an approach will bring structure to the vast amount of Web based information that relates to the communities associated with the documents in a digital library. Panorama, an implementation of the approach, demonstrates its feasibility. We find that not only can we guide topical crawlers with information from research papers, we can effectively use small collections (few thousand pages) downloaded by the crawlers to build visual maps that describe the semantics of the involved communities. The implementation for most parts uses well known machine learning, information retrieval and Web mining techniques.

In the future we would like to improve our cluster labelling technique to include labels that are phrases rather than words alone. Also, labels should maintain the case at least for what appear to be acronyms. For example, the word "dads" is used in the label for cluster 3 in Figure 3(a). It may not be apparent that is an acronym that stands for "Directory of algorithms and data structures." On a different note we would like to try various types of classifiers to guide the crawler, such as Support Vector Machines. Using some of the other extracted information, such as the author names and keywords, to train the classifiers is another extension to the current research. Also we would like to experiment with a wide variety of clustering algorithms based on both lexical features as well and links to discover potential communities.

Acknowledgements

Thanks to Weka group for the machine learning software and Walter Zorn (`www.walterzorn.com`) for the JavaScript libraries used for visualization.

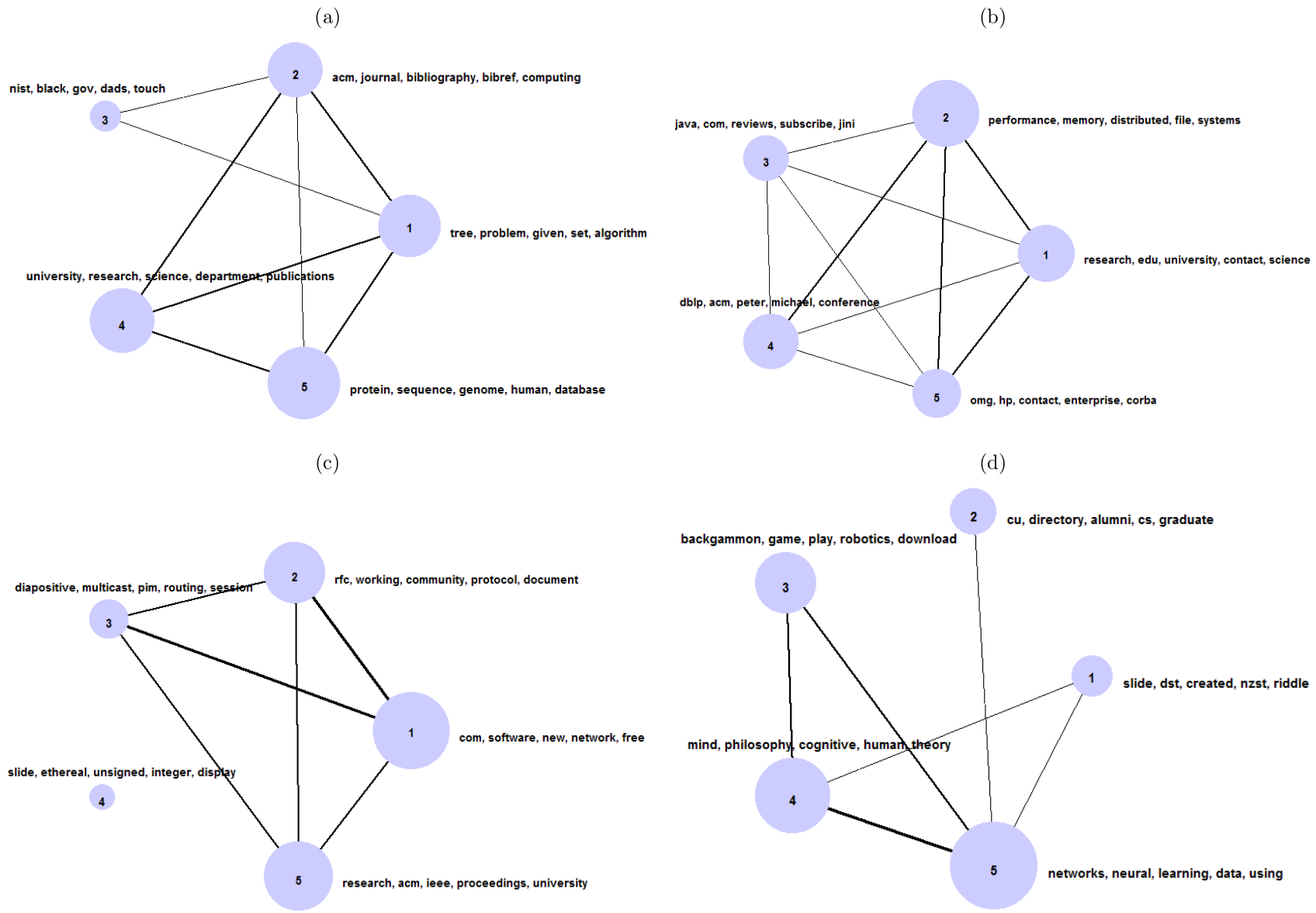

Figure 3: Visual maps of potential communities for papers titled (a) Combinatorial algorithms for DNA sequence assembly (b) InterWeave: A middleware system for distributed shared state (c) Scalable session messages in SRM (d) Extracting provably correct rules from artificial neural networks

References

[1] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *WWW10*, Hong Kong, May 2001.

[2] D. Bergmark. Collection synthesis. In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pages 253–262, 2002.

[3] D. Bergmark, C. Lagoze, and A. Sbityakov. Focused crawls, tunneling, and digital libraries. In *6th European Conf. on Research and Advances Technology for Digital Technology (ECDL 2002)*, 2002.

[4] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. In *Proc. 21st ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 104–111, 1998.

[5] K. Bollacker, S. Lawrence, and C. L. Giles. A system for automatic personalized tracking of scientific literature on the web. In *Digital Libraries 99 - The Fourth ACM Conference on Digital Libraries*, pages 105–113, New York, 1999. ACM Press.

[6] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW2002*, Hawaii, May 2002.

[7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[8] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks*, 30(1–7):161–172, 1998.

[9] D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. 15th ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 318–329, 1992.

[10] P. M. E. De Bra and R. D. J. Post. Information retrieval in the World Wide Web: Making client-based searching feasible. In *Proc. 1st International World Wide Web Conference*, 1994.

[11] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB 2000*, Cairo, Egypt.

[12] J. Johnson, K. Tsioutsiouliklis, and C. L. Giles. Evolving strategies for focused web crawling. In *Proc. 20th Intl. Conf. on Machine Learning (ICML-2003)*, Washington DC, 2003.

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

[14] F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2–3):203–242, 2000.

[15] F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.

[16] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[17] S. RaviKumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the Web graph. In *FOCS*, pages 57–65, Nov. 2000.

[18] G. Salton. *The SMART Retrieval System – Experiments in automatic document processing*. Prentice Hall Inc., Englewood Cliffs, NL, 1971.

[19] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[20] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. 14th Intl. Conf. on Machine Learning (ICML-97)*, pages 412–420. Morgan Kaufmann Publishers, 1997.

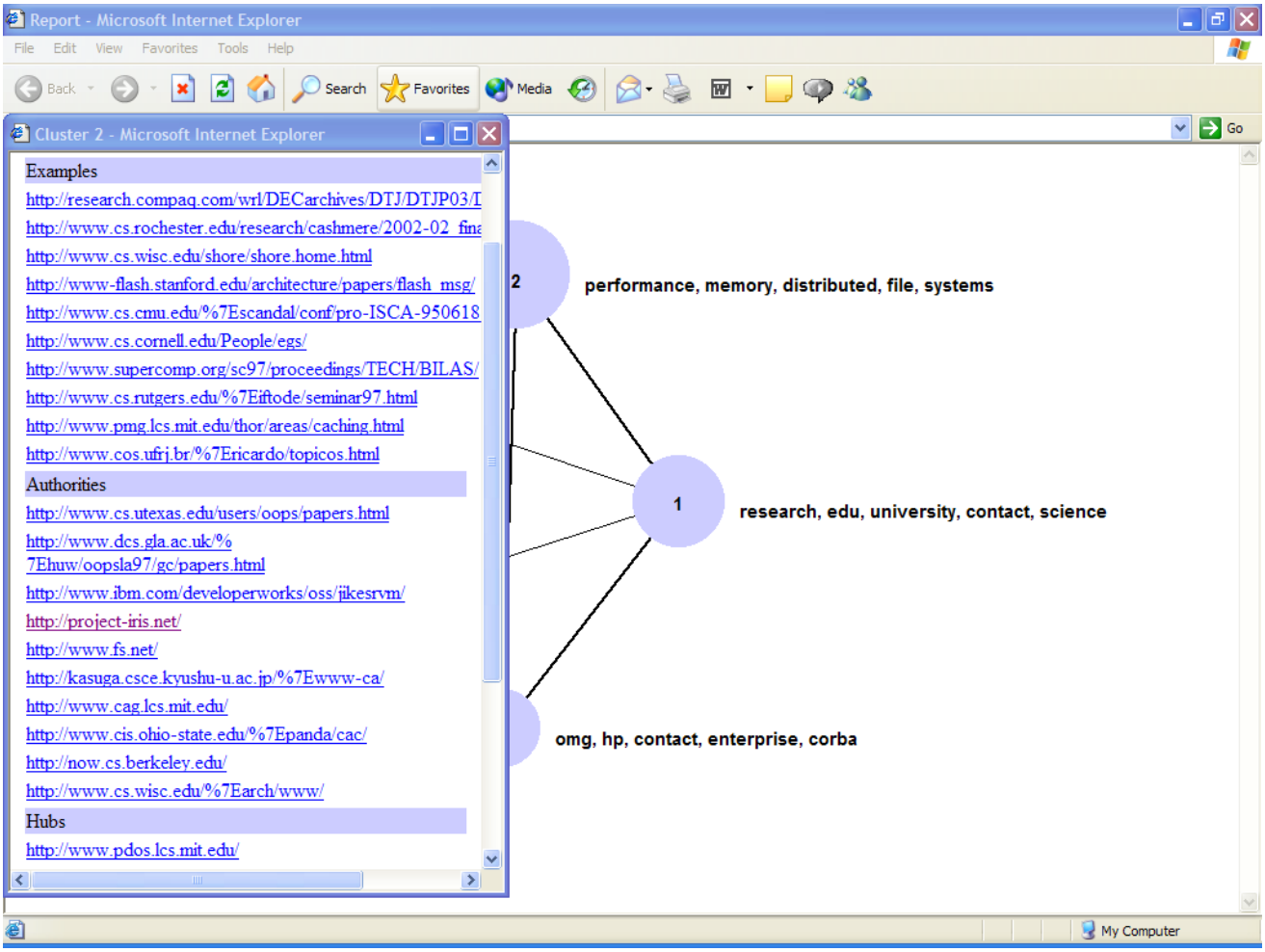

Figure 4: A sample HTML report with details of a cluster (cluster 2) that seems to concentrate on a community involved with operating systems, architectures and distributed computing.