Collection Synthesis

Donna Bergmark
Cornell Digital Library Research Group
Upson Hall
Ithaca, NY 14853
bergmark@cs.cornell.edu

ABSTRACT

The invention of the hyperlink and the HTTP transmission protocol caused an amazing new structure to appear on the Internet – the World Wide Web. With the Web, there came spiders, robots, and Web crawlers, which go from one link to the next checking Web health, ferreting out information and resources, and imposing organization on the huge collection of information (and dross) residing on the net. This paper reports on the use of one such crawler to synthesize document collections on various topics in science, mathematics, engineering and technology. Such collections could be part of a digital library.

Categories and Subject Descriptors

D.2.12 [**Digital Libraries**]; H.3.1 [**Information Systems**]: Information Search & Retrieval; H.5.4 [**Information Systems**]: Hypertext/Hypermedia; H.1 [**Models & Principles**]

General Terms

Experimentation, Design, Performance

Keywords

World Wide Web, NSDL, Mercator, topic management, crawling, information retrieval,clustering

1. INTRODUCTION

Digital libraries are made up of collections of on-line resources. In particular, we view collections as being a set of semantically related resources as represented by their URLs. Many on-line collections are hand-crafted, with federal support, and accompanied by a top page or portal. However, this approach does not scale. In order to control costs, and therefore to allow the long-term success and growth of digital libraries, we will have to rely increasingly on automated procedures, including ways to find or build collections.

JCDL'02, July 13-17, 2002, Portland, Oregon, USA.

This paper reports on our effort to automatically put together collections on scientific topics, starting only with the Web and a subject hierarchy. The general idea is to download pages from the Web in a focused way and classify them into the various topic areas. At the moment, the effort is highly experimental and the results are preliminary; however our approach can be varied in many ways until optimal results are achieved.

Section 2 describes the Web crawler we have at our disposal which is used to locate resources on the Web; Section 3 describes the digital library that is to benefit from this research and reviews other projects devoted to building topic-specific collections. Section 4 explains in more detail the concept of *focused crawling* and why it is superior to searching and clustering; it also explains our version of the focused crawl. Section 5 briefly covers the implementation of our focused crawl in Mercator; Section 6 contains our current results. The paper concludes with ideas of where to go next.

2. MERCATOR

To automatically build collections of topic-related Web documents, whether by searching or clustering, one must start with a Web crawler. Mercator[1] is a powerful, inherently parallel, and extensible Java-based Web crawler developed by System Research Center (SRC) [27, 37] in Palo Alto. Because Mercator can be configured in many different ways, it is a perfect vehicle for exploring various Web crawling strategies for collection synthesis. Thanks to Compaq, we have access to Mercator and thus join the list of researchers who have done interesting work with this crawler [10, 25, 26, 38, 41].

Figure 1, adapted from [37], shows conceptually how Mercator works. Mercator itself has no special personality. It is simply fast, scalable and distributed. Its job is to download pages, run one or more analyzers on each downloaded page, and optionally enqueue child links for further downloading.

The loop shown in Figure 1 is executed continuously by typically hundreds of threads, all of which belong to a single instantiation of a Java *Crawl* object, henceforth called "the crawl". The primary data object in the crawl is the URL frontier; it is shared by all the threads.

The researcher has several opportunities to add "flavor" to the crawl. The first opportunity is the analysis stage, where a sequence of analyzers A_1, A_2, ... A_n are applied to each downloaded page. Each A_i is an extension of Mercator's base Analyzer class, which consists of a constructor

[1] `<http://www.research.compaq.com/SRC/mercator/>`

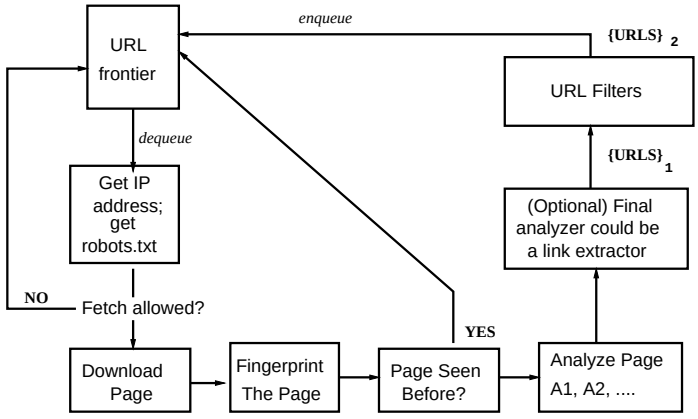

Figure 1: System block diagram of the Mercator Web Crawler. Main components are: URL frontier, a DNS resolver, an HTTP downloader, a Document Fingerprinter, and a Link Extractor.

called once per crawl, and a *process* method called for each downloaded page. The process method gets the content of the page as well as information about the page, such as its URL.

Which analyzers to apply and in which order are specified in a configuration file. The final analyzer might be the Link Extractor, supplied with Mercator, which extracts links from the downloaded page and sends them on to the URL filtering stage.

At this point it might be mentioned that the extracted child links are derelativized and encapsulated into a *DocBundle* object. During page processing, user-specified information can be added to the DocBundle.

The URL filters are several. Some filters can simply be encoded in the configuration file, e.g. to discard URLs from particular domains. The remaining URLS ($URLS_2 \subset URLS_1$) are passed to the *URLSet* object, which has a chance to add parent information to the DocBundle and if desired, pick a download priority for this DocBundle. It then goes to the *Frontier* object which adds the DocBundle to the proper queue. The user can override URLSet and Frontier by naming custom classes in the configuration file.

The machine which runs Mercator is accessed via OpenSSH Version 2. All of our work is done remotely from Cornell to the SRC in Palo Alto. We have noted that if the only analyzer used is the Link Extractor, Mercator can download more than 400 pages per second when running 500 threads.

In general, all of our programming additions have been made by extending basic Mercator objects rather than altering any of the Mercator code. This is a testimony to the clean design and construction of the Java implementation of Mercator.

3. THE NSDL PROJECT

The Cornell Digital Library Research Group is involved in a large project with other institutions – the National Science Digital Library[40, 47]. By Fall 2002 this project hopes to have a couple hundred scientific collections on-line, organized, and searchable.

Hand-curated collections are valuable but costly. In the beginning they will form the bulk of the NSDL. However, the long-term success of the NSDL relies on automated techniques for forming and maintaining content. Therefore we must come up with ways in which collections can be generated automatically, even if they are initially of lesser quality than funded collections carefully built by trained researchers.

Having access to a powerful Web crawler gives us a platform for experimentation, so we decided to see how far we could go in automatic collection acquisition. There are two ways to use a crawler to come up with collections: one is to synthesize the collection from resources on the Web; the other is to discover existing collections on the Web. This paper focuses on collection generation.

There are two objectives for the automated collection building project:

- Build a number of collections with minimum user input because automation is the key to future digital library projects [1, 42].
- Aim for limited-sized collections of high precision with respect to the topic *and* which are educational. Collections of about 20 to 50 good documents would be best.

3.1 On Automatic Collection Generation

Fortunately, there has been copious research in recent years on building digital collections of semantically related materials.

The earliest collection builder, starting in 1994, is probably "Harvest" [6, 7]. The motivation of the Harvest project [17] was to index community- or topic-specific collections. Harvest was a selective document gatherer, which took source data off the Web, indexed it, and redistributed it. Harvest used the Mosaic browser at a time when the Web was tiny. An experiment at NCSA used Harvest to recognize and then index over 7,000 WWW home pages among the several hundred-thousand documents available then on the Web. Harvest could be configured to automatically collect and summarize related objects from around the Internet into a large collection or to collect, summarize, and hand annotate a small, specialized collections.

Harvest was the forerunner of the much more sophisticated and functional Greenstone system [44, 45]. Greenstone is a complete Digital Library system, one part of which is "the Collector." Given a collection of sources (a URL for a Web site, an ftp URL for a repository, or a file URL for a directory), the Collector will convert them to XML, index them, and extract metadata. The software is available at `<http://nzdl.org>`.

Chekuri *et al.* [16] start with broad topics selected from the Yahoo subject category. They start with a training set of pre-classified pages for each topic. Then they build a term vector[2] for each topic, and then crawl, outputting an ordered list of categories for each downloaded page, which allows them to categorize a page into several different topics.

In 2000, Kluev and others at the University of Aizu added topic-based collection building to the OASIS distributed Web crawler [29]. The aim was to produce some high precision, high recall collections about computer science. Results for building collections about "Algorithms" and "Programming

[2]The words *term vector* and *document* vector space will be explained later.

Languages" are reported. Their collection builder starts with a set of human-selected papers relevant to the topic. As papers are downloaded, they are converted to plain text from Postscript, HTML, and/or PDF; then 4 or 5 relevance figures are computed (including term similarity, phrase count, etc). The children of the downloaded paper are placed in a priority queue depending on a combination of the parent's metrics. Furthermore, papers are categorized as "undoubtedly relevant", "probably relevant", and "need to be checked". A random human sampling at the end determines the goodness of the collection. Hence their collection builder is a combination of automatic and nonautomatic techniques.

Cora is a Web crawling system that creates domain-specific portals [32], but is also not completely automatic. They train their search engine on *one* topic, using a set of (presumably recent) research papers and then use references and hyperlinks to build a larger collection on that topic.

Mukherjea [34, 35] describes one system, WTMS (Web Topic Management System) which computes hubs and authorities[28] for a single topic. It uses a representative document vector based on the terms contained in the seed URLs (or search responses for a query with specified key words).

In general, our approach is different from these projects for two reasons: we are after narrowly focused collections, whereas many of these projects started with quite broad queries and divided the web into topic areas; secondly we plan to build many collections at once, rather than one at a time.

4. SEARCHING VS. CLUSTERING

There are a number of possible techniques for collection synthesis. The three main approaches are: search engines[2], clustering[9], and focused crawls[15]. Focused crawls are a combination of search and clustering with additional technology added into the mix.

4.1 Search Engines

An obvious question to ask is why does the NSDL not simply pose its various topics of interest to a search engine and make a collection out of the top 50 results? Search engines work by crawling the Web offline, indexing parts of the downloaded text, building an efficient inverse index, possibly computing page ranks based on the link structure of downloaded documents, and then retrieving "important" URLs which overlap the query.

The problem is that search engines index only a very small portion of the Web [30], and cover all topics. While search engines aim for uniform coverage of the Web, no search engine can download and index the entire Web. Thus the coverage of any particular topic is relatively small.

Another problem with many if not all search engines is that in the interests of efficiency, they examine only a part of the text, such as only the header data or the first paragraph.

Finally search engines are susceptible to search engine persuasion and keyword spamming [31]. It is much harder to waylay an analyzer of complete page content by arranging the words so that the page looks more attractive to the collector.

4.2 Clustering

Clustering has long been a favorite information retrieval technique for obtaining related subgroups from a larger, amorphous collection of documents. Clustering is inherently different from searching. The latter starts with a topic and returns (ideally) a related result. The former starts with a corpora and detects topic clusters.

A number of researchers have used clustering to build topic-based collections. The idea here is that given corpora of documents, one can divide it into clusters of related documents, hopefully one cluster per topic. The cluster-building process can be parameterized as to how big the clusters may be, how many there will be, and whether or not overlap is allowed. In classic information retrieval, after the clusters were built, a *centroid* was computed for each cluster. This was the surrogate for the cluster and was meant to speed up searching, make classification of new documents more efficient, and to describe the cluster.

P. Willet [43] is an excellent survey of how clustering is used for collection building. The basic types of classical clustering methods are:

1. hierarchical grouping, starting with a complete similarity matrix of distance between documents in the collection (also called agglomerative clustering)
2. iterative partitioning, starting with a full collection
3. one pass methods, which cluster all the documents on the fly by updating centroids as documents are dropped into the clusters or start new clusters

Obviously methods 1 and 2, which start with the complete corpora, are unsuitable for classifying Web documents, simply because the Web is not finite. The same characteristic of the Web rules out iterative or multi-pass methods, such as method 2. Method 3 is promising but it is not clear what centroids to start with.

4.3 Focused Crawling

Since we wish to build a large number of topic-specific collections, neither searching nor clustering seem to fit the bill. Focused crawling can be used like a search engine, but rather than attempting to download every possible document on the Web, it can get further, faster by narrowing its crawl to specific topics.

This approach was introduced by Chakrabarti in 1998 [12, 13]. Since then a number of variations have been introduced [4, 5, 8, 11, 18, 20, 22, 29, 31, 33]. The unifying theme is efficiency (see Figure 2). Suppose 30% of the current Web is downloaded by the search engine's crawler and by the focused crawler. When looking at a particular topic (the dotted area on the right), the search engine's result contains fewer relevant documents than the collection returned by the focused crawler.

The main problem is keeping the crawl focused, and therefore efficient. The two main focusing techniques are *link structure* analysis and *content* analysis.

Link analysis is based on the theory that a document pointed to by one document is likely to be similar to it, or that two documents linked to by the same document might be similar to each other [19]. Content analysis looks at the word similarity between documents [46]. This is based on the premise that two documents related to the same subject will use the same words [39]. There are hybrid approaches which combine link and text analysis [13, 24].

The point of both link and content analysis is to keep related documents together. This leads naturally to the concept of a "distance" metric; in the past researchers have

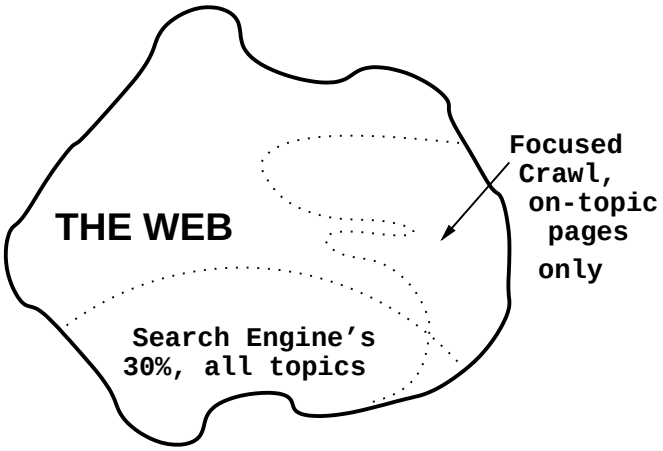

Figure 2: Focused Crawl vs. Search Engine

used citation relationships [21], term similarity [3, 39], and co-authorship [36]. For content analysis, we use term similarity. For link analysis, we follow links only from pages that are deemed relevant to one of our topics.

4.4 A Collection Synthesis Approach

Our basic approach to collection building is accretion, or agglomeration (see Figure 3).

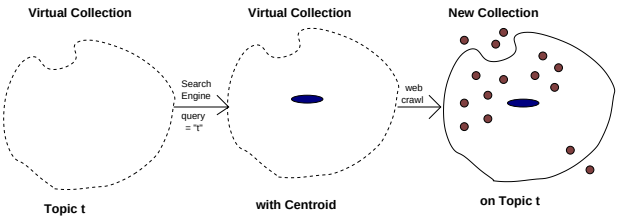

Figure 3: One way to build collections.

First, we assume that for any topic t, a virtual collection of online documents exists about that topic. We select a few authorities from this collection (using a search engine) and from them we construct a *Centroid*.

We build several collections at the same time, starting with a number of centroids for a number of distinct (virtual) collections. This keeps the crawl efficient.[3]

Once the centroids have been defined, a massive Web crawl is performed. The right part of Figure 3 illustrates what we might have after a partial crawl. Each down-loaded document is matched against all the centroids using some distance measure.[4] As the figure illustrates, some documents fall close to a given centroid, most documents are not close to any of the centroids. At the end of the crawl, we hope to have as many different collections as centroids.

[3]Because document download time is the major factor in crawl speed, it is more efficient to do a single massive crawl classifying documents into one of many classes rather than to do a number of individual crawls, one per topic.

[4]Actually, we download pages. The definition of "document" is tricky. If a page's filename is `node20.html`, for example, we can assume that it is part of a larger Latex document.

As a distance metric, we use the cosine correlation and the term vector and document vector space model:

$$corr\left(\vec{c},\vec{d}\right) = \frac{\vec{c}\cdot\vec{d}}{||\vec{c}||_2}$$

where

$$c_i = \text{weight of term } i \text{ in the centroid } \vec{c}$$

$$d_i = \text{weight of term } i \text{ in the document } \vec{d}$$

Term i means this is the ith word in the crawl's dictionary. Weights are computed by the classical TF-IDF technique [3, 23, 39]:

$$idf_i = log\left(\frac{N}{df_i}\right)$$

$$d_i = f_i \times idf_i, \text{ normalized by } ||\vec{d}||_2$$

N is the number of documents (or search results) used to construct the centroids, and df_i is the document frequency of term i. Belew [3] summarizes other metrics that could be used here.

The cosine correlation yields a value between 0.0 and 1.0, assuming that the weights in $\vec{d}$ have been normalized so that $||\vec{d}||_2$ is 1.0. The weights in the centroid are not normalized, which is why we divide the inner product by the centroid's norm.

The crawl's *Dictionary* is the union on the terms found in the centroids.

5. FOCUSED CRAWL WITH MERCATOR

It is possible to configure Mercator in many different ways to accomplish a variety of focused crawls. The ways to configure Mercator fall into three general categories: how to start a crawl, how to stay on topic, and how to stop a crawl. This section describes our configuration.

5.1 Where To Start

In Mercator, the `SeedURLs` parameter in the configuration file gives the seed URL[s] at which the crawl should start. These are immediately added to the frontier. It is unclear what makes a good seed URL, but intuitively it should be rich in links, yet not too broad in scope. Yahoo is the Mercator default. We added a few other sites, such as `http://www.search4science.com`. An interesting experiment would be to start with the top Google search result for each topic.

Our focused crawl begins armed with a dictionary and a set of centroids, constructed as explained in Section 4.4. These are loaded at the start of the run during instantiation of the first analyzer A_1 (the Term Extractor). Which files to load are specified by two new configuration file parameters "Dictionary" and "Centroids".

`TermExtractor` is an extension of Mercator's base Analyzer object. Its process method is programmed to do the following with the DocBundle and page content passed to it:

1. Extract all text strings between ">" and "<".
2. Split the text string into individual words.
3. Remove any words that are in the stop list.

4. Add/update the word in a hashtable of word frequencies. The key of the hash is the word, and the value is the frequency of the word in this document.

5. Add the completed hashtable to the DocBundle.

The words are not stemmed, mainly because stemming is a recall-enhancing technique, and we are more interested in precision.

5.2 Selecting Documents for the Collection

The DocBundle is passed on to the next analyzer, an extension of Mercator's Link Extractor. This new subclass is programmed to use the hashtable in the DocBundle to compute the cosine correlation between the downloaded document and each of the centroids. The document is classified with that centroid yielding the highest correlation value. The correlation is the *degree* to which the document belongs to this collection.

It is well known that document length is important. True, we normalize the document vectors to length 1 but if there is only one term in common with the centroids, then it will automatically be classed with the centroid in which that term happens to have the greatest weight. Since this sort of categorization is quite arbitrary,[5] we made the decision to ignore documents that had fewer than 4 words overlap with the centroids. This value is tunable.

Downloaded documents frequently contain terms not included in the dictionary. Unlike classical Information Retrieval, we are not obliged to use all the significant words in the collection. We are interested only in documents that overlap our centroids, so we simply ignore new words. The dictionary remains small, and thus does not slow up the crawl.

Term discrimination values of existing words, however, could be adjusted on the fly. In this way, some words which appeared to have a high discrimination value with respect to the centroids could be adjusted downwards. An advantage of Mercator is that we can try both approaches: keep the Dictionary fixed, or update it as documents are downloaded.

5.3 Stopping the Crawl

A crawl can be viewed as traversing a tree of links. It is a tree, because Mercator never downloads the same URL twice in a given crawl and so there can be no loops. (Mercator also automatically detects and avoids crawler traps, which generate an infinite series of URLs.)

The question then becomes how far down the tree to go before abandoning a path. One may well be willing, in a focused crawl, to traverse several less valuable pages in order to get to one that is valuable. How many pages is controlled by a two new Mercator parameters, *cutoff* and *threshold*. The threshold determines the difference between a "nugget" (a page with cosine correlation above the threshold) and an off-topic page. The cutoff is the limit on the number of off-topic pages are allowed before abandoning a crawl down a given path in the tree.

The cutoff can be anywhere from 0 (in which case no links of an off-topic page are followed) to the diameter of the Web (in which case you'll look across the whole Web if necessary to find the next nugget).

An area for future research is to dynamically adjust the cutoff and threshold depending on what has been learned in the crawl so far. This would represent a learned response for how far to peer into the future to find the next relevant item.

Other stopping criteria include: a list or description of links never to follow (e.g. "next", "previous", "about"); a list of URLs never to download (already supported by Mercator URL filters); when a collection is big enough (at which time its centroid may be ignored in future comparisons).

The overall crawl (the union of all the individual crawls) is also subject to stopping criteria. Mercator offers the choice of time limit and empty frontier. Another (uncontrollable) stopping condition of course is failure of Mercator or the computer it is running on. For this reason, Mercator has support for check-pointing. Thus it is possible to make multi-day crawls.

6. RESULTS

Our current experiments start with a topic hierarchy in mathematics extracted from `mathforum.org/library/toc.html`. This yielded 26 leaves about mathematics, an example of which is topic 24: "Probability/Statistics Stochastic Processes". The top 7 URLs returned by Google in response to this query were the following (formatted as the SeedURLs clause for a Mercator configuration file):

```
(SeedURLs
(http://www.dam.brown.edu/graduate/handbook/node19.html
 http://ubmail.ubalt.edu/~harsham/statistics/REFSTAT.HTM
 http://www.santafe.edu/~shalizi/prob-notes/
 http://www.amazon.com/exec/obidos/ISBN=0070484775/
 http://www-2.cs.cmu.edu/~harchol/Perfclass/Booklist/...
 http://science.ntu.ac.uk/msor/research/
 http://www.siam.org/journals/tvp/Tvp.htm
))
```

Each of the seven page sets were downloaded by Mercator and run through the TermExtractor and WordCounter analyzers.[6] The term vector, or centroid, for topic 24 in order by descending term importance for this class is:

```
(statistics,probability,statistical,stochastic,...)
```

For centroid weights we chose term frequency × document frequency / max(7,|result set|). We did not construct a centroid if the result set had less than 4 items. Each of the 26 centroids was truncated to its 20 highest weight terms (we also tried the 40 highest, see Sections 6.1 and 6.3); these terms combined to make a dictionary of 258 words, weighted by TF-IDF as explained earlier. Details of the resulting dictionary and centroids follow.

6.1 The Dictionary

For a faster crawl, we wish to have a fairly small dictionary containing words pertinent to this area (here, mathematics) with a range of IDF, or discrimination, values. We built two different dictionaries, based on different truncation levels on the centroids. As shown in Section 6.3, the dictionary based on 20 terms resulted in at least as many highly correlated documents as the one based on 40-term centroids, and ran much faster. Words in the larger dictionary that are not in the smaller one are for example "worksheets", "world", "write", "understand", which tend to bring in more off-topic

[5] that single word could be "university", for example

[6] `WordCounter` is our Analyzer subclass that keeps a global histogram of all the term frequencies and writes it to a file at checkpoint time.

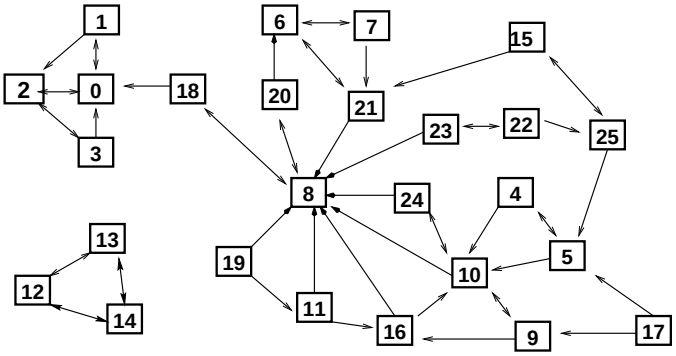

Figure 4: A graphical layout of 26 centroids based on their similarity. Each directed arrow points from a centroid to the two centroids closest to it. That is, edge (i,j) means that of all cosine correlations (i,k), centroid j was one of the two closest to i.

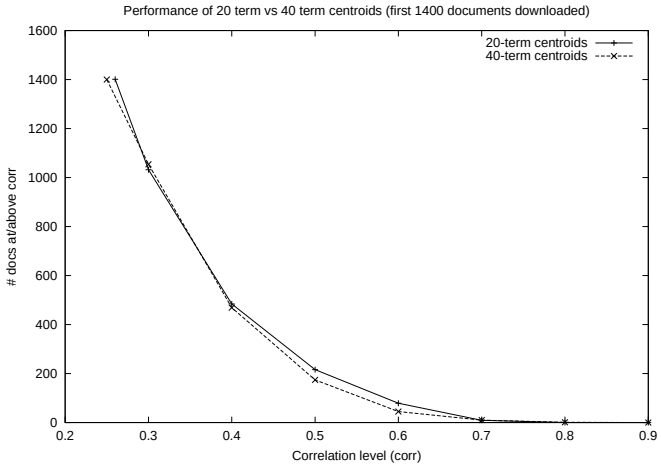

Figure 5: Relative efficiency of 20- vs. 40-term centroids. The 20-term case finds slightly more highly correlating documents during the first 1400 downloaded than does the 40-term set. The 20-term case with its shorter centroids and smaller dictionary is also faster: it downloads more than 5000 documents in 8 minutes vs. 2100 for the larger centroids.

pages.

Dict = 258 words

	min	max	avg
df	1	82	9.1
tf	4	804	87.76
IDF	.319	2.23	1.46
|C|	20	20	20

26 Centroids

Dict = 582 words

	min	max	avg
df	1	101	7.87
tf	2	1089	60.18
IDF	.33	2.34	1.69
|C|	40	40	40

36 Centroids

The range in IDF is about the same for the two cases, so from that point of view, cutting down on the dictionary size doesn't hurt. All of the words in the original topics got into the smaller dictionary except for three non-discriminators, like "other".

6.2 The Centroids

Note that the centroids are built automatically. We are currently ruling out human intervention in this step. For example, relevance feedback from a human looking at the search results for a particular topic is considered too expensive. We don't even allow a few training runs (in the AI sense) followed by automatic centroid generation.

Research on what makes a good centroid for retrieval purposes is scarce. Intuitively, the centroids should not be too close or overlap. We measured this by computing the complete similarity matrix of cosine correlations between the vectors. Figure 4 shows for each centroid the two centroids most similar to it. The top left area is about algebra, the middle top is about calculus, and the unconnected component is about geometry. Surprisingly, the center (8) is about partial differential equations.

Although the graph looks nicely laid out, we did another test on the centroids by using our Mercator classification code (with cutoff 0 and threshold 1.0 to prevent a crawl). We downloaded the approximately 150 centroid URLs; at least 90% of them were classified with "their" centroid, leading us to believe that there was enough space among the centroids to have a hope of accreting useful URLs on their topic.

6.3 The Crawl

The crawl was performed for 8 minutes, with a threshold of 0.3 and a cutoff of 0. Thus the crawl was focused to those links occurring on pages correlating 0.3 or higher with the centroid closest to it. We assembled 26 collections of HTML, PDF, and PostScript documents related to topics in our Mathematics term hierarchy.

Crawl performance is measured in two different ways. First, plotting correlation vs. download order gives a measure of the *efficiency* of the crawl. Assuming that high correlations imply relevance to the topic, the more efficient crawl will download highly relevant documents early in the crawl, and more often. A different measure, *precision*, will be explained later.

Our initial crawls were meant to determine whether long centroids would be better than shorter ones (see Figure 5). A snapshot of the first 1400 downloaded documents above 0.25 cosine correlation shows that the 20-term case was as good as (if not better than) the 40-term case.

Next we concentrate on just one of the collections found in the crawl, to get an idea of whether high correlations actually mean anything. (Ideally, high rank should imply high relevance.) The relevance judgments are of course highly subjective, but were not too difficult to make. With relevance judgments in hand, we can measure the precision of the crawl.

In classical information retrieval, precision was defined as the number of relevant retrieved vs. number of documents retrieved. Here, however, we can download as many documents as we wish, as long as we find some relevant ones. Precision is therefore defined as number relevant vs. *rank*, where documents are ranked in decreasing correlation order. For example, a collection of size 1 would simply contain the document which correlated most strongly with the collection's centroid. If that document were relevant, the precision is 1.0, otherwise 0. If half the items in the collection are relevant, the precision is 0.5.

Figure 6 shows the results for the topic query "plane geometry euclidean triangles and other polygons". The highest three ranking documents were relevant to the topic, which

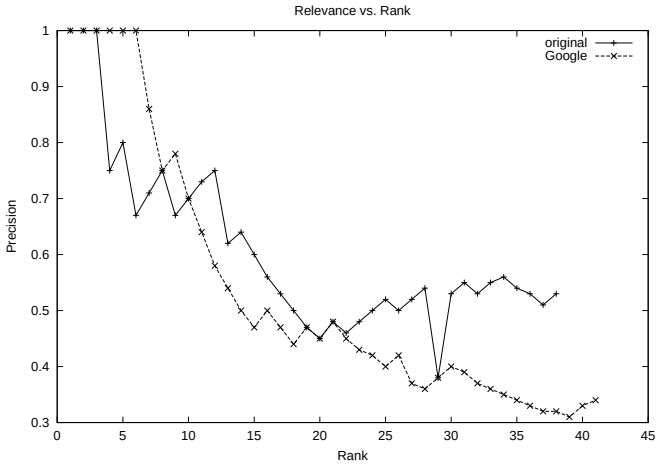

Figure 6: Precision plot for the geometry collection based on a 4-minute crawl in which 5785 documents were downloaded which had a correlation between .25 and 0.53. Precision seems to be stabilizing around 0.5. For comparison, the precision for the first 40-some search results on a Google search of the same query is shown.

is good. In this short crawl, however, the collection precision is only 0.5 (the final point in the plot). It is expected that lengthening the crawl will lift the precision curve, but that has yet to be tested. For comparison purposes, we also did a Google search on this query, made relevance assessments for the first 40-some results, and plotted the resulting precision. Google starts off with amazingly high precision but then quickly lapses into irrelevant topics, such as course descriptions.

6.4 New IDFs

Closer inspection of some of the results indicated that calculating IDF values only on the basis of the centroids was not optimal. Hence we did a 4-minute crawl with the initial dictionary, getting term frequencies and document frequencies from approximately 10,000 documents. Those frequencies were used to recompute the term IDF's. The range expanded, and some words moved up in value, others down, but the size of the dictionary (necessarily) remained the same. The picture for the first 1400 documents is in Figure 7, which can be compared with the original dictionary's crawl in Figure 8. It would appear that the recomputed IDF's had a good effect on discriminating good documents from bad. However, it is not clear that precision was improved, even though higher ranks were retrieved. Figure 9 shows precision using the original term weights vs. the updated weights.

7. CONCLUSIONS AND FURTHER WORK

It is absolutely necessary that if libraries like the NSDL are going to scale, automatic methods such as those discussed here are required. We have shown plausibility of building collections using a crawler. We have knobs and scope for investigating all sorts of variations to our collection synthesis approach. Our main goal now is to lift the precision curve above the 50% mark. Here are some possible

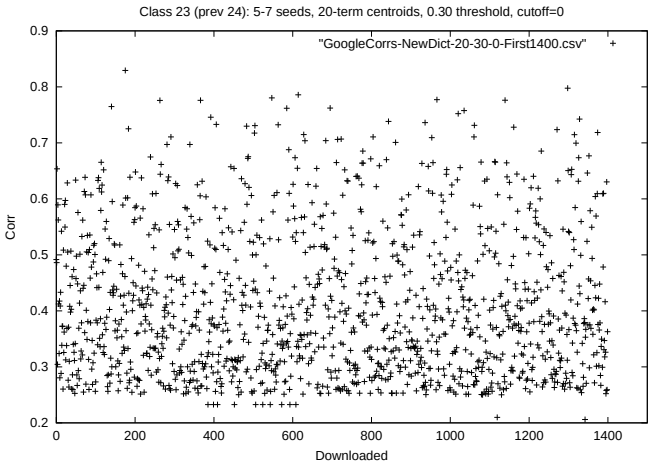

Figure 7: Results for the first 1400 downloaded documents above 0.25, using recomputed IDF weights. In this point plot, the dots represent the similarity of the document with its closest centroid, in the order in which documents were downloaded (the order is approximate since 500 threads were being run at once). The Y-axis is similarity.

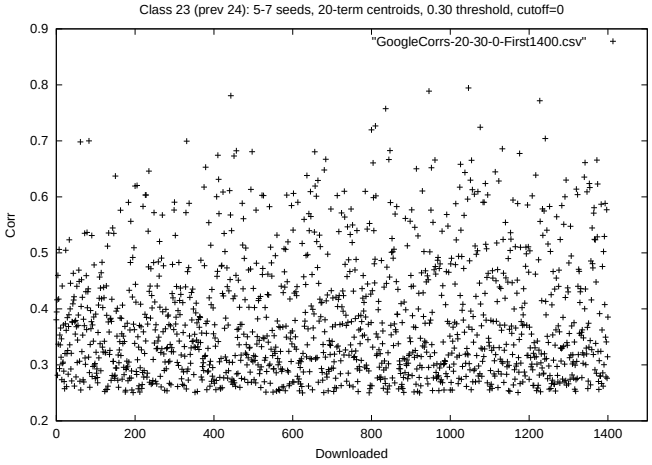

Figure 8: Results of a crawl with the original dictionary based on the 20-term centroids. The first 1400 documents downloaded that have a correlation > 0.25 with their closest centroid are shown. In general, not as many documents have high correlation, and the point-plot is darker to the bottom, meaning that more junk was retrieved.

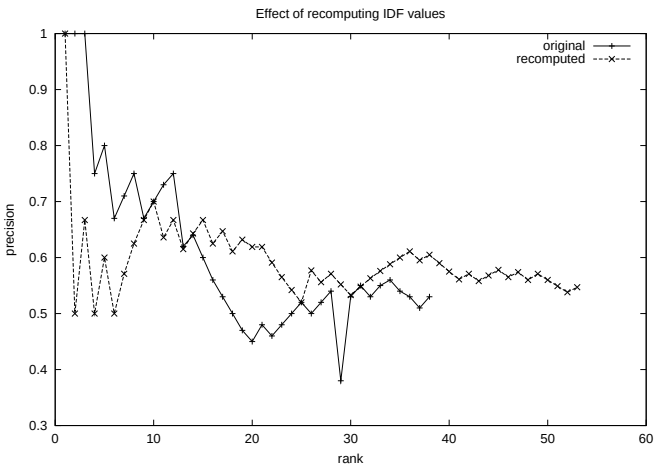

Figure 9: Original IDF values vs. recomputed IDF values. Recomputation resulted in higher ranks and apparently higher precision.

next steps:

- Let the clusters overlap. For example, if a document correlated very highly with two centroids, then put it into both collections.
- Vary the starting place by selecting different SeedURLs for the crawl.
- Vary the threshold; so far the cosine correlation 0.3 looks like a good line between "could be relevant" and "probably not relevant".
- Vary the cutoff. So far 0 is looking best.
- Update dictionary IDF values on the fly. Initial results show this might be something worth implementing.
- Vary the initial centroid length. So far 20-term centroids look better than 40-term centroids.
- Vary the minimum acceptable overlap between document terms and centroid terms.
- Chakrabarti's paper on CLEVER [14] suggests 3 further things we might try (prioritize links so off-site links are higher; don't follow links from documents almost identical to ones previously seen; use only a single point-of-entry into any given site).

8. ACKNOWLEDEGMENTS

This work was funded in part by the NSF grant on Project Prism, IIS 9817416. Thanks go to Bill Arms for suggesting this project, and to Carl Lagoze for improving the clarity of this presentation. We also acknowledge considerable technical help from the Systems Research Center at Compaq.

9. REFERENCES

[1] W. Arms. Automated digital libraries: How effectively can computers be used for the skill tasks of professional librarianship. *D-Lib Magazine: The Magazine of Digital Library Research*, July 2000. <http://www.dlib.org/dlib/july00/arms/07arms.html>.

[2] A. Arvind, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43, Aug. 1002.

[3] R. K. Belew. *Finding Out About*. Cambridge Press, 2001.

[4] I. Ben-Shaul, M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalheim, V. Soroka, and S. Ur. Adding support for dynamic and focussed search with Fetuccino. In *Proceedings of the Eighth International World-Wide Web Conference*, pages 575–588, Toronto, Canada, May 1999. Available: <http://www8.org/w8-papers/5a-search-query/adding/adding.html> (current as of August 2001).

[5] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111, Aug. 1998. Available: <http://www.acm.org/pubs/proceedings/ir/290941/p104-bharat/p104-bharat.pdf>.

[6] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, July 1994.

[7] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system, 1994. Additional information available <http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/schwartz.harvest/schwartz.harvest.html>.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, Brisbane, Australia, 1998. Available online at <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm>, (current as of 28 Feb. 2001).

[9] A. Broder, S. Glassman, and M. Manasse. Clustering the Web, 1999. Available: <http://www.research.compaq.com/SRC/articles/199707/cluster.html>.

[10] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proceedings of the Sixth International World-Wide Web Conference, Santa Clara, Ca.*, pages 391–404, Apr. 1997. Available: <http://www.scope.gmd.de/info/www6/technical/paper205/paper205.html> (current as of September 2001).

[11] J. Carriere and R. Kazman. WebQuery: Searching and visualizing the Web through connectivity. In *Proceedings of the Sixth International World-Wide Web Conference*, pages 701–711, Santa Clara, Ca, Apr. 1997.

[12] S. Chakrabarti. Recent results in automatic Web resource discovery. *ACM Computing Surveys*, Dec. 1999. Available: <http://www.acm.org/pubs/articles/journals/surveys/1999-31-43es/a17-chakrabarti/a17-chakrabarti.pdf>.

[13] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh*

International World-Wide Web Conference, pages 65–74, Brisbane, Australia, Apr. 1998. Available: Computer Networks and ISDN Systems special issue, 30(1-7).

[14] S. Chakrabarti, B. E. Dom, D. Gibson, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Experiments in topic distillation. In *Proceedings of the ACM SIGIR Workshop on Hypertext Information Retrieval on the Web*, Melbourne, Australia, 1998. ACM. Available: <http://www.almaden.ibm.com/cs/k53/abstract.html>.

[15] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. In *Proceedings of the Eighth International World-Wide Web Conference.*, pages 545–562, Toronto, Canada, May 1999. Available: <http://www8.org/w8-papers/5a-search-query/crawling/index.html> and <http://www.cs.berkeley.edu/~soumen/doc/www99focus/> (current as of August 2001).

[16] C. Chekuri, M. Goldwasser, P. Raghavan, and E. Upfal. Web search using automatic classification, 1997. Available at <http://cm.bell-labs.com/who/chekuri/postscript/web.ps.gz> Current as of December 5, 2001.

[17] F.-C. Cheong. *Internet Agents: Spiders, Wanderers, Brokers and Bots*. New Riders Publishing, Indianapolis, Indiana, 1996. ISBN:1-56205-463-5.

[18] B. D. Davison. Topical locality in the Web. In *Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval (SIGIR 2000)*, Athens, Greece, July 2000. ACM.

[19] J. Dean and M. R. Henzinger. Finding related pages in the world wide web. In *Proceedings of the Eighth International World-Wide Web Conference, Toronto, Canada*, pages 575–588, May 1999. Available: <http://www8.org/w8-papers/4a-search-mining/finding/finding.html> (current as of August 2001).

[20] M. Diligenti, F. Coetzee, S. Lawrence, C. . Giles, and M. Gori. Focused crawling using context graphs. In *Proceedings of the 26th International Conference on Very Large Databases*, 2000.

[21] E. Garfield. *Mapping the structure of science*, pages 98–147. John Wiley & Sons, Inc. NY, 1979. Available at <http://www.garfield.library.upenn.edu/ci/chapter8.pdf>.

[22] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring Web communities from link topology. In *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space – Structure in Hypermedia Systems (*HYPERTEXT'98, PITTSBURGE, PA, pages 225–234, June 20–24 1998.

[23] E.-H. S. Han and G. Karypis. Centroid-based document classification: Analysis & experimental results. Technical Report 00-017, Computer Science, University of Minnesota, Mar. 2000.

[24] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the Web. In *WebDB'2000: Third International Workshop on the Web and Databases*, May 2000. Available <http://www.research.att.com/conf/webdb2000/PAPERS/8c.ps>.

[25] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform URL sampling. In *Proceedings of the 9th International World Wide Web Conference: The Web: The Next Generation*, Amsterdam, May 2000. Elsevier. Available: <http://www9.org/w9cdrom/88/88.html>.

[26] M. R. Henzinger, A. Heydon, M. LMitzenmacher, and M. Najork. Measuring index quality using random walks on the web. In *Proceedings of the Eighth International World Wide Web Conference*, pages 213–225, May 1999. Available <http://www8.org/w8-papers/2c-search-discover/measuring/measuring.html>.

[27] A. Heydon and M. Najork. Mercator: A scalable, extensible Web crawler. *World Wide Web*, 2(4), Dec. 1999.

[28] J. M. Kleinberg. Hubs, authorities, and communities. *ACM Computing Surveys*, Dec. 1999. Available: <http://www.acm.org/pubs/articles/journals/surveys/1999-31-43es/a5-kleinberg/a5-kleinberg.pdf>.

[29] V. Kluev. Compiling document collections from the internet. *SIGIR Forum*, 34(2), Fall 2000. Available at <http://www.acm.org/sigir/forum/F2000/Kluev00.pdf>.

[30] S. Lawrence and C. L. Giles. Accessibility of information on the Web. *Nature*, 400(8), July 1999.

[31] R. Lempel and S. Moran. SALSA: the stochastic approach for link-structure analysis. *ACM Transations on Information Systems*, 19(2):131–160, Apr. 2001.

[32] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

[33] F. Menczer and R. K. Belew. *Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web*, pages 1–45. 1999.

[34] S. Mukherjea. Organizing topic-specific Web information. In *Proceedings of the Eleventh ACM Conference on Hypertext and Hypermedia*, pages 133–141, San Antonio, Tx, May 2000. Available: <http://www.acm.org/pubs/articles/proceedings/hypertext/336296/p133-mukherjea/p133-mukherjea.pdf>.

[35] S. Mukherjea. WTMS: A system for collecting and analyzing topic-specific Web information. In *Proceedings of the 9th International World Wide Web Conference: The Web: The Next Generation*, Amsterdam, May 2000. Elsevier. Available: <http://www9.org/w9cdrom/293/293.html> (current as of August 2001).

[36] P. Mutschke. Enhancing information retrieval in federated bibliographic data sources using author network based strategems. In *Proceedings of the 5th European Conference ECDL, Darmstadt, Germany*, pages 287–299, Sept. 2001.

[37] M. Najork and A. Heydon. High-performance Web crawling. Technical Report Research Report 173, Compaq SRC, Sept. 2001. Available at <http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-173.html>.

[38] M. Najork and J. Wiener. Breadth-first search

crawling yields high-quality pages. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong, May 2001. ACM. Available: <http://www10.org/cdrom/papers/208/>.

[39] G. Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill, New York, 1968.

[40] B. Saulnier. Portal power. *Cornell Engineering Magazine*, pages 16–21, Fall 2001. Available: <http://www.engineering.cornell.edu/engrMagazine/>.

[41] R. Stata, K. Bharat, and F. Maghoul. The term vector database: Fast access to indexing terms for Web pages. In *Proceedings of the 9th International World Wide Web Conference: The Web: The Next Generation*, Amsterdam, May 2000. Elsevier. Available: <http://www9.org/w9cdrom/159/159.html>.

[42] D. Voss. Better searching through science. *Science*, 293(5537):2024, 2001. Available: <http://www.sciencemag.org/cgi/content/full/293/5537/2024>.

[43] P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing & Management*, 24:577–597, 1988.

[44] I. H. Witten, D. Bainbridge, and S. J. Boddie. Power to the people: End-user building of digital library collections. In *Proceedings of the first ACM/IEEE-CS joint conference on Digital Libraries*, pages 94–103, 2001. Available: <http://www.acm.org/pubs/articles/proceedings/dl/379437/p94-witten/p94-witten.pdf>.

[45] I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge. Greenstone: A comprehensive open-source digital library software system. In *5th ACM Conference on Digital Libraries, San Antonio, Texas, June 2 - June 7, 2000, also titled ACM Proceedings of Digital Libraries, 2000 (DL2000), San Antonio, Texas*, pages 113–121, 2000. Available: <http://www.acm.org/pubs/articles/proceedings/dl/379437/p94-witten/p94-witten.pdf>.

[46] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *SIGIR 98*, Melbourne, Australia, 1998.

[47] L. L. Zia. The NSF national science, technology, engineering, and mathematics education digital library (NSDL) program: New projects and a project report. *D-Lib Magazine: The Magazine of Digital Library Research*, 7(11), Nov. 2001.