Low-Bandwidth Web Searching Using Categorization and Summarization

Frank Bentley and Georgi Peev
(bentley, georgepf)@mit.edu

December 12, 2002

Abstract

We describe a system that acts as a proxy between web search engines and low-bandwidth, feature-poor clients such as cell phones. We discuss the implementation design choices we made in creating multi-resolution descriptions of documents found in a web search query. Finally, we analyze the results from a user evaluation that showed a 38% reduction in bandwidth when using this approach over a text-based Google search.

1 Introduction

As cell phones start to offer web access with limited display size and very low bandwidth, it is helpful allow a low-bandwidth, but high information means to search and retrieve content from the Internet. We present a web proxy that offers the client a guided search at multiple levels in order to limit the bandwidth and time to find the desired page.

The guided search first returns a set of categories representing the different general areas of information that were found for a given query. The user can select a category and receive one sentence summaries of each document in the category. Then, selecting a document, they can retrieve a multiple sentence summary, and finally the entire document if they feel that they have found what they are looking for.

This system was guided by several principles:

1. The desire to send as little information as possible over the slow/expensive link.
2. The ability for the user to guide the search further (using categories) without having to search through many results or perform new searches.
3. To utilize the high bandwidth/fast processing of the proxy.

These principles guided us to create a proxy that guides a user through a search instead of just leaving them with a (potentially large) set of documents matching their query. Our system lets a user be more explorative without committing to loading an entire document and could be used in applications beyond handheld devices.

2 Related Work

Over the past few years, many groups have built systems for browsing the web on feature-poor or low-bandwidth devices. Concurrently, the areas of categorization and summarization have been hot topics in linguistic and AI circles.

2.1 Web Display on Handhelds

Throughout the past two years, several groups have worked on solutions for browsing the web, a media normally viewed on large, color displays at high speeds, on small, feature-poor devices such as Palm Pilots and Cell Phones.

WAP was one of the first endeavors that provided a way to get textual content to cellular phones and other WAP enabled devices. WAP proxies translate information from the web into a standardized format, similar, but different from the original HTML. There have been many studies on WAP and the type of traffic that WAP generates. Mainly WAP generates small packets (most under 250 bytes) over a small period of time (most under a minute or two). [13]

Although WAP uses less bandwidth solution than sending the full HTML documents (with table, image tags, javascripts, etc), it still does not provide a suitable solution for searching through large amounts of content on small displays. Buyukkokten et al have created a client for a Palm Pilot handheld called the Power Browser. [5] [4] The Power Browser allows a user to view a web page as a tree and expand and contract various portions of the page based on its structural content. For example, if there was a long list or a table, this could be collapsed into a single item that could be expanded if the user wanted to know more about it. [6] This allows for better use of the limited screen space, but since the entire tree is sent at once, this does not help with bandwidth concerns. The power browser facilitates searching by returning the HTML `<title>` of the top results of a keyword search. A user can click on a title to view the tree for that page.

The WEST system, developed by Bjork et al, is another browser designed for the Palm Pilot. [2] This card-based approach finds multiple-word summaries of important parts of a document and displays them on cards in a matrix on the screen. The user clicks on the card of interest to view the text on the page relating to the summary. Search was not supported.

Finally, Aridor et al use a categorization-based approach to help a search. [1] However, their categories are static and a search can be performed only within a given category. This is similar to browsing the tree of categories in Yahoo and was chosen so that most of the focusing can happen locally without sending request to the server (the tree was stored on the device).

Our system works to extend these works by providing a solution that uses search to produce categories that lead to short summarizations that finally lead to complete documents, specially formatted for display on a text-based browser.

2.2 Categorization

Categorization is a very well-developed topic in artificial intelligence. Many researchers have been working on algorithms for text categorization when the set of categories is predefined, as is in our system. Most of the research has been focused on learning (training) algorithms. For example, Lewis et al describe two such approaches, the Widrow-Hoff and EG algorithms, which can both be used for training linear text classifiers. [16] Two other possible examples of such algorithms include a Bayesian classifier and a decision tree learning algorithm, as described by Lewis and Ringuette. [15] Finally, Han presents a weight adjusted k-nearest neighbor classification algorithm in [10].

A variety of software systems for document classification is also available. Kreulen et al have implemented a software tool called eClassifier, which is based on their research in the field. [12] Another software package for classification is CLUTO, which was developed by George Karypis. [11] Apart from the two systems described above, there are also many commercial software packages. One specific commercial software package that is worth mentioning is the Vivisimo Clustering Engine $^{\text{TM}}$. [18] Although its description lead us to believe it would be perfect for our needs, unfortunately, Vivisimo does not currently provide free academic use of its product.

We chose to implement a generic algorithm of our own, which is loosely based on ideas found in [10]. The specific approach was chosen because of its simplicity and ease of implementation, since the development of complicated learning AI algorithms was beyond the scope of this project.

2.3 Summarization

Many researchers have been working on text summarization techniques. Some choose learnable approaches such as Markov Models. [7] Others choose statistical methods such as [19] or lexical models such as [8].

Other, simpler summarizing methods are practical when dealing with web content such as using a title or heading or simply the first sentence from each paragraph. Some of these techniques have been used in earlier systems such as those mentioned above. [5] [2]

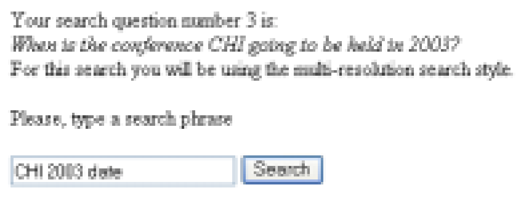
2.4 Low Connectivity Searching

The TEK project at MIT's Lab for Computer Science [14] is a project that works to provide search to communities where bandwidth is intermittent or expensive. TEK provides searching via email and allows users to submit queries and return hours or days later to view the results (which by then have been downloaded to a local server). While this is not practical for searching on interactive devices such as cell phones or handhelds, it uses some of the same principles that we use in our system: well connected hosts can act as search proxies and filter results for less well-connected users. TEK also shows that some less-well connected users are willing to sacrifice speed of searching for quality results.

3 Example Search

To provide an example for better illustration of our search proxy, consider a user searching for the date of the CHI conference in 2003 (this was one of the search questions in our user testing explained below).

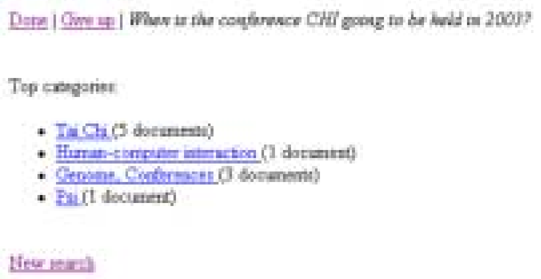
Figures 1 - 4 illustrate the search process from the initial query to the final answer. In Figure 1, the user enters the search words, in this case "CHI 2003 date." Note that in the figure, there is a prompt posing the question. This is part of the user evaluation. Although our proxy allows users to search for any


Your search question number 3 is:
When is the conference CHI going to be held in 2003?
For this search you will be using the multi-resolution search style.

Please, type a search phrase

CHI 2003 date

Figure 1: The main search page as shown from our user study.

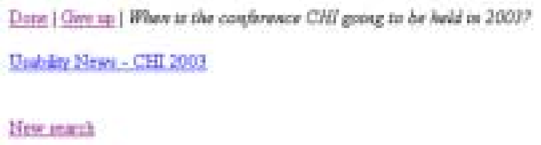


When is the conference CHI going to be held in 2003?

Top categories:

- Tai Chi (5 documents)
- Human-computer interaction (1 document)
- Genome, Conferences (3 documents)
- Fai (1 document)

Figure 2: The main category page for the search "CHI date 2003."



When is the conference CHI going to be held in 2003?

Usability News - CHI 2003

Figure 3: The short summary for the document in "Human-Computer Interaction"

When is the conference CHI going to be held in 2003?

The CHI 2003 Conference on Human Factors in Computing Systems will take place in Fort Lauderdale, Florida, April 5-10, 2003

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

UsabilityNews.

[illegible]

[illegible]

[illegible]

Figure 4: The document summary (10 sentences) for the document "Usability News."

query, we chose six focused questions to ask as part of our user evaluation.

Figure 2 illustrates the categories that are returned for this search. In this case the correct category, "Human-Computer Interaction" appears with one document. Because the search terms were relatively vague (no mention of conferences or computers or interfaces, etc.), note the other interesting categories such as "Tai Chi." The categories are intended to guide the user towards the correct documents, especially in cases when search terms were vague and cover many topic areas. This saves a user from searching through pages of flat results as is currently the standard way to search (using search engines like Google).

Clicking on "Human-Computer Interaction" brings up a document summary page. These summaries, as shown in Figure 3 are one sentence summaries of the document as described in Section 4.5.

Clicking on the short document summary will bring up the extended document summary, as seen in Figure 4 (and described in Section 4.6). In this case, the document contains our solution, "April 5-10, 2003" in the very first sentence. If the solution was not present in the summary, we could have gotten the full text content of the document or returned to the category page, or performed a new search.

4 Implementation

4.1 Design

The search proxy system has two components. The more important one is the server side program, implemented as a Java servlet, which performs the following actions:

1. Processes user queries
2. Performs the interaction with the Internet search engines to fetch the query results
3. Categorizes the results
4. Creates summaries of the documents in each category
5. Serves short result pages to the client.

A somewhat less important component is the optional thin client, which can be installed on the client side. Its purpose is to improve user interaction by utilizing the time, which the user spends on reading the results and deciding which ones to choose, to download from the server summaries that can then be viewed without having to wait additionally. This thin client can also act as and end to end module that allows for compression of documents over the network (i.e. zipping all data sent for added savings in bandwidth).

4.2 The server side Java servlet

This component implements all the vital functionality of the system. It performs the following tasks:

1. Interaction with the chosen Internet search engines.
2. Categorization of the documents in a query.
3. Summarization of a certain category of documents.
4. Summarization of a certain document.
5. Construction of short web pages to serve content to the client.

4.3 Interaction with Internet search engines

Since the system serves as a proxy between the client and Internet search engines, it was necessary to choose which specific engines to utilize, among the quite large number of such engines. We chose to interact with two of them - Google [9], and Vivisimo [17].

The first one, Google, is a well-known search engine that is very popular, which is probably due to the fact that it usually produces very good and relevant search results. Another reason why it was especially suitable for our needs is that Google has developed a SOAP API, which facilitates programmatic interaction with the search engine in a variety of environments, including Java. In our system, Google's API is used to communicate with the search engine and fetch the document links for the user query. Unfortunately, the Google API only allows us to retrieve 10 results per query. Extensions of this work could use an interface that allows more results or screen scrapes multiple pages of Google results.

The second search engine, Vivisimo, was chosen because it produces a categorization tree for the search results. A query is performed via constructing a simple query URL, since Vivisimo has not yet implemented an publicly available API. After the search results page has been fetched, and Vivisimo has built a result file on its servers, another URL is constructed in order to obtain the categorization tree for the search results. Thus, the URLs that Vivisimo returns as search results are not used themselves. Rather, only the top level of the categorization tree is parsed and then used to categorize the documents resulting from the Google search.

At a first glance it might seem somewhat cumbersome to have to perform two different queries for every user query. The reason why this approach was chosen is that independent implementation of a document categorization engine of adequate quality was beyond the scope of this project. It was, therefore, appropriate for us to implement the categorization scheme as described in the following section. We admit that a much better solution would be to implement a generic categorization algorithm that could operate on the Google results and never involve Vivisimo in the process.

4.4 Categorization of documents

The documents that Google returns as search results are divided into the categories produced by Vivisimo. In order to do that, first all ten documents that Google returns are fetched from the Google cache. This is necessary to carry out both the categorization and the summarization, and enables us to serve pages very quickly, after a small initial delay. The approach of getting the documents from the Google cache was chosen to reduce the variance in the server response time. Namely, we initially implemented fetching of the real documents from their respective URLs, but that proved to result in delays in the order of 30-60 seconds. We then decided that, although the data in the Google cache might not be the same as the actual data on the URL, it is being refreshed frequently enough to make this trade-off a worthwhile one. The server manages to fetch the ten documents from the Google cache in 20 to 30 seconds, which is approximately twice as fast, and with much less variance, making its behavior more predictable.

After the documents are fetched, they are stripped of most of the HTML tags inside to render them suitable as input for the categorization and summarization algorithms (see Section 4.7).

The categories obtained from Vivisimo are then processed using the WordNet database. More specifically, all distinct words within each category are parsed out and WordNet is then queried to find their synonyms, coordinate terms, and the first-level hypernyms. In this manner, a set of similar words is constructed for each word within a category. The union of these sets is then called the representative set of the category.

There were some technical problems while designing a system written as a Java applet that has to interface with the provided C interface to WordNet. In order to overcome that we decided to use WNJN, which is a package under a GPL GNU license, and is, thus, available for non-commercial use. [3] The WNJN package uses the Java Native Interface (JNI), instead of the older Native Method Interface (NMI) used by other similar packages, to link to a DLL that contains the C code implementing the functionality of WordNet.

Finally, all documents are ranked for each category by summing the reciprocals of the familiarity rankings (Word Net term for how common a particular word is) multiplied by the number of occurrences of all words within the given category inside the plain-text versions of the document (All HTML tags removed). In this manner, a word counts more towards a document's category score if it either occurs more times inside the document, or its familiarity is low. The document's category score is then weighed against the number of words in the representative set of the category, so as to avoid giving higher ranks for categories with larger representative sets. Consequently, a document is assigned the category for which it ranked the best. All documents are placed in exactly one category.

4.5 Summarization of the categories

A category summary page is created by obtaining either the HTML header for "description" (if it exists) or otherwise the first sentence of each document. The first sentence is defined by the string starting at the beginning of the file and going until there is a period or there are 100 characters, whichever comes first. This gives the user an opportunity to see what the page is most likely about. These short summaries are displayed as links to a per-document summary, separated by paragraph breaks, on the summary page. If a user is interested in a specific link, he can simply select the one of interest. However, if he is unhappy with the results, he can return to pick another category or perform another search.

4.6 Per-Document Summaries

Per-document summaries are currently created by taking the first sentence of each of the first ten paragraphs of a document. This technique has been used in early versions of Microsoft Office's summarizer and generally selects the most important, or, topic sentence, in well written text.

At the end of the short summary is a link to retrieve the entire content. By this point, the user should be rather sure that this page will contain all of the information he is looking for and can commit to receiving the entire page. Of course, the user can always go back to the category page, the category summary page, or perform a new search if the results are not pleasing.

The summary page is sent in HTML, with all inner tags except for the title, links, and line feeds removed to help conserve bandwidth. HTML tables and other simple structures (i.e. lists) are converted to just use linefeeds at to be supported on even the most primitive of devices and also to conserve some bandwidth.

4.7 Full Documents

The full document is fed to a user who requests it off of a summary page. The document is retrieved from the Google cache and all HTML tags are stripped except for the title and line feeds. Again, structures such as tables and lists are converted into paragraphs for easy reading on feature-poor devices. Images are replaced by their ALT tags, if any. In general, we find these documents to be quite readable even given their extreme lack of formatting (just paragraph splits).

We use this full document format for our final resolution in our multi-resolutioned search and also as the only resolution in the Google-style text search in our user study explained below. This was to see if we could improve bandwidth over even a very basic text-based traditional search using our multi-resolutioned approach.

5 Analysis

We performed a small user study over a period of two weeks to test the usefulness and characteristics of this method. There were a few key metrics we examined including total bytes served per query and total elapsed time per query. Participants were mainly college students with 3-8 years experience with the Internet.

Each participant was shown an instructions page which explained the concept of multi-resolutioned searching and the formalities of the user interface (how to start a new search, give up, etc.). Each participant was asked to find the answer to six questions.

The six questions asked were:

1. What year was MIT founded? [1]

2. When will the conference CHI be held in 2003? [2]

3. Who was the first author listed on the paper "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications" [3]

[1] 1861

[2] April 5-10, 2003

[3] Ion Stoica

Question Number	Percent Improvement in Band-Width Over Text-Based Google
1	+38.0
2	+64.4
3	-15.5
4	+61.0
5	-42.0
6	+66.6
Avg	+38.2

Table 1: Results of user study comparing multi-resolutioned search to a text-based Google search. Note that some questions ended up being asked more frequently in one search-style or another due to the random nature of the study. Thus, the average shown is over all queries not over all questions.

4. What was Abraham Linclon's middle name? [4]

5. What is the population of Chicago? [5]

6. What year did the Cubs last win the world series? [6]

Questions were chosen to require a number of different types of answers to test the robustness of our system to different types of searches. Some answers were numbers, years, dates, names, or simply no answer (Lincoln had no middle name).

Questions were chosen at random to be served with our multi-resolutioned proxy or with a text based Google interface (our baseline). The Google interface displayed the titles of the Google results and clicking on a title returned the full page (stripped of all HTML tags except simple formatting). Obviously this type of search is more bandwidth efficient than a current Google search. We chose this method to ensure that our multi-resolutioned search would be compared to a system that serves the same number of bytes for a given full document, thus showing that something was different about the multi-resolutioned style of searching.

Eighteen users participated in our study, performing 107 queries (some queries were not specific enough and had to be performed again with new search terms). Over these 107 queries, the multi-resulutioed search style used on average 38.2% less bandwidth than the Google-style search. On average, this represented a savings of 8694 bytes of data. We attribute this to several factors.

First, a user does not always have to download a full document in order to receive an answer. An answer was found without retrieving a single complete document 14% of the time using the multi-resolutioned proxy. This helped make the search much more efficient in bandwidth overall as results were found in the summaries.

[4]He didn't have one

[5]2,896,016

[6]1908

The second property of our search that likely led to this improved performance is the fact that a wrong choice when searching (clicking on a link in the search results) has a relatively low cost (in bytes) associated with it. Choosing a category sends just a few hundred bytes, and summaries average about 1000 bytes. Thus the user can explore the results more readily without reading through multiple full documents and this sending many bytes of data across the network.

Finally, by categorizing results, the user was often led to the correct document more easily than just searching through a series of documents in a flat list.

6 Future Work

This work can be extended in many ways. First, the categorization mechanisms can be improved to perform more rigorous document analysis and to create the categories from the actual Google results instead of relying on Vivisimo to produce categories. This would allow for more accurate testing of the categorization process.

Secondly, the summarizations can be produced by statistical means or through more complex lunguistic analysis. This would provide summaries that better capture the content of the document and do not rely on the authors to place important content at the start of each paragraph. This method would also work much better for longer documents, as currently we only use the first ten paragraphs in the summarization process.

It would also be interesting to see if adding more documents (beyond the ten that the Google API provides for free) would reduce the number of queries performed (due to the guided search of the categories over just exploring links in a standard list of search results).

More user testing, especially across ages, genders, and experience levels would be useful in verifying our results in a more general user set. Also asking users specific questions about the interaction experience would help in determining if this style of search is considered more usable from the user's perspective (a more qualitative assessment).

Another way to use this multi-resolution style and further improve results would be to make specialized clients and zip content between the proxy and the client. Assuming that on average for random text zip decreases data by 50%, zipped data along with our multi resoultioned approach would save approximately 69% of the bandwidth of a text-based Google search.

Finally, if one were more interested in speed than bandwidth (pricing models for cell phones with unlimited data for a fixed cost per month), a client could be built to accept category and summary data while a user was browsing to hopefully predict the page they would select next (for example, loading the summary data for all documents in the currently active category). This would allow for a quicker user experience even over a small bandwidth link if the caching was able to work well. Since a majority of a web surfer's time is spent reading (with the link idle), it seems reasonable that this approach could work reasonably well in cases where bandwidth was not the main issue.

7 Conclusion

We have presented a multi-resolutioned search proxy and have demonstrated that it can achieve a significant improvement in bandwidth (38%) over a text-based traditional web search. We have listed several properties of this search that have contributed to its performance and have described our implementation of this algorithm.

We feel that this method of searching is useful for many applications beyond that of browsing from low bandwidth devices. The categorization helps a user find results in a more structured manner and thus could be useful for any task that involves searching through a large number of documents/files. The levels of summary help a user to determine if a document might contain the information he is searching for without scanning an entire document by hand. These features are applicable to web searches from desktops as well as from handheld devices.

The use of an multi-resolutioned algorithm like ours would help reduce overall bandwidth related to searching from the proxy to the user. It would be ideal for search providers such as Google, to implement this functionality on their own servers where the data is already cached to provide fast performance and limit data sent out to all of its users as well as provide a more focused search experience.

References

[1] Yariv Aridor, David Carmel, Yoelle S. Maarek, Aya Soffer, and Ronny Lempel. Knowledge encapsulation for focused search from pervasive devices. *ACM Transactions on Information Systems (TOIS)*, 20(1):25–46, 2002.

[2] S. Bj, R. Holmquist, J. Redstr, M. Bretan, R. Danielsson, J. Karlgren, and K. Franzen. West: A web browser for small terminals, 1999.

[3] Bernard Bou. Sourceforge project: Wordnet java native interface.

[4] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Text summarization of web pages on handheld devices.

[5] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Focused Web searching with PDAs. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):213–230, 2000.

[6] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Seeing the whole in parts: Text summarization for web browsing on handheld devices. In *Proceedings of the Tenth International World-Wide Web Conference*, 2001.

[7] John M. Conroy and Dianne P. O'Leary. Text summarization via hidden markov models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 406–407. ACM Press, 2001.

[8] Jade Goldstein, Mark Kantrowitz, Vibhu O. Mittal, and Jaime G. Carbonell. Summarizing text documents: Sentence selection and evaluation metrics. In *Research and Development in Information Retrieval*, pages 121–128, 1999.

[9] Google, http://www.google.com/.

[10] Eui-Hong Han. Text categorization using weight adjusted k-nearest neighbor classification. In *PhD thesis, University of Minnesota*, October 1999.

[11] George Karypis. Cluto: Software for clustering high-dimensional datasets.

[12] Jeff Kreulen, Dharmendra Modha, Scott Spangler, and Ray Strong. An interactive approach to document classification.

[13] Thomas Kunz, Thomas Barry, James P. Black, and Hugh M. Mahoney. Wap traffic: description and comparison to www traffic. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 11–19. ACM Press, 2000.

[14] Libby Levison, William Thies, and Saman Amarasinghe. Providing web search capability for low-connectivity communities. In *Proceedings of*

the 2002 International Symposium on Technology and Society (ISTAS'02): Social Implications of Information and Communication Technology, June 2002.

[15] D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.

[16] D. Lewis, R. Schapire, J. Callan, and R. Papka. Training algorithms for linear text classifiers. In *Proceedings of ACM SIGIR*, pages 81–93, 1996.

[17] Vivisimo, http://www.vivisimo.com/.

[18] Vivisimo Clustering Engine $^{\text{TM}}$, http://vivisimo.com/products/Clustering _Engine/Introduction.html.

[19] Hongyuan Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 113–120. ACM Press, 2002.