Search Tool Implementation for Historical Archive

Mike Scarborough
mikescar@vt.edu

Dr. Edward A. Fox
Department of Computer Science
fox@vt.edu

Dr. Linda Arnold
Department of History
redtape@vt.edu

Virginia Polytechnic Institute & State University

Abstract

Dr. Linda Arnold's archival project "Mexican-American War and the Media" is an underutilized resource. Providing contrasting primary sources on the War, it is the only archive of its kind. In order to make the archive's massive amount of information more accessible to researchers and students, I added search functionality to the site. Several tools were implemented and tested. Perlfect, a Perl-based open-source approach, was determined to be the best option. This report includes an outline of the steps taken to implement the search tool, a user's manual, a developer's manual, and options for future work. The archive may be accessed at www.majbill.vt.edu/history/mxamwar/index.htm.

Table of Contents

Table of Figures and Tables

1. Background

This independent study was conceived in order to provide a search capability for Dr. Linda Arnold's Mexican-American War and the Media archive project [1]. The goal of the archive is to serve as a resource for teaching and research. It is the only archive of its kind covering the Mexican-American War.

Dr. Arnold originally designed the archive to support browsing through large HTML files, which contain transcribed newspaper articles from four newspapers (*Martinsburg Gazette*, *Richmond Whig*, *Times of London,* and *Niles National Register*), for the years 1844-1848. The archive is broken into four sections, one for each newspaper. Users may browse each newspaper by time period. For example, *The Times* has a subsection for January-December 1845, January-July 1846, August-December 1846, etc. For each subsection, a large HTML file (about 513 KB) contains a list of the titles for each article from that period, followed by the articles themselves. Some titles in the list contain a link to the corresponding newspaper article; some do not. In some cases, the HTML file is divided into a text list of the articles contained below, and then the articles themselves, but without hyperlinks to connect the content.

Previously, if users wanted to search for a term in the archive, they would have to go to each of the 34 subsections and use their browser's Ctrl-F “Find” function to locate that term within that page. Alternatively, they could look throughout an index, hoping to find relevant terms from the article titles, and follow the link to that article (if available). This project was meant to provide improved access to the huge amount of information in the archive and to make improvements to address a critique by Matt Karush of George Mason University [2]. Karush acknowledges the extensive resources the archive provides, but laments the skeletal nature of the indices and warns teachers not to send their students to the site unprepared.

This independent study was meant to achieve two goals: 1) to add search functionality to the archive and 2) to provide hands-on experience in web programming and in implementing a search tool.

2. Implementation

Before providing search functionality to this archive, I had a good deal of learning to do. This was the first time I had ever used anything other than FTP and HTML for making content available on the web. After reading *Understanding Search Engines* by Michael Berry and Murray Browne [3], I had a better understanding of how the back end works.

My plan for the semester was based upon the procedures followed by the University of Pennsylvania when they added search functionality to their websites [4]. Circumstances differed though, as U-Penn had six people working on the project and a budget to pay for a commercially provided solution; however, the procedures worked well for my situation. My plan consisted of:

1. Create a schedule.
2. Conduct a preliminary screening.
3. Create a technical requirements document.
4. Evaluate the options and select final candidate solutions.
5. Install test versions of each solution.
6. Test each solution. Get Dr. Arnold's input and preferences.
7. Perform local customizations.
8. Install the final product in a permanent location.
9. Create users' manual and developers' manual.
10. Present results.

2.1 Schedule

My schedule spaced the work out over the course of the semester, aiming to finish two weeks before the end of the semester. The implementation and testing phase took more time than expected. Initially I planned to have a simple usability study to test each possible solution, but due to time constraints this was not possible.

2.2 Preliminary Screening

I did not know much about how to actually implement the search tool before I did the preliminary screening. I knew that the solution needed to be free to implement and operate, which meant either starting from scratch or using an open-source approach. I conducted the preliminary screening on the Internet. Most comparative discussions concerning search engines focus on the differences between the major commercial services (Google, Lycos, Yahoo!, etc). However, numerous search tool listings are available on the Web, with one of the most comprehensive available provided by Search Tools Consulting [5].

During the preliminary screening, I identified eight possible open-source solutions: ASPSeek, MnoGoSearch, Glimpse/WebGlimpse, Ksearch, Lucene, ht:/dig, Perlfect, SwishE, and Zebra [6-14]. I chose these tools based on their suitability to my most basic criteria: open-source, free of charge, and the ability to run on a Linux/UNIX server.

2.3 Technical Requirements Document

The technical requirements for candidate solutions centered on cost, platform, indexing method, and ease of maintenance. The Technical Requirements Document is Appendix A of this report.

2.4 Evaluate the Options

In order to evaluate each option available I used a spreadsheet to compare pros and cons (see Table 1). As a result, I selected five final candidates that seemed best suited for the task. All were free, with mailing lists and web boards with varying degrees of activity. The final candidates and primary reasons for their selection:

1. Glimpse: web-based administration; Spanish support
2. KSearch: primarily for local server searches; configurable stopwords.
3. MnoGoSearch: good reputation; fast search; SQL backend; UNICODE support.
4. Perlfect: Boolean searches; completely Perl-based; fast.
5. SwishE: simple to maintain; administration interface available.

I thought it would be best to implement as many test tools as possible, in case any did not live up to their billing. Sometimes the advertised features for a product don't work the way they should. I did not want to be stuck with one or two implementations that did not provide the expected functionality. This was an adaptation of the LOCKSS ("Lots of Copies Keeps Stuff Safe") philosophy—in this case, lots of implementations ensured success.

Requirements Analysis Table

Criterion	ASPseek	MnoGoSearch	Glimpse	Ksearch	Lucene	ht:/dig	Perlfect	SwishE	Zebra
				User Functionality					
Optimized local indexing?	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Crawling capability?	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No
Indexes PDF files?	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No
Spanish language support?	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
				Cost of Ownership & Operation					
Free to implement, run?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Runs on available hardware?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Runs on Linux/UNIX servers?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
				Administration & Maintenance					
Active development communit	Yes	Yes	Yes	Somewhat	Yes	Yes	Yes	Yes	Somewhat
Customizable search interface	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Customizable results page?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Admin simple for non-techies?	No	No	Yes	Maybe	No	Maybe	Yes	Yes	Maybe

Table 1: Feature Comparison

2.5 Install Test Versions

This was the most time-consuming and important step. I also learned the most during this phase of the project. I started by getting rid of Windows on my home machine, and installing RedHat Linux 8. I've never used Linux much, but I thought it would force me to learn it faster this way; I also would be able to develop on an operating system more similar to what most servers run. While it was worthwhile to develop in Linux on the home machine, most of the bugs and problems were introduced once I had space on a server and began developing the search tools there. Originally the scripts executed from Ming Luo's uther.dlib.vt.edu server; currently they use Yuxin Chen's tuppence.dlib.vt.edu server.

2.6 Test Each Solution

Extensive pilot testing identified a few bugs, which I fixed.

Dr. Arnold also identified a problem with the SwishE interface when she evaluated each solution. Most of the debugging that I did was in the previous phase, however, and so testing went well.

Out of the five test implementations, Ksearch, SwishE and Perlfect turned out the best, and I kept them as the options to show Dr. Arnold. The MnoGoSearch implementation ran into some database problems. MnoGoSearch utilizes a mySQL database for its indexes. Initially I thought MnoGoSearch would be the best overall, but this was wrong on two counts. While it uses UNICODE to represent characters, and while it would allow for Spanish texts to be added easily in the future, most other implementations also recognize the Spanish-language characters that are absent in English anyhow. Furthermore, due to the relatively small size of the archive, an SQL backend did not make for a faster search. So I decided to scrap the MnoGoSearch implementation, and focus on the others.

For each engine, I constructed various searches that would, if working correctly, return specific documents with high relevancy. Each engine did quite well, so it came down to a matter of interface preferences and indexing/search speeds. The Ksearch scripts took entirely too long to index the archive—around 40 minutes. It was also too slow for searches including common terms (“mexico”, “squadron”, etc). Perlfect and SwishE both indexed quite quickly, and returned search results equally well.

I chose Perlfect as the final solution for two reasons. First, it is purely Perl-based—making code alterations and debugging much simpler than that of SwishE, which includes C, C++, and Perl code. Perlfect scripts have well-named variables and good documentation. Second, SwishE’s web-based administrative GUI has some usability problems—including the sequencing of tasks and poor feedback—making it a poor choice for novice, infrequent users. Setting up Perlfect to periodically index using the UNIX crontab program is simpler and more reliable. Any other changes that the administrator may want to make are discussed in Section 4.

2.7 Perform Local Customizations

The size of the HTML files presented the main problem to adding search functionality to the site. Users could get results with pages containing their search terms, but would have to rely on the Ctrl-F browser function to find those terms on the page. Originally, I planned to have the search engine recognize the internal anchors at the start of each article, but this did not work. Dr. Arnold and I devised a new organization scheme to deal with the large-file problem.

Each newspaper now has a file for each date of publication. For example, the file `Times1848Jan30.htm` contains all the articles from *The Times* edition published on January 30, 1848. Some of the files contain one article; others contain multiple articles. These smaller files are stored on tuppence.dlib.vt.edu. They were created manually from the large HTML files using Microsoft Frontpage. In the future, this extra work will not have to be done. Dr. Arnold receives the files individually from students and creates the large files herself; therefore, creating smaller files organized by date of publication will not present much of a problem.

2.8 Install the Final Product

Unfortunately, this phase did not go as planned. Originally, the plan was to store the final tool on majbill.vt.edu, the History Department's server. The administrator for the server, Sanjiv Parikh, was wary of allowing scripts to be installed that he did not create. He had problems in the past with allowing individuals to execute scripts, and did not want to revisit the situation, so he would not allow me to install the software there, until the department moves to a university server. For now, the search tool executables and the small files will remain at tuppence.dlib.vt.edu, and the large files will be on majbill.vt.edu.

The search tool can be accessed at www.majbill.vt.edu/history/mxamwar/index.htm.

2.9 Create User Manual and Developer's Manual

The user's manual is found in Section 3 of this report. The developer's manual is in Section 4.

2.10 Present Results

To present my results, I met with Bruce Pencek, the College Librarian for Social Sciences. He had worked previously with Dr. Arnold on the archive, and was interested to see the new functionality and organization. He felt that the increased accessibility of the archive’s information would lead to increased usage and exposure. We discussed at length possible new ways to store and display the information in the future (see Section 6).

3 User's Manual

This manual is divided into two sections: searching and administration.

3.1 Searching

The search tool may be accessed at www.majbill.vt.edu/history/mxamwar/index.htm.

In order to execute a search using Perlfect, type a query into the search box. Choose whether you want to find documents that contain all of your search terms, or that contain any of them. The default option is “ALL”. With “ALL”, only documents that contain every term in the query will be returned. With “ANY”, the search engine will return documents that contain at least one of the terms, but not necessarily any of the other terms.

You also can enter phrases that you would like to find. To do this, put quotes around your phrase (e.g., “President Tyler”, “Baltimore Convention Oregon question”, “June 1846”). The search engine will return documents that contain that exact phrase.

To easily find the search terms within the documents that contain them, follow the “highlight matches” link that corresponds to a search result. By following that link, Perlfect will place a color background behind each occurrence of the search term, making it stand out from the rest of the text.

3.2 Administration

Administration of Perlfect is fairly simple. Greater detail concerning how Perlfect works can be found in the Developer's Manual (see Section 5), but the administrator only needs to perform a few simple tasks to keep the search tool up and running.

As previously mentioned, all files are stored at tuppence.dlib.vt.edu. To log in, use an SSH client to connect to tuppence.dlib.vt.edu with the username “mexamwar” and the correct password.

All files mentioned below can be found in the tuppence directory /home/mexamwar/public_html/cgi-bin/perlfect/search. This is the directory that contains all the files that Perlfect needs to run.

3.2.1 Changing the Perlfect Configuration

Perlfect determines which options to use by looking at the conf.pl file. This file is actually a Perl script, but it can be edited the same as any text file. The file is basically a list of variables that can be changed in order to alter Perlfect's behavior. Table 2 lists variables that could be changed by the administrator, and their approximate location in the file.

Variable Name	Line Number	What to Do
DOCUMENT_ROOT	12	If you change the location of stored files, update this variable to reflect new location.
BASE_URL	15	Corresponds to DOCUMENT_ROOT. If you change the location of stored files, update this variable to reflect the correct URL that points to the files.
INDEXER_CGI_PASSWORD	34	This sets the password used by the indexer_web script which allows the admin to execute the indexer over the web. To disable this feature, leave the value empty.
RESULTS_PER_PAGE	85	Sets the number of results per page that are returned to the user.
HIGHLIGHT_MATCHES	91	Enables highlighting of query terms in the search results. Change value to zero (0) to disable.
INDEX_NUMBERS	102	Enables indexing of numbers, which allows users to search for dates. Change value to zero (0) to disable.

Table 2: Perlfect variables the administrator may want to change

3.2.2 Adding or Removing Stopwords

Perlfect uses a list of “stopwords” to prevent common words (e.g., “the”, “at,” “those”) from being indexed. This list is a text file, found at conf/stopwords.txt. The file contains one word per line, and words can be added or removed from the list.

3.2.3 Reindexing the Document Set

Whenever you add or remove documents from the directory that you want to be able to search, you must run the indexer. The indexer creates a database, which keeps track of what words and phrases are contained in each file. When files are deleted, the indexer will still point to missing files. When files are added, the indexer won't know they exist, unless it is run again. Running the indexer whenever you make changes ensures that users will be searching the correct set of documents.

There are two ways to run the indexer. It may be run over the web, or directly by using an SSH client.

3.2.3a Over the Web

To run the indexer over the web, simply enter the URL tuppence.dlib.vt.edu/~mexamwar/cgi-bin/perlfect/search/indexer.pl?password= into your web browser location bar, and complete the URL by adding the indexer password (INDEXER_CGI_PASSWORD, set in conf.pl as noted in Section 3.2.1). For example, if the password was "greatday", you would enter the URL tuppence.dlib.vt.edu/~mexamwar/cgi-bin/perlfect/search/indexer.pl?password=greatday. This will tell the indexer to index all the documents in the directory specified in conf.pl.

The script will display the indexing progress while you wait. First, a list of every file being indexed will be generated. Next, a progress indicator will track the script while it is writing the final database files. You should not exit your browser, go to a different page, or click the stop button until indexing is completely finished. Instead, wait until the message "Indexer finished" is printed at the bottom of the browser window.

3.2.3b Direct Execution

To run the indexer directly, SSH or Telnet into tuppence.dlib.vt.edu. Change into the Perlfect installation directory (`public_html/cgi-bin/perlfect/search/`). Tell the indexer to run by typing "`./indexer.pl`".

You can make the indexer run reliably at certain intervals by adding `indexer.pl` to your `crontab` file. `Crontab` is a UNIX program that allows users to specify intervals in between automatic execution of programs. For more information on `crontab`, see [15]. On tuppence.dlib.vt.edu, the indexer is set to run every Friday at 4 AM.

3.2.4 Adding Files to the Archive

To add files to the archive, use an SSH file transfer client to login tuppence.dlib.vt.edu. Your username is mexamwar, and enter the correct password. Change into the `public_html` directory. That is where the article files are stored, and this is the directory that is crawled by the indexer. There are four subdirectories, one for each newspaper. Change into a subdirectory to add more articles from a particular paper, or make a new directory to store articles from a new newspaper. If a new directory is added within the `public_html` directory, there is no need to change the Perlfect configuration. The scripts will index the new directory automatically the next time the indexer runs. When finished uploading, terminate the connection. Run the indexer again using either method outlined above, so that the new documents may be searched.

4. Developer's Manual

Perlfect is a completely Perl-based search tool implementation, available at http://www.perlfect.com. It is an open source product under the GNU General Public License. To generate results rankings, Perlfect uses a document vector model. Perlfect requires a Perl interpreter (5.004 or later) and the DB_File Perl module (1.72 or later). It can run on Linux, UNIX, and Windows servers. This implementation runs under Linux on tuppence.dlib.vt.edu.

Perlfect consists of several Perl scripts, HTML template files, and database files. Figure 1 below provides a basic overview of component interaction. For more details than are provided in this report, please see the Perlfect developer's page [19].

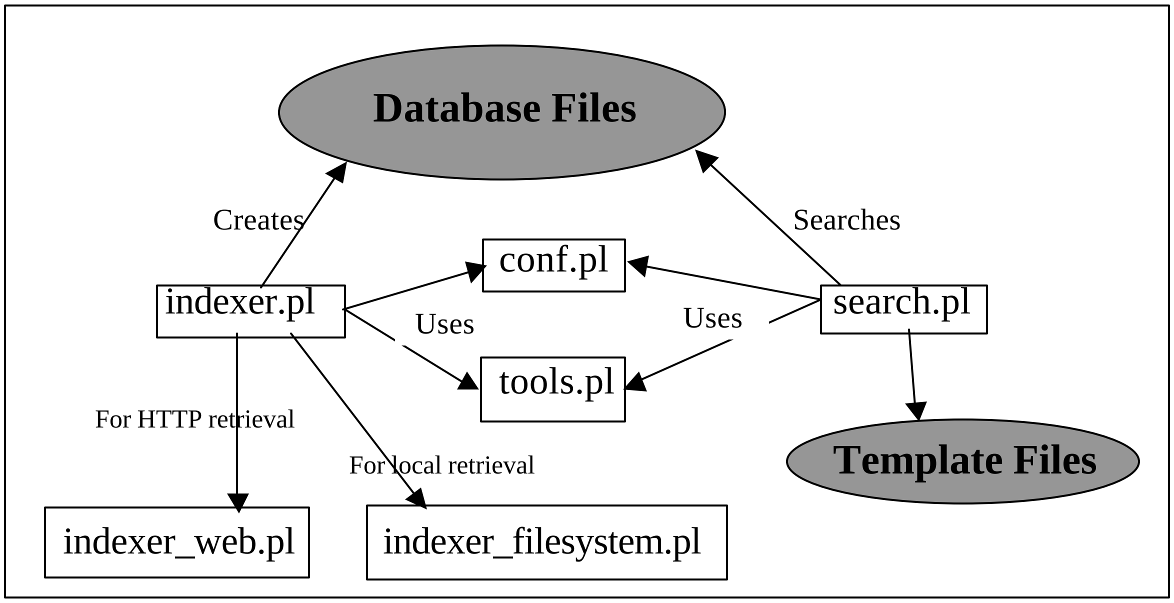

Figure 1: Perlfect architecture

4.1 Indexer.pl

The `indexer.pl` script contains most of the code that powers the indexer. The indexer uses variables defined within `conf.pl` to determine what directories to index as well as what indexing options should be enabled (special characters to index, file extensions to index, etc). The indexer relies on three other scripts to provide necessary modules: `indexer_web.pl`, `indexer_filesystem.pl`, and `tools.pl`.

During the indexing process, `indexer.pl` writes to temporary database files in the `data` directory of the Perlfect installation. When the indexing is complete, the files lose their _tmp suffix, and are ready for access by search.pl.

To eliminate worries about whether the indexer is up to date, you may insert `indexer.pl` into your `crontab` file on your server, allowing you to specify an interval between automatic executions of the indexer script. On tuppence.dlib.vt.edu, the indexer is setup to run every Friday at 4AM.

4.2 Indexer_web.pl

`Indexer_web.pl` allows the indexer to gather files using HTTP. This should only be used to index files on the local server that are dynamically created, such as PHP files.

4.3 Indexer_filesystem.pl

`Indexer_filesystem.pl` allows the indexer to crawl the local filesystem to index files.

4.4 Tools.pl

`Tools.pl` mainly provides functions for string manipulation error checking, and error avoidance. Also included in this script are functions for parsing files, dealing with special characters, and building lists. It is used by both `indexer.pl` and `search.pl`.

4.5 Search.pl

`Search.pl` takes a search query, checks the database files, and returns results to the user's web browser. It makes use of template files found in the `template/` directory to generate the results. There should be no need to modify `search.pl`, as options it uses are specified in `conf.pl`. Furthermore, the results pages can be customized by modifying the template files.

Results rankings are calculated by search.pl using the following formula:
score = word occurrences in document * log (# of documents / # of documents containing this word)

4.6 Conf.pl

`Conf.pl` is the backbone of Perlfect. Options regarding all aspects of Perlfect operation are selected in this configuration file. Both `search.pl` and `indexer.pl` require `conf.pl` to operate. Each variable is well-explained within the file. Changing some variables may require the indexer to be run before the changes will take effect. For variables fitting this special case, the tag `[re-index]` is at the end of the variable's description. If you change any of these variables, be sure to run the indexer afterwards.

4.7 Database Files

The database files used by Perlfect are created and accessed using Berkeley DB, a type of lightweight database that requires the Perl module DB_File version 1.72 or later. The database files are stored in the `data/` directory. The files consist of multiple tables that hold key/value pairs. Each indexed document has a document identification number. There is one table each for the following attributes: the URL, title, and description for each document indexed. In each of these tables, the document identification number allows the search script to identify documents relevant to the search query and to construct the search results page.

4.8 Template Files

The template files used by search.pl to generate the search results pages can be found in the `templates/` directory. These files include templates for not returning any matches (`no_match.html`), for successful search queries (`search.html`), as well as templates for Italian, German, and French. Perlfect has been configured to automatically return the results in the language of the user's browser. The templates may be altered to change the look and feel of the results pages.

5. Lessons Learned

I learned a great deal this semester about programming for the Web. While I am still far from being an expert or even intermediate web programmer, my experience was valuable as an introduction to the area. It is a different world than making C++ programs for undergraduate classes. I learned a lot about working in Linux environments, and now I prefer it to programming in Windows. Numerous books were a great help in this project, most notably [15-18].

I now have good, but basic, familiarity with Perl, and this project motivated me to pursue this. Throughout the course of the semester, debugging and modifying .cgi and .pl scripts was a good way to learn. I now have a good understanding about how scripting and CGI programming works, about which I knew nothing before. I am now quite familiar with Apache error logs, `.htaccess` files, setting permissions, and debugging Internal Server Errors. I worked with mySQL for the first time in order to get MnoGoSearch working correctly.

6. Future Work

This project made me interested in web programming, and has resulted in another practical application of digital libraries. I am interested in continuing my work on this project, and taking the archive to the next level. Working with Dr. Arnold to modify the site structure and organization will do much towards this goal. Currently, even with the search functionality, the system is still somewhat rudimentary.

Now that the site can be searched, I would like to improve the efficacy of the search engine. The many smaller files now in use, while making it easier for the user to find where the desired keywords occur within the text, are not ideal. I have discussed converting all the large files to PDF with both Dr. Arnold and Bruce Pencek. That way, users could browse and search on the same files. It may be possible to index the PDF files by page, and return links to certain pages in the PDF as results.

Initial research into converting the large files into PDF indicates that this solution seems promising. Surprisingly, the HTML files converted to PDF generally shed 100-150 KB, depending on size. The PDF could have a new page for each article, and a new script could be written to index the PDF by page. This would maintain the large-file-browsing structure currently employed by the archive, without requiring separate storage of smaller, one-article-per-page HTML files. One downside to this approach would be the increased load times required for PDF viewers (as noted by Mr. Pencek, this could prove particularly problematic with the "bloated" Acrobat 6). Dr. Arnold wants to retain the ability to browse through large files; she says such an approach to browsing is a common method of viewing information for historians. After further analysis and testing, she can confidently decide whether the PDF option would be an improvement.

More immediately, the new, smaller files could use better titles. Due to the inconsistent nature of the labels given each article—as well as having no reason to label each article with the newspaper name (within a huge file containing articles all from the same source)--the search results returned to the user don't indicate very well which newspaper they come from. The newspaper name can be seen in the URL, but adding more meaningful information to the document titles and/or HTML body, could more quickly signal to users what paper they are reading. Also, adding HTML tags to give the smaller files the same colors and feel of the site at large would improve the visual continuity of the site. These improvements could be accomplished with Perl implementation, and I am writing a Perl script to solve this usability problem.

Dr. Arnold plans to add more transcriptions in the future, and perhaps more newspapers. It is possible that Spanish-language texts from other sources will be added as well. The search implementation described herein is ready and able to accommodate any such future changes.

7. Acknowledgements

I would like to thank Ming Luo and Yuxin Chen for providing me with space on their servers, and for allowing me to develop my scripts. Also, I thank Dr. Fox for marshalling resources for me when I needed them. Bruce Pencek, Virginia Tech's Librarian for Social Sciences, was enthusiastic about the project and offered good advice. Finally, it was a pleasure working with Dr. Arnold again.

8. References

[1] Arnold, Linda. "The Mexican-American War and the Media". 2004.
http://www.majbill.vt.edu/history/mxamwar/index.htm

[2] Karush, Matt. "The Mexican-American War and the Media (Review)". March 2003.
http://chnm.gmu.edu/whm/d/91.html

[3] Berry, Michael W. and Murray Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval.* Philadelphia: Society for Industrial and Applied Mathematics, 1999.

[4] Search Tools Consulting. "Choosing a Site Search Tool". 2001.

http://www.searchtools.com/guide/index.html#basic

[5] Search Tools Consulting. "Alphabetical List of SearchTools Product Reports". 2001. http://www.searchtools.com/tools/tools.html

[6] SWSoft. ASPseek search engine software. 2003. http://www.aspseek.org

[7] LavTech Com Corp. MnoGoSearch Search Engine. 2003. http://search.mnogo.ru

[8] Internet Workshop. WebGlimpse Search Engine. 2002. http://www.webglimpse.net

[9] Kscripts.com. Ksearch. 2000. http://www.kscripts.com/scripts.shtml

[10] Apache Software Foundation. Jakarta Lucene. 2004. http://jakarta.apache.org/lucene/docs/index.html

[11] The ht://Dig Group. ht://Dig: Search Software. 2004. http://www.htdig.org

[12] Perlfect Solutions Ltd. Perlfect Search 3.31. 2004. http://www.perlfect.com/freescripts/search

[13] Swish-E Development Team. SWISH-Enhanced. 2004. http://www.swish-e.org

[14] Index Data Aps. Zebra. 2003. http://www.indexdata.dk/zebra

[15] Gilly, Daniel. *Unix in a Nutshell*. Cambridge: O'Reilly and Associates, 1992.

[16] Gundavaram, Shishir. *CGI Programming on the World Wide Web*. Cambridge: O'Reilly & Associates, 1996.

[17] Schwartz, Randal L. and Tom Christiansen. *Learning Perl*. Cambridge: O'Reilly & Associates, 1997.

[18] Asbury, Stephen, et al. *CGI How-To: The Definitive CGI Scripting Problem-Solver*. Corte Modero, CA: Waite Group Press, 1996.

[19] Perlfect Solutions Ltd. "Perlfect Search – Development". http://perlfect.com/freescripts/search/development.shtml

[20] Giorgos. "Search Algorithm Explanation". August 11, 2000. http://www.perlmonks.org/index.pl?node_id=27509

APPENDIX A

Technical Requirements for Search Tool

User Functionality

-- Need to index a local server—search restricted to archive files only. However, crawling ability is a plus, the future structure of the archive is not known at this point.

-- Easy to use search interface. Search results page also must be clearly presented and easily understood. Users should not have to use complicated regular expressions in order to execute a search.

-- Needs to index .htm and PDF files. Support for other formats (to plan for future changes to the archive) essential.

-- Needs to index and provide software support for English and Spanish (archive may include Spanish language materials in the future).

Cost of Ownership/Operation

-- Must be free software, both to implement and run for an indefinite period.

-- Must not require any new hardware or hardware upgrades.

Administration / Maintenance

-- An active user and/or development community.

-- Customizable search interface and results page.

-- Must be easy for a non-technical person to administer changes to the software.

-- Must allow for changes in archive content and structure, and be as flexible as possible to allow for any future changes.