

Selective Searching over the Networked Computer Science Technical Report Library

A Thesis
in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in Computer Science

by

Otis Benjamin Young III

11/20/97

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in Humanities Courses.

Otis Benjamin Young III (Full signature)

Approved _____ (Technical Advisor)
James C. French

Approved _____ (TCC Advisor)
Michael E. Gorman

TECHNICAL ADVISOR'S EVALUATION

Date: _____

To: _____, Technical Advisor

From: _____, Student

Subject: Technical Advisor's Evaluation of my Thesis

Return Completed Evaluation to: _____, TCC Advisor

The role of the technical advisor on the undergraduate thesis is a very important one. To help judge fairly the overall quality of the undergraduate thesis, please comment on the technical aspect of my work and the working relationship I established with you. Please rate the technical quality of the completed work according to the following check list, using a scale from one to five.

5=excellent; 4=good; 3=average; 2=poor, but still acceptable; 1=unacceptable.

Please feel free to add any notes below or on the back of this sheet.

1 2 3 4 5 I. Did I maintain an appropriate working relationship with you throughout the project?

1 2 3 4 5 II. Based on the following criteria, how do you rate the overall technical quality of this thesis?

III. Detailed Evaluation:

1 2 3 4 5 a. Are the subject and scope of the project clearly stated?

1 2 3 4 5 b. Is the conclusion consistent with the statement of subject and scope?

1 2 3 4 5 c. Is the conclusion adequately supported by data, theory, or calculations in the body of the paper?

1 2 3 4 5 d. Is the level of work reported consistent with what you agreed would be a reasonable project?

1 2 3 4 5 e. Does the paper show unusual merit as a result of my contribution, such as imaginative conception, innovative approach, or insights into the subject?

1 2 3 4 5 f. Does the paper show skills in handling engineering or other technical concepts at a level appropriate to an undergraduate degree candidate?

1 2 3 4 5 g. Does the bibliography show a sound background in the literature appropriate to the subject?

yes no h. Was this thesis part of a project course in the major field?

yes no i. Would you like to discuss my work with my TCC Advisor?

Approved _____, Technical Advisor Date _____

TCC ADVISOR'S EVALUATION

Author's Name _____

TCC Advisor _____

Technical Advisor _____

5 = Excellent; 4 = Good; 3 = Average; 2 = Poor but still acceptable; 1 = Unacceptable

5 4 3 2 1 Overall Design: This project is imaginatively conceived, thoroughly researched, convincingly expressed, and/or potentially significant.

5 4 3 2 1 Introduction: Establishes meaningful context and significance of project for multiple audiences; grounds research thoroughly in literature; clearly defines problem and scope; explains key concepts; previews structure of rest of report.

5 4 3 2 1 Methods: Adequately justifies and clearly describes project design.

5 4 3 2 1 Results: Presents data with verbal and visual clarity; emphasizes central research finding; fully explains and illustrates design and use of final product or conclusion.

5 4 3 2 1 Interpretation: Conclusions follow logically from data and/or argument; reaches synthesis which is related to context established in introduction; assesses ethical and social impact; recognizes limitations of this research and makes recommendations for future work.

5 4 3 2 1 Organization: Logically divided into unified and coherent chapters, sections, and paragraphs; transitions link each chapter to the whole and establish flow between sentences and paragraphs.

5 4 3 2 1 Graphics: Includes enough appropriate and clearly understandable figures and tables which are well integrated with text, adequately interpreted, clearly labelled and captioned, large enough, and printed darkly.

5 4 3 2 1 Documentation: Adequately supports argument with references cited in the text, using correct and consistent style; bibliography shows thorough depth and breadth and is correct in form.

5 4 3 2 1 Style: Clear, concise, readable, mature, and polished; minimum of grammatical errors.

5 4 3 2 1 Mechanics: Chapter and section headings meaningful and consistent; proper use of formatting conventions; pages numbered; capitalization standard; minimum of typographical errors.

Supporting Documentation:

5 4 3 2 1 Title: accurate, complete and concise.

5 4 3 2 1 Abstract: miniature of report, summarizes argument and conclusions.

5 4 3 2 1 Table of Contents and List of Figures: neat and complete.

5 4 3 2 1 Glossary: informative, helpful and grammatically consistent.

5 4 3 2 1 Appendices: informative and neatly presented.

5 4 3 2 1 IF LITERATURE REVIEW: Summarizes, interprets, and synthesizes relevant sources; draws new conclusions for which a convincing argument has been logically developed throughout the report.

APPROVED _____ DATE: _____

(TCC Advisor's Signature)

FINAL ASSESSMENT

Thesis Author _____

_____ Not approved at this time. Revision required according to comments below and on reverse. Submit rewrite with this evaluation, technical advisor's evaluation, and original report.

_____ Approved, provided typographical/mechanical errors are corrected; revision is recommended to improve grade, but is not required. Submit rewrite with this evaluation, technical advisor's evaluation and original report.

_____ Approved. Correct typographical/mechanical errors for final binder.

CURRENT GRADE: _____ GRADE AFTER REVISION: _____

Table of Contents

| | |
|---|----|
| Glossary of Terms | 1 |
| Abstract | 2 |
| 1.0 Introduction | 3 |
| 1.1 Introductory Statement | 3 |
| 1.2 Problem Definition | 4 |
| 1.3 Rationale and Objectives | 5 |
| 1.4 Impact Statement | 6 |
| 2.0 Methods | 7 |
| 2.1 Overview | 7 |
| 2.2 Phase I – Structure Program Development | 8 |
| 2.3 Phase II – The Hash Table Data Structure | 10 |
| 2.4 Phase III – The Selective Author Server | 13 |
| 3.0 Results | 16 |
| 3.1 Overview | 16 |
| 3.2 Test 1 – Accuracy | 16 |
| 3.3 Test 2 – Data Structure Loading Time | 17 |
| 3.4 Efficiency Tests | 18 |
| 3.4.1 Test A – Optimized Delta Time | 18 |
| 3.4.2 Test B – Manipulation Time | 19 |
| 3.5 Conclusion | 19 |
| 4.0 Discussion | 20 |
| 4.1 Objectives Discussion | 20 |
| 4.2 Future Research | 21 |
| Bibliography | 23 |
| Appendix A | 25 |
| Appendix B | 27 |
| Appendix C | 29 |
| Appendix D | 30 |

Glossary of Terms

AuthorIndex – Defined as a global *dictionary array* with a *hash table implementation parameter*. Used to store the author names and their related sites.

C++ – An object-oriented programming language widely used today.

Dictionary Array – A data structure, which organizes specific items. Every item in the dictionary contains a key, a user defined linearly ordered data type, and corresponding information, a user defined data type.

Directed Query – A query where specific sites are select to be searched.

Fielded Search Form – the *NCSTRL* form that needs to be filled out to conduct a search over the collection. Located at: <http://www.ncstrl.org/Dienst/UI/2.0/Search>

Hash Table – A large array to which items map to a specific position through the use of a hash function.

Implementation Parameter – A parameter included in the definition of a *LEDA* data structure used to define how that data structure will be implemented.

LEDA – A library of the data types and algorithms of combinatorial computing.

LEDA set – A *LEDA* structure modeled after basic set theory.

Manipulation Time – The time it takes for an undirected query to be returned to the client in its new directed form.

NCSTRL – The Networked Computer Science Technical Report Library.

Optimized Delta Time – The time difference between searching all the sites as oppose to only the relevant ones.

Perl – A powerful, string processing programming language.

Search Fields – The portion of a query that holds the search information (e.g. sites to search, author names, title, etc.)

SQL – Standard search query language. Developed to perform logical operations over sets of data to obtain a specific set of desired information.

Undirected Query – A query where all sites are to be searched.

Unix – A common operating system developed for commercial and industrial use.

Abstract

The Networked Computer Science Technical Reports Library (*NCSTRL*) is an international collection of computer science technical reports from CS departments and industrial and government research laboratories, made available for non-commercial and educational use[1]. The overall purpose of the thesis project is to increase the speed of information retrieval within *NCSTRL* by implementing a selective author server, which would modify queries to be directed to select relevant sites rather than the entire *NCSTRL* collection. Studying the collection reveals a low distribution of author publications at multiple sites. Comparing the speed of a *directed query*, a query where specific sites are select to be searched, versus an *undirected query*, a query where all sites are to be searched, clearly justifies the need for a selective author server to perform this transformation. A *directed query* performs approximately 2.4 times faster than an *undirected query*. A Perl script downloads the author information and transforms that data into a text database. The selective author server, implemented in C++, loads this information into a *hash table* data structure for fast access. When an *NCSTRL* search involves an author and has the ‘search all organizations’ button depressed the undirected query is sent to the selective author server for transformation. The *manipulation time* of this transformation is an average 4.5 milliseconds, significantly less than the time difference between a *directed query* and an undirected one. Thus, it favorable to have a selective author server linked to the *NCSTRL* search engine.

1.0 Introduction

1.1 Introductory Statement

The thesis project I have chosen to design and implement is a selective searching mechanism to be integrated into the Networked Computer Science Technical Report Library (*NCSTRL*). With the rapid growth of distributed networks, retrieval of information has slowed down considerably. One of the great research challenges posed by this environment is the efficient and effective search of the vast distributed archive for useful information[2]. This project is an attempt to increase the speed of information retrieval in *NCSTRL*.

1.2 Problem Definition

A distributed IR collection is a series of computer servers linked over a network to handle large tasks. Distributed information retrieval is often based on a model where many independent servers index local document collections and a directory server (or servers) guides users towards the independent indexes[8]. This model assumes that the documents stored at a particular location define a collection[8]. The problems with current distributed networks is that as you scale the size of the network, the speed of information retrieval is slowed by a magnitude related directly to the network size. This slow down is detrimental to information systems because the need for larger distributed systems is becoming necessary with the growing trends in the scientific community. Take the Internet for example: search times have grown monstrous as the Internet continues to become more popular.

The Networked Computer Science Technical Reports Library (*NCSTRL*) is an international collection of computer science technical reports from CS departments and industrial and government research laboratories, made available for non-commercial and educational use[1]. The *NCSTRL* collection is distributed among a set of inter-operating servers operated by participating institutions[1]. The library can be accessed with a web browser at <http://www.ncstrl.org>. The major concern is that *NCSTRL* is rapidly growing causing the searches to become increasingly less efficient.

1.3 Rationale and Objectives

The goal I have undertaken with this thesis project is to increase the efficiency of document retrieval within *NCSTRL*. Increased efficiency will allow relevant documents to be retrieved from the library faster, providing the searcher more convenience. Presently, the search engine probes every site when a single query is submitted, unless the user through a selective option in the user interface restricts the sites searched.

My primary objective is to implement a selective search system that would eliminate the need to search sites that do not contain relevant documents, thus increasing retrieval efficiency. This modification would be transparent to the user who would be conducting a query as normal. A database containing specific knowledge of the contents of each site would be referenced. This would allow the search engine to direct the query to only sites that may contain relevant documents. Optimally, the selection system would return the exact same results faster than a search of the entire set of repositories. In the future, *NCSTRL* will be merged with other digital libraries and the number of repositories added to the collection will increase even further[3]. It is obvious that selective searching will be desirable, it might even be necessary for the survival of *NCSTRL* and other distributed libraries.

1.4 Impact Statement

The changes I have proposed to the search engine will be a necessity, rather than a convenient feature, as the Networked Computer Science Technical Report Library continues to grow. As the number of participating sites in the library increases the efficiency in the present search engine decreases. Presently each selected site has to bear the load of each request. As the number of repositories increase there will be more time-outs per search resulting in extended retrieval delay. A search engine capable of appropriately selecting only sites containing potentially relevant documents would prove to be more efficient than a search engine that performed an extensive, blind search.

The only negative impact would result if the selective searching system is not implemented correctly. Incorrect implementation could result in the failure to deliver relevant information to the end-user. The modified system should return the same results as the pre-selective system. With proper care and testing this negative aspect will be avoided.

2.0 Methods

2.1 Overview

The following sections will describe the procedures I use to increase the overall efficiency of document retrieval within *NCSTRL*. The project is divided into three phases each reflecting an important, unique aspect of the development of the final product. The specific design I have chosen to implement is a selective searching server that will direct queries only to potentially relevant sites rather than the entire spectrum. The server is modeled as a selective author engine, which means it only affects queries that have specified an author in the search form. A *hash table* data structure, containing authors in the *NCSTRL* collection and their corresponding publishing institutions, is an integral part of the server. Since the *NCSTRL* collection is frequently growing, the *hash table* is updated periodically by the selective author server. The user can set the specific interval of time between updates. This updated information is obtained from a text file created by a structure program. The structure program accesses all the author information and restructures it into a readable format, a textual database. Another integral part of the server is a protocol, which listens for an *undirected query*, one that has a non-empty author field and is to be sent to all sites. This query is cross-referenced against the data structure to determine to which specific sites to direct the query. Once this information is obtained the query is restructured and returned to the original *NCSTRL* server for distribution. The process is transparent to the end user meaning the user is unaware of the restructuring of the query. In the long run this selective author server will serve as an archetype for various other selective servers within *NCSTRL* and within other distributed digital libraries.

2.2 Phase I – Structure Program Development

To develop a fully functional selective author server, the first task that needed to be completed is to obtain a list of all the authors and sites. The *NCSTRL server* present at the University of Virginia Computer Science has a script program that downloads all the bibliographic information of all the *NCSTRL* sites. This information is vital for several research studies conducted by the Information Retrieval Research Group. Another script program was developed to create an author text file for each site in the collection. Each text file name is exactly the same as the site's URL address; within the each text file is a list of every author published at that site.

The structure program's job is to take the author text files and create a large text database. I decided to write this program in the programming language *Perl* because of its exceptional string handling capabilities. Perl includes a variety of functions that allow for complex string manipulation and pattern matching. The functionality of the program is divided into three parts.

In Part I the program first opens a temporary file named `authors.temp` for the output storage. Each author file is opened and the author's name is read from the file into a storage buffer, by means of an iterative loop. The name of the file, which is the site, is appended to each authors name and then the entire buffer is written to the output file.

Part II sorts the output of `authors.temp` and outputs the result to `authors.temp2`. There are two main reasons why the list of authors needs to be sorted. The primary reason is as follows: There is a list of each author matched with one site of publication. If an author is listed at more than one site there is an element pair within the list

representing each occurrence of that author. By sorting the list, these occurrences are grouped together so they can be easily merged by Part III. This creates an n-tuple element, where n is the number of sites an author is published at. The secondary reason is a textual database is more efficient when sorted. The sorting method I chose was to have the *Perl* script call the *Unix* system sort function. I choose this because the sort is reasonably efficient and easy to implement.

The final stage, Part III, loops over the sorted list and creates the text database of authors and matching publishing institutions. All element pairs with the same author are merged together to create a single row within the database. The author's name is always located in the first field and each institution is located consecutively in the remaining fields within that row. The beauty of a text database is there is no limit on the size and number of the fields. The following figure shows a fragment from the database.

```
ferguson ercim.forth.ics ncstrl.lite
fernandez ncstrl.lite ncstrl.princeton
fernau ncstrl.ubka_cs
ferrante ncstrl.cornell.tc ncstrl.lite ncstrl.mit.lcs
ferrari ncstrl.cornell ncstrl.lite ncstrl.ucb
ferreira ncstrl.cornell ncstrl.lite
ferrie ncstrl.lite
ferrill ncstrl.mit.lcs
ferris ncstrl.lite ncstrl.uwmadison
fessler ncstrl.lite ncstrl.tu.munich_cs
feustel ncstrl.vatech_cs
fialli ncstrl.umassa_cs
fiasconaro ncstrl.mit.lcs
fichtner ncstrl.lite
fidler ncstrl.lite
field ncstrl.cornell ncstrl.lite
fielding ncstrl.uci
```

Figure A. Excerpt from author.txt database

2.3 Phase II – The Hash Table Data Structure

To choose the proper data structure for storage of the textual database, I had to examine three key properties common to all data structures: information access time, new information storage time, and deletion time. The purpose of my thesis to increase the speed of information retrieval within *NCSTRL*, therefore the only true relevant factor is access time. The data structure is reconstructed from scratch and is never updated dynamically, therefore new information storage time and deletion time are irrelevant.

I choose to store the information in a *hash table* format because it has an average access time of $O(1)$. This is fastest possible access time of a data structure, and no other data structure could match this. Instead of creating data structures from scratch I will implement predefined structures from the *LEDA* library. *LEDA* is a library of the data types and algorithms of combinatorial computing [11]. The reasons I selected *LEDA* are as follows:

- *LEDA* provides a sizable collection of data types and algorithms, each with a precise and readable specification.
- *LEDA* contains efficient implementations for each of the data types, e.g., Fibonacci heaps for priority queues, red-black trees and dynamic perfect hashing for dictionaries, ...
- *LEDA* is implemented with a C++ class library. It can be used with almost any C++ compiler.

[11]

The *LEDA* data structure I defined for storage, named *AuthorIndex*, is a global *dictionary array* with a *hash table implementation parameter*. A *LEDA dictionary array* is a data structure, which matches a unique key of a specified type to a piece of information of another specified type. An *implementation parameter* is used to define how an ambiguous *LEDA* data structure will be implemented. The key of the *AuthorIndex* is a string, which is used to represent the author's name. The Information portion of *AuthorIndex* is a *LEDA set* data structure. The *LEDA set* is modeled after basic set theory, and is implemented as a linear list of elements of a specified data type. Unions, intersections and other set operations can be performed over the *LEDA set*. This is the very reason I chose to store the sites in a set as oppose to a basic linear list. If an individual desired to search all the sites which have papers co-authored by a specific group of authors, a simple set intersection would yield the results. If an individual desired to search all the sites containing papers by one or more authors of a specific group, a simple set union would yield the results.

The *dictionary array's* scope is global to allow access by the server as well as the build function. The build function's purpose is to load the structure. The function does this by first opening the text database created by the *Perl* structure program. The author name, which is located in the first field of the text database, is loaded into a temporary key variable. An iterative loop extracts all the sites located within the same line, inserting them into a temporary set. The temporary key and set variables are then inserted into the *AuthorIndex* for permanent storage. Next, the temporary set and key then are cleared for reuse. This process is iterated until the end of the text database file is reached. Once this process is finished the *AuthorIndex* is complete.

The following figure shows a graphical representation of *AuthorIndex*, displaying how information within the structure is related.

AuthorIndex

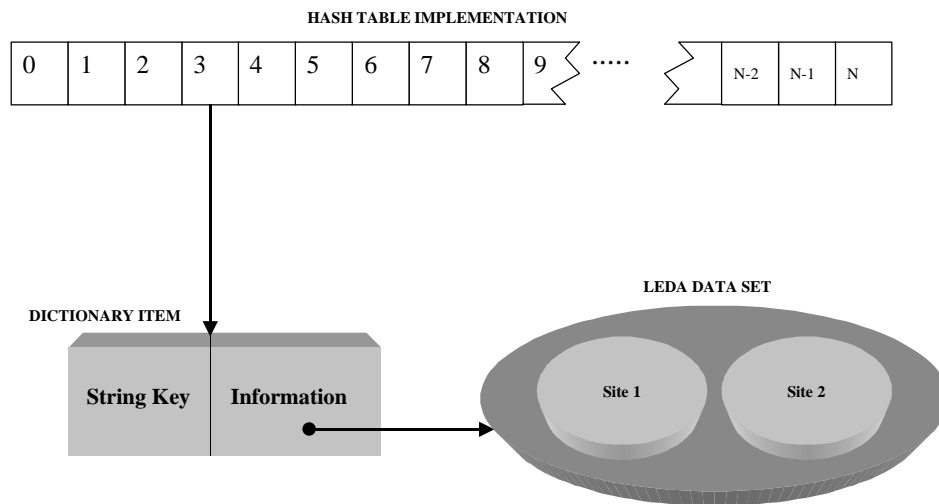


Figure B. Graphical Representation of *AuthorIndex*.

2.4 Phase III - The Selective Author Server

The server acts as the main driving engine behind the scene. It has the job of accepting, evaluating, and modifying queries. Queries are sent to the server, evaluated to determine the necessary changes, modified, and returned to the *NCSTRL* server. This process needs to be optimized to occur as fast as possible. A more efficient result is produced as long as the *manipulation time*, the time required to perform this operation, is not greater than the *optimized delta time*, the time difference between searching all the sites as oppose to the relevant ones.

I chose to implement the server in *C++* for two reasons. The primary reason is because the server needs to access the data structure, which is also written in *C++*. The second reason is because I am familiar with *C++* socket protocol. The first step in constructing the server is to set up a socket to accept incoming character streams. After the socket is initialized, and a port is established, the server enters an infinite loop, which listens for incoming queries.

Contained within this loop is the algorithm that performs the necessary modifications to the query. When a query is first received from the socket it is stored in a character buffer array. Next a loop is performed over the array evaluating each character. If the character is not an ampersand (&), it is appended to a temporary string named *Qsection*, which is initially 'empty'. The ampersand character separates the *search fields*, parts of a query that determine where to search. If an ampersand is reached *Qsection* is stored as an element in a *LEDA list* named *querystore*. If this occurs *Qsection* is set to 'empty' and the loop resumes to repeat the process until another ampersand or the end of the query is reached.

Next the evaluation of the query occurs. The second to the last item of *querystore* contains information on whether all sites are to be searched. This item is evaluated to determine if the query needs to be modified. If so the modification algorithm begins to take place.

The modification algorithm works in a rather simple procedural fashion. At this stage the query is already broken up into each relevant search field. The first four fields contain the boolean, author, title, and abstract search keys in this respective order. The last field contains the name field. The remaining fields, between the fourth and the last field, contain the information about which sites need to be searched. The algorithm first takes the first four fields and joins them back together with ampersand characters and stores this into a string variable named *query_mod*. Next the author field is stripped to obtain the author name. The name is referenced into the *dictionary array AuthorIndex* to obtain the set containing all of the publishing sites. The set is iterated over and each site is appended to *query_mod* along with an ampersand character as a separator. Finally the name field is appended to *query_mod*. The modified query is then placed back into the character buffer by iterating over the *query_mod* string and placing each respective character into the matching position within the character buffer array. Finally, the modified query is sent back to the client.

The following figure shows a basic graphical representation of the Selective Author Server's roles.

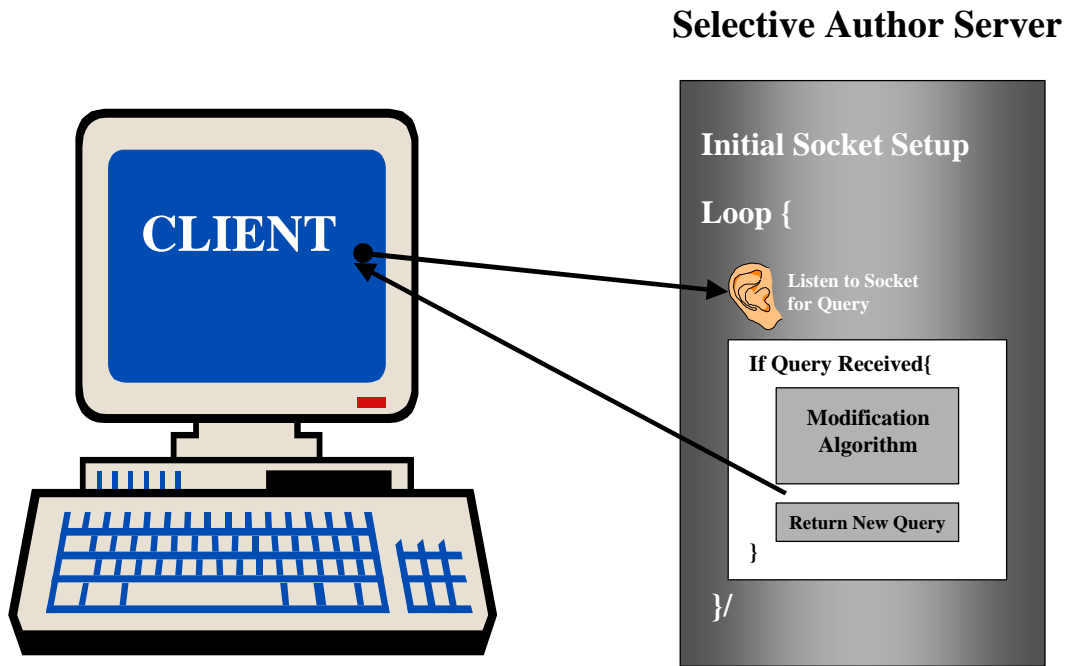


Figure C. Graphical Representation of the Selective Author Server.

3.0 RESULTS

3.1 Overview

The purpose of this thesis is to implement an author server, which would modify queries to be directed to select relevant sites rather than the entire *NCSTRL* collection.

This in turn will increase the performance of an *NCSTRL* search by decreasing the retrieval time for information. The selective author server would be accessed if the search all sites button was activated, and the author field was not left empty on the *fielded search form*. On sheer observation, a *directed query*, where the specific sites to be searched were selected, performs substantially faster than an *undirected query*.

To obtain results on the success of my server implementation, I performed several tests. Several of the tests were quantitative.

3.2 Test 1 – Accuracy

My first goal was to determine if an *undirected query* transforms successfully into a *directed query*. Several test cases were used and the return results were recorded. To determine if the correct sites were included, *undirected query* searches with the corresponding authors were conducted over *NCSTRL*. The resulting sites displayed by these searches were compared to the sites within the *search field* of the modified query provided by the author server. Each test case matched precisely.

3.3 Test 2 – Data Structure Loading Time

I decided to conduct a test to calculate the average time it takes to load the *AuthorIndex*. The rationale behind this test was to determine how long the server would have to be inoperable while the *AuthorIndex* is being rebuilt. This is fundamental to measuring efficiency.

I decided to conduct a series of 100 trials to obtain the average loading time. First I had to modify the author server to time the build function, and then terminate before it entered into the infinite loop. Next I created a simple *Perl* script to automate my test procedure. First, the script activated the server 100 times and output each timed result to a text file. Finally, the script computed the average loading time value and appended that to the end of this file. The output can be found in Appendix B.

The average time, based on my tests, for the data structure to be loaded is 2.85 seconds. This result is exceptional, due to the fact this process can be completed quickly during a time the server is not busy with a query.

3.3 Overview of Efficiency Tests

My next goal was to test the speed of the author server. If I could show the query *manipulation time* is less than the *optimized delta time*, the time difference between searching all the sites as oppose to the relevant ones, the addition of the author server would prove to be beneficial. Moreover, the efficiency of searches over the *NCSTRL* collection will greatly increase if this *manipulation time* was significantly less.

3.4.1 Test B - Optimized Delta Time

The *optimized delta time* is the time difference between searching all the sites as opposed to only the relevant ones. To obtain this *optimized delta time*, two sets of timings needed to be conducted. First an average *undirected query* time needs to be calculated. Second, an average *directed query* time needs to be calculated. The difference between the two is the *optimized delta time*.

First the *undirected query* with author name = french was selected. A series of 20 trials with a stopwatch were conducted. The test data can be found in Appendix C. The results yielded an average of 26.20 seconds.

Second a *directed query* with author name = french was selected and the appropriate sites were selected in the fielded search form. A series of 20 trials with a stopwatch were conducted. The test data can be found in Appendix D. The results yielded an average of 10.75 seconds.

The average *optimized delta time* is: $26.20 - 10.75 = 15.45$ seconds

3.4.2 Test A - Manipulation Time

To test the *manipulation time* I modified the C++ client to send 1000 queries to the author server and wait for the modified versions to return. The purpose of measuring a time block of 1000 queries in succession is to obtain an average value for one query. I edited the client to time this process and output the timed result. Next I modified the *Perl* script used in Test 2, to automate this test procedure. First, the script activated the client 100 times and output each timed result to a text file. Next, the script computed the average value and appended that to the end of this file. The output can be found in Appendix A.

The average time, based on my tests, for 1000 queries to be modified and returned to the client is 4.52 seconds. Thus, the average time for 1 query is 4.52 milliseconds. This result is phenomenal, because this value is remarkably less than the average *optimized delta time*, whose value is a factor of seconds.

3.5 Conclusion

Based on the previous tests, implementing the selective author server would greatly improve the efficiency of *NCSTRL*. The average *manipulation time* of 1 query is significantly less than the average *optimized delta time*. My results support the notion that the addition of the selective author server would decrease the retrieval time by a factor of approximately 2.43.

4.0 Discussion

4.1 Objectives Discussion

The overall purpose of the thesis project is to increase the efficiency of information retrieval within NCSTRL. Several benchmarks were outlined in this attempt to meet this goal.

First of all, it was necessary to determine if a *directed query* was more efficient than an undirected one. This was determined with relative ease by performing a series of timed tests. Testing shows the speed of a *directed query* is more than twice as fast as an undirected one. This provides the justification for the development of a selective author server.

Secondly, the selective author server needed to be developed. This was implemented successfully. Author based, *undirected queries* were modified to the appropriately matching, *directed queries*. The relevant sites of the modified, *directed queries* matched the returned sites of the corresponding *undirected queries*. Thus, no relevant sites were lost in the transformation.

Third, the speed of the server's ability, to accept an *undirected query* and return it's equivalent directed query, needed to be tested. This *modification speed* turned out to be several fractions of a second. Since the *modification speed* is extremely low, linking the author selective server to *NCSTRL* will not reduce speed of the information retrieval.

Finally, the selective author server needed to be linked to the *NCSTRL server*.

4.2 Future Research

As with most thesis projects there is always room for future research to expand on. The selective author server only improves information retrieval time when searching by author. Since, the *NCSTRL* collection is steadily growing there will be a need to investigate alternative methods to increase the efficiency of all categories of searches. Future research would involve exploring how the property of selectivity could be expanded to handle different information categories other than author. In the future the author server may function as an archetype for the development of other selective servers within the *NCSTRL* collection and other distributed libraries.

Other areas of future research involve the modification of the query manipulation algorithm located within the selective author server. Perhaps the algorithm could be written more efficiently. Currently, the algorithm works for only searches involving a single author. The *AuthorIndex* uses the *LEDA set* to hold sites, making the current code maintainable and expandable. Through the use of the *LEDA sets*, unions, intersections and other set operations can be performed. These operations will support the boolean logic used in searches involving multiple authors, effectively duplicating the properties of *SQL* operations. Expanding the server to handle multiple authors opens up an exciting area of future research.

Another future area of research involves determining why a directed query performs only at a constant improvement of around 2.4. One would guess that this time would be linear, based on the number of sites excluded from a search. My results however did not yield this. Perhaps several sites are packaged regionally by the *NCSTRL* server effectively neutralizing linear results. Perhaps there is a constant client

side set up time that is creating a distortion to the values. Exploring how the client side operates may help to learn more ways to increase efficiency within *NCSTR*L.

Bibliography

1. Networked Computer Science Technical Report home page. <http://www.ncstrl.org/>. Maintained at Cornell University.
2. Viles, Charles L. & French, James C. "Dissemination of Collection Wide Information in a Distributed Information Retrieval System" proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995.
3. Davis, James. September 1995. "Creating a Networked Computer Science Technical Report Library." D-Lib Magazine. Obtainable through NCSTRL.
4. Schatz, Bruce R. January 1997. "Information Retrieval in Digital Libraries: Bringing Search to the Net." Science. 275:327-334.
5. Salton, Gerard. August 1991. "Developments in Automatic Text Retrieval." Science. 253:974-980.
6. Harter, Stephen P. "Scholarly Communication and the Digital Library: Problems and Issues". Paper from School of Library and Information Science, Indiana University.
7. The National Center for Intelligent Information Retrieval. <http://cobar.cs.umass.edu/> University of Massachusetts at Amherst.
8. Hylton, Jeremy. Creating Collections with a Distributed Indexing Infrastructure. <http://www.cnri.reston.va.us/home/dobi/collections.html> Position statement for Distributed Indexing/Searching Workshop held at MIT, May 28-29, 1996.
9. Definition and Purposes of a Digital Library. <http://ifla.inist.fr/ifla/documents/libraries/net/arl-dlib.txt>. Association of Research Libraries October 23, 1995
10. Ackernman, Mark S. & Fielding, Roy T. "Collection Maintenance in the Digital Library". <http://www.csdl.tamu.edu/DL95/papers/ackerman/ackerman.html>. Maintained at Information and Computer Science University of California, Irvine.
11. Mehlhorn, Kurt & Naher, Stefan & Uhrig, Christian. "The LEDA User Manual Version 3.5.2"
12. Schwartz, Randal L. "Learning Perl", 1993 O'Reilly & Associates.

13. Oram, Andrew & Talbot, Steve. "Managing Projects with make", 1991 O'Reilly & Associates.

Appendix A

The following is the output file for 100 query modification trials. Each number represents, in seconds, the time it took for 1000 modified *directed queries* to be returned to client who sent the original *undirected queries* to the server. The number is a composite value of the network traffic time plus the actual transformation time. The average time, which is 4.5 seconds, is computed at the end. The average time for 1 query is 4.5 milliseconds.

3.500000
4.300000
4.400000
4.900000
3.900000
5.300000
4.100000
5.300000
4.600000
4.100000
4.600000
4.700000
5.100000
5.500000
6.000000
5.900000
4.600000
4.200000
5.100000
5.300000
4.500000
3.600000
4.100000
4.100000
4.500000
4.100000
3.700000
4.200000
4.300000
4.300000
3.800000
4.600000
5.100000
5.300000
4.200000
4.600000
3.800000
4.300000
4.600000
5.000000
5.600000
5.500000
4.700000
4.600000
4.900000

4.900000
4.100000
4.200000
5.000000
4.800000
4.400000
3.300000
3.700000
4.400000
4.700000
4.600000
4.100000
4.700000
4.200000
3.500000
4.000000
4.200000
2.900000
4.300000
4.000000
3.600000
4.700000
4.900000
4.400000
5.700000
4.100000
5.100000
4.000000
4.400000
4.100000
4.400000
5.800000
3.900000
6.000000
4.200000
4.300000
4.300000
4.800000
3.900000
5.000000
4.900000
4.000000
4.200000
4.900000
4.200000
4.800000
4.600000
3.400000
5.200000
5.900000
4.900000
4.300000
4.200000
5.200000
4.100000

Avg of 4.5s for 1000 query modifications.

Appendix B

The following is the output file for 100 build trials. Each number represents, in seconds, the time it took for the *AuthorIndex LEDA dictionary array* to be loaded. An average time of 2.85 seconds is required to currently load the data structure. Currently the number of authors in the *NCSTRL* collection is 7437.

```
2.690000
2.930000
2.870000
2.740000
3.240000
2.720000
2.810000
2.930000
2.830000
2.920000
2.820000
2.910000
2.710000
2.780000
2.810000
2.820000
2.650000
2.630000
2.700000
2.870000
2.860000
2.720000
2.640000
2.700000
2.950000
3.020000
2.890000
3.150000
2.830000
2.970000
3.200000
2.990000
3.180000
2.840000
2.860000
2.760000
2.730000
2.790000
2.720000
2.750000
2.740000
2.730000
3.150000
2.730000
2.740000
2.710000
```

2.840000
2.760000
3.050000
3.060000
2.800000
2.700000
2.940000
2.700000
2.740000
2.750000
2.860000
2.760000
2.800000
2.860000
2.710000
2.760000
2.910000
2.850000
2.700000
2.730000
2.810000
2.720000
3.140000
2.830000
2.750000
2.750000
2.930000
3.230000
2.840000
2.870000
2.860000
2.830000
3.210000
2.840000
2.920000
2.900000
3.230000
2.950000
2.670000
2.720000
2.770000
2.800000
2.750000
2.840000
3.190000
2.970000
2.950000
2.940000
2.720000
2.730000
2.790000
2.750000
2.920000
2.740000

Avg Build Time = 2.85s

Appendix C

The following values were recorded with a stopwatch. Each value represents the time, in seconds, it took the *NCSTRL* search engine to search all institutions for papers with the author name = french. Each trial was conducted immediately after each other to reduce possible variances in time.

| | |
|-----------|-------|
| Trial 1: | 23.33 |
| Trial 2: | 37.58 |
| Trial 3: | 23.50 |
| Trial 4: | 23.21 |
| Trial 5: | 25.02 |
| Trial 6: | 36.27 |
| Trial 7: | 25.39 |
| Trial 8: | 21.40 |
| Trial 9: | 22.45 |
| Trial 10: | 25.29 |
| Trial 11: | 24.06 |
| Trial 12: | 23.41 |
| Trial 13: | 25.53 |
| Trial 14: | 35.38 |
| Trial 15: | 22.11 |
| Trial 16: | 36.05 |
| Trial 17: | 24.49 |
| Trial 18: | 24.08 |
| Trial 19: | 23.14 |
| Trial 20: | 21.27 |

The average time is: 26.20 seconds

Appendix D

The following values were recorded with a stopwatch. Each value represents the time, in seconds, it took the *NCSTRL* search engine to search only relevant institutions for papers with the author name = french. Each trial was conducted immediately after each other to reduce possible variances in time.

| | |
|-----------|-------|
| Trial 1: | 10.47 |
| Trial 2: | 11.21 |
| Trial 3: | 11.13 |
| Trial 4: | 10.42 |
| Trial 5: | 10.56 |
| Trial 6: | 10.55 |
| Trial 7: | 11.26 |
| Trial 8: | 11.01 |
| Trial 9: | 10.52 |
| Trial 10: | 10.15 |
| Trial 11: | 10.59 |
| Trial 12: | 11.16 |
| Trial 13: | 10.52 |
| Trial 14: | 10.30 |
| Trial 15: | 11.05 |
| Trial 16: | 11.03 |
| Trial 17: | 11.10 |
| Trial 18: | 10.26 |
| Trial 19: | 11.15 |
| Trial 20: | 10.48 |

The average time is: 10.75 seconds