2008-08-13

An Infrastructure for Performance Measurement and Comparison of Information Retrieval Solutions

Gary Saunders
Brigham Young University - Provo

AN INFRASTRUCTURE FOR PERFORMANCE MEASUREMENT AND COMPARISON OF INFORMATION RETRIEVAL SOLUTIONS

by

Gary Saunders

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

School of Technology

Brigham Young University

December 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Gary Saunders

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Joseph J. Ekstrom, Chair
Date	C. Richard G. Helps
Date	Chia-Chi Teng

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Gary Saunders in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

_____ _____

Date

Joseph J. Ekstrom
Chair, Graduate Committee

Accepted for School

_____

Barry M. Lunt
Graduate Coordinator

Accepted for the College

_____

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

AN INFRASTRUCTURE FOR PERFORMANCE MEASUREMENT
AND COMPARISON OF INFORMATION
RETRIEVAL SOLUTIONS

Gary Saunders

School of Technology

Master of Science

The amount of information available on both public and private networks continues to grow at a phenomenal rate. This information is contained within a wide variety of objects, including documents, e-mail archives, medical records, manuals, pictures and music. To be of any value, this data must be easily searchable and accessible. Information Retrieval (IR) is concerned with the ability to find and gain access to relevant information.

As electronic data repositories continue to proliferate, so too, grows the variety of methods used to locate and access the information contained therein. Similarly, the introduction of innovative retrieval strategies—and the optimization of older strategies—emphasizes the need for an infrastructure capable of measuring and comparing the

performance of competing Information Retrieval solutions, but such an environment does not yet exist.

The purpose of this research is to develop an infrastructure wherein Information Retrieval solutions may be evaluated and compared. In 1979, an expert in the field believed the need for a system-independent benchmarking utility was long overdue—twenty-five years later, progress in this area has been minimal. Contrastingly, new theories have emerged; new techniques have been introduced; all with the goal of improving retrieval performance. The need for a system-independent analysis of retrieval performance is more critical now.

ACKNOWLEGEMENTS

This thesis is the result of the combined efforts of many people; I am grateful for their advice, support, and encouragement. I thank the executives at WriteExpress.com and Cemaphore Systems for their support of this research, including funding used to create the test environment. I thank my graduate chair, Dr. Joseph Ekstrom, for his vision and the time he has dedicated to this endeavor. I thank my wife, Bridgette, who inspires me to dream and empowers me to achieve.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 INTRODUCTION

1.1 Background

The amount of information available on both public and private networks continues to grow at a phenomenal rate. This information exists in a wide variety of formats, including word processing documents, e-mail, spreadsheets, medical records, manuals, pictures, audio files, and many more. However, to be of any significant value to users, this information must be readily accessible.

In its broadest sense, Information Retrieval (IR) is concerned with the ability to find and gain access to relevant information. It follows that two primary components are required in order for IR to take place. First, there must be a collection of information. This collection may comprise the files on a personal computer, the electronic texts available in a public library, breaking news stories available via online news reporting services, or the content of a group of web pages. The second component required for IR is an information need. For text retrieval, this may consist of one or more keywords of interest to the user. In a library, it might include the author, title, or date of publication of a particular literary work. IR then occurs as a user's search criteria are compared to the contents of the information repository and relevant matches are presented to the user.

Information Retrieval is categorized as either *ad hoc* or filtered. In an *ad hoc* environment, such as a library, the information collection remains relatively static while

the search criteria vary from user to user. In a filtered environment, such as a subscription to an online news feed, the opposite is true: the search criteria remain constant while the information collection—in this case, the news stories—changes from day to day. In the past, *ad hoc* systems have been more common because of their academic applications. However, with the advent of online information services and other applications that favor filtered retrieval, some anticipate increased attention to be given to filtered retrieval strategies (Baeza, 1999).

Whether *ad hoc* or filtered, Information Retrieval comprises three basic steps. First, an index is created for the information collection. Akin to a table of contents, an index may be as simple as a list of keywords found in each document or as complex as a list of keywords, spelling variations, synonyms and syntactical locations of each. An index may include every word within a document or strategies may be employed to eliminate stopwords (i.e., "the" and "a") from the index. Once an index of the collection has been created, a user's search criteria are parsed in a similar fashion. For example, if stopwords are removed during the indexing process, then stopwords will also be removed from the user's search criteria. Finally, a comparison of the index and parsed search criteria is performed. Matches are determined based on rules defined by the IR model implemented by the retrieval software. There is a great interdependency between these processes and, as a result, the development of IR software and research into new strategies must necessarily focus on each (Croft, 1992).

Information Retrieval models specify the methodology for generating indices for collections and search criteria. There are several, although most models can be classified into one of two general categories. Classic, or Traditional, models create an index of

keywords to represent objects in a collection. These models allow users to search for objects containing a certain single word or phrase. In addition to the information contained within the objects in the collection, an index may also include information about the objects, for example, the title, author, date of publication, etc. Slightly more complicated, Structured Text models index both content and structure, thus preserving an object's original format. These models allow for more elaborate information needs: for example, a user may search for objects containing a certain word or phrase occurring within the first paragraph. The processes of creating and maintaining an index and interpreting search criteria have the greatest impact on overall system capability and performance (Robertson, 1976 and Burkowski, 1992 and Navarro, 1997).

1.2 Problem Statement

As electronic data repositories continue to proliferate, Information Retrieval will play a pivotal role in the identification and accessibility of relevant information. It follows that IR performance will play a pivotal role in the identification and implementation of IR solutions that best match operating environments. At its core, IR performance is the measure of two characteristics: the speed with which the IR processes are executed and the accuracy of the results. Some operating environments may place more value on one or the other. The growing variety of implementation scenarios and IR strategies introduce a need for an infrastructure capable of measuring and comparing the performance of competing software solutions, but such an environment does not yet exist.

In the past, the evaluation of IR solutions has comprised two steps. First, develop and build an IR solution. Second, design and build a test environment around the solution. Gao, Murugesan, and Lo (2004) explained that "current evaluation methods are

system-oriented…. These methods have limitations in achieving the objective of evaluation for improvement of information retrieval results." System-oriented evaluations, they cited, have an inherent inability to identify the causes behind performance differences. Such evaluation models fail to consider factors beyond the IR solution (i.e., available system resources, incompatible file formats, etc.).

In 1979, an expert in the field noted the need for a system-independent benchmarking utility as long overdue—twenty-five years later, progress in this area has been minimal. Contrastingly, several advances in retrieval theories and parsing techniques have been introduced; all with the goal of improving retrieval performance. The need for a system-independent analysis of retrieval performance is more critical now (Cahoon, 2000 and Van Rijsbergen, 1979).

1.3 Hypothesis

The purpose of this research is to develop an infrastructure wherein IR solutions may be evaluated and compared. A valuable comparison of competing retrieval solutions may be obtained if an objective testing environment can be established. This environment will be capable of measuring how quickly certain tasks are performed, the resources required to complete those tasks, and the thoroughness of the results of those tasks. The following approach is used to design an infrastructure for the evaluation of IR solutions:

- Current measures are analyzed according to their ability to objectively evaluate retrieval performance and how those measurements are obtained.
- A testing environment is created consisting of multiple randomly-generated object collections of varying sizes.

- An analysis is performed on several IR solutions, including the measure of the time required to index the object collections, maintain (i.e., update) the index, and execute a series of queries against the object collections.
- The results of the analyses are subjected to a set of user-defined heuristics for an objective side-by-side comparison and identification of the best-suited IR solution.

In addition to timed experiments, further tests are executed to determine the accuracy of each retrieval solution. These tests will attempt to exploit weaknesses of IR strategies, including, for example, the handling of abbreviations, acronyms, and hyphenations. These additional tests also set forth a method for creating an environment wherein more complicated measures, like Precision and Recall—to be described later—may be evaluated.

The results of the system-independent performance measurement will allow for an accurate and effective comparison of existing IR solutions. Measuring the time required to complete IR processes will identify the most efficient IR solutions. Measuring the retrieval accuracy—the *quality* of the results—in conjunction with the speed tests will add a second dimension to the time-based analysis and reveal any tradeoffs that exist between the qualitative accuracy and quantitative performance.

1.4 Justification

The objective of evaluating IR solutions goes beyond identifying the "best" solution for a given application. A useful evaluation model helps users decide if they want to use an IR solution and if it will be worth the investment. The evaluation model makes it easy to compare new algorithms to old ones. Where conflicting opinions exist,

an effective evaluation model serves to validate the one side or the other (Van Rijsbergen, 1979 and Landoni, 2000). Finally, an effective evaluation makes it possible to identify the strengths and weaknesses of a retrieval solution. Rather than simply report that Solution A is faster than Solution B, the side-by-side comparison will guide developers in the design of more efficient retrieval strategies (Gao, 2004).

Evaluation models should be designed with no particular IR solution in mind. In 1979, C. J. van Rijsbergen saw the need for a standardized benchmark when he noted that "[more research] on *experimental* information retrieval, covering the design and evaluation of retrieval solutions from a point of view which is independent of any particular system, will be a great help to other workers in the field and indeed is long overdue." This thesis will detail a new evaluation model independent of any IR strategy or operating environment.

There is need for an infrastructure designed to objectively measure the performance of competing IR solutions. The ability to quickly index an information collection, to keep the index as current as possible, and to retrieve relevant information are important characteristics to evaluate. Other important factors that contribute to a proper evaluation model include retrieval accuracy, resource utilization, and scalability. Such a utility will aid researchers and developers in the pursuit of superior retrieval strategies and assist IT professionals in the selection of an IR solution that best satisfies a customer's needs.

1.5 Assumptions

This research assumes that the resources required to measure retrieval performance will be equivalent despite the IR solution being tested. Thus, resources consumed for the administration of the testing infrastructure are ignored.

1.6 Delimitations

Due to the subjective nature of some popular measures (to be discussed later on) and because the usefulness of other measures is often determined by application and environment, this research will measure and compare a finite set of characteristics for each of the IR solutions.

While scores of IR solutions are available, this research will be limited to five popular open-source solutions. The solutions implement different retrieval strategies which result in significantly differently performance results. This is done to show the differences that exist due to increased functionality. For example, while one IR solution may take longer to build an index for an object collection, it allows for more complicated queries and, in the end, retrieves more relevant information.

In order to simulate a server environment, this research will be conducted on a Toshiba Satellite comprising one Genuine Intel® T2300 1.99GHz dual core processor and 0.99 GB of RAM. VMWare Player version 2.0.0 build-45731 will be installed. A Debian-based virtual machine will be established. The operating system is SuSE version 10.0; featuring Linux v2.6.11-9-amd64-k8-smp to take advantage of dual processing capability. The server is a dedicated retrieval server—no other applications will be running during the tests. The tests are executed against purely text-based collections—all files are stored electronically in plain text format.

Chapter 2

2 REVIEW OF LITERATURE

2.1 Information Retrieval

The history of information retrieval dates back to the third century B.C. when cataloging techniques were first used to store and access collections of over 100,000 documents (Hessel, 1955). The first electronic information retrieval solutions were introduced beginning in 1947 when Mooers invented the Zatacoding method of information retrieval. In that same year, Luhn developed the Luhn Scanner for the CIA. In 1950, IBM released the Electronic Statistical Machine, Type 101, used for literary searches by the US Patent Office. In 1954, the US Naval Ordnance Test Station completed what has since been called the first subject search ever made by a digital computer (Gull, 1987). Today, information retrieval occurs in variety of environments. A growing number of methods and strategies are used to find and retrieve data, and there are a number of popular measures used to analyze performance.

2.1.1 Retrieval Behavior

IR solutions allow users to browse through and retrieve information from an information collection. Browsing occurs when a user has an undefined information need. Information Retrieval occurs when a user has a known information need and is able to present that need to someone or something capable of interpreting it and returning

relevant information. Information Retrieval can occur in one of two ways: *ad hoc* or filtering.

2.1.1.1 Ad hoc Retrieval

The definition of *ad hoc* retrieval has changed over the past few years. As defined by Croft in 1992, *ad hoc* retrieval has no underlying theoretical foundation—as compared to theory-based formal information retrieval. More recently; however, an *ad hoc* system has been defined as a system wherein the document collection remains relatively static as the information need changes. *Ad hoc* retrieval is most common in libraries; where the overall information collection changes little over time (Beaza, 1999).

2.1.1.2 Filtered Retrieval

Filtered retrieval was introduced in the late 1980s and early 1990s and has become a popular retrieval method. In a filtering system, a user establishes a profile—an information need. This profile then remains constant as the contents of the information collection changes. An example of a filtered system is a subscription to an online news source. Rather than sort through all of the day's headlines, a user is notified when an article is matched to the stored profile. In the past, *ad hoc* systems have been the most common because of their academic applications; however, with the advent of numerous online applications that favor the filtering functionality, many anticipate increased attention to be given to filtering systems (Baeza, 1999).

2.1.2 Traditional Retrieval Models

A retrieval model specifies the methods and strategies used to retrieve information. Most models are classified as either traditional or structural. The most

prominent traditional models include the Boolean, vector space, cluster-based, and probabilistic models. They are presented here to demonstrate the differences between them and the implications of those differences.

2.1.2.1 Boolean Model

The Boolean model—the oldest of all retrieval models—allows users to submit queries based on Boolean algebra and Set Theory. The Boolean model is intuitive and was widely adopted in the past for use in libraries and commercial bibliographic systems. A weakness of the Boolean model lies in the rigid format of the queries: the strict matching requirements often ignore legitimate partial matches. Additionally, the lack of term weighting makes it impossible to return a list of ranked objects; they are all equally relevant. As a result, variants of the Boolean model were developed in an effort to relax the matching criteria for Boolean searches and thereby increase retrieval performance. These variants include the fuzzy-set and extended Boolean models.

The fuzzy-set model relaxes the rigidity of Boolean searches by allowing partial matches. Rather than classifying terms as member/non-member, terms are assigned a degree of membership in a set. The degree of membership is determined by a thesaurus or other term-matching utility. If an object contains a term that is similar to a term in a query, the object will be retrieved. Synonymous terms (i.e., computer and laptop) will have a higher degree of membership than different terms (i.e., computer and Sunday). In the end, the membership scores are used to sort and rank the retrieved objects (Baeza-Yates, 1999).

Salton introduced the extended Boolean model in 1983. This model extends the Boolean model with vector space model characteristics. In a standard Boolean AND

search for multiple terms (i.e., sports AND baseball AND homerun), objects that contain only one or two of the search terms are counted as irrelevant. The extended Boolean model implements vector space algebra to give these objects a partial relevancy. While generally believed to outperform the Boolean model, adoption of this model was little—due in large part to other models that will be presented hereafter (Salton, 1983).

2.1.2.2 Vector Space Model

Introduced by Salton in the early 1970s, the vector space model is arguably the most popular of all retrieval models. The vector space model translates document and queries containing *N* index terms into an *N*-dimensional vector. All vectors have a normalized length of one and their end-points lie somewhere on the resulting n-sphere. A query is a point on the sphere and relevant documents are points located nearby. Relevancy between a query and a document, then, is inversely proportional to the distance between the two points. The vector space model includes partial matches in the answer set and automatically generates a degree of relevancy—sometimes called a weight or rank—for each document. This degree of relevancy enables a system to present the retrieved documents in order beginning with the most relevant. Each indexed word is assigned a weight based on, among others, *tf* (frequency of the term in the object) and *idf* (frequency of the term in the collection). It is generally accepted that the vector space model is faster and returns better results than the Boolean model (Salton, 1975 and Salton, 1983).

Variants of the vector space model include the generalized vector space, latent semantic indexing, and neural network models. The generalized vector space model states that if two index terms are contained in the same object, there is dependence

between those terms. This dependence is combined with the typical weighting used by the vector space model to determine relevancy.

The latent semantic indexing model addresses the issue of ambiguity that arises due to the indexing of text. This model is based on the notion that much of the context associated with a term is lost during the indexing process. Rather than index documents by terms, the latent semantic indexing model groups objects together by concepts (Furnas, 1988).

The neural network model functions like the human brain. A network is created with three basic levels: queries, terms, and objects. A query containing one or more terms will activate the corresponding nodes and these nodes in turn activate object nodes. The process may repeat itself several times for a single query. Neural networks are able to retrieve relevant documents that often don't contain any of the terms set forth in the query.

Like the variants of the Boolean model, the generalized vector space, latent semantic indexing, and neural network models have failed to deliver significantly increased retrieval performance. As such, their adoption has been minimal (Baeza-Yates, 1999).

2.1.2.3 Cluster-based Model

The cluster-based model is based on the Cluster Hypothesis and assumes that an object's relevance correlates to other objects within the collection. The cluster-based model groups objects into clusters based on their similarity. The cluster is then represented by a sample object. The information need is then compared to each cluster of

objects rather than each individual object and the most relevant cluster is returned (Croft, 1992).

2.1.2.4 Probabilistic Model

The probabilistic model was introduced by Robertson and Sparck Jones in 1976. As the name implies, the probabilistic model utilizes probability theory to generate an ideal answer set for a query. An underlying assumption is that an object's relevance is completely independent of other objects in the collection (van Rijsbergen, 1979).

Variants of probabilistic model include the inference network and belief network models. These models are based on Bayesian Networks and seek to combine the strengths the Boolean and vector space models while eliminating their weaknesses. Indeed, in their simplest form, these models can be tuned to perform exactly as a Boolean or vector space model. These models; however, are also able to take advantage of multiple statistics (i.e., past queries and user feedback) when determining relevance. Turtle and Croft have shown that these models provide increased retrieval performance over other classic retrieval models (Croft, 1992).

2.1.3 Structural Retrieval Models

Structural—or structured text—retrieval models are very different from the traditional models. The first notable difference is that these models are founded upon concepts rather than accepted mathematical theory. Structured text models index both content and structure, whereas the classic models index only content. Thus, structured text models are able to preserve an object's original format. These models, then, are best distinguished by their method of text representation. Structured text models are able to

handle more complex queries. For example, a user may be searching for a keyword near the beginning of a chapter, or appearing on the left-hand side of a document. These models are based on non-overlapping lists or proximal nodes.

2.1.3.1 Non-overlapping Lists

The structural model based on non-overlapping lists divides the text of an object into distinct sections. A book, for example, might be divided into chapters, paragraphs, and sentences. A list is created for each section. Each list, therefore, contains the entire text of the object. One list, for example, comprises the text divided into chapters; another comprises the text divided into paragraphs. The text contained within a single list will not overlap; however, sections from two different lists may have portions that overlap. An inverted index is used to match keywords to the sections where the keyword is found (Burkowski, 1992).

2.1.3.2 Proximal Nodes

The structural model based on proximal nodes was developed by Navarro and Baeza-Yates and is a variant of the non-overlapping lists model. This model divides the text into hierarchical lists. Each list contains only a section of the text and its subsections. Where a non-overlapping list contains the entire text divided into chapters, a list based on proximal nodes contains one chapter, that chapter divided into paragraphs, and those paragraphs divided into sentences. Thus, the non-overlapping lists are referred to as flat structures and proximal node lists are often called hierarchical structures. Like non-overlapping lists, different proximal node lists may point to overlapping sections (Navarro, 1997 and Baeza-Yates, 1996).

2.1.4 Retrieval Processes

To measure the performance of IR solutions—and to better understand their strengths and weaknesses—it is important to understand the elements that comprise information retrieval. A complete evaluation of an IR solution requires consideration of each of the three retrieval steps, including the parsing and indexing of an information collection, the interpretation of user-submitted queries, and the retrieval of relevant information (Croft, 1992).

2.1.4.1 Index

The three most common indexing techniques create either an inverted index, a suffix array, or one or more signature files. An inverted index contains a list of all of the words in an object and each word's position in that object. Inverted indices are the most popular technique used today. Suffix arrays are more efficient versions of suffix trees; both contain pointers to the text suffixes contained in an object. Suffix arrays are generally used for more complex searching (i.e., phrase search) of objects that are not entirely text-based (i.e., genetic databases). A signature file creates a hash for each word in an object. Signature files require minimal overhead and are more suitable for smaller-sized objects.

Increasing storage capacities allow retrieval solutions to take advantage of full-text indexing—where each word in an object is indexed. Where large collections make this less feasible, retrieval solutions may take advantage of one or more indexing operations in order to reduce the index size. Five popular indexing operations include: lexical analysis, elimination of stopwords, stemming, keyword selection, and thesauri matching. Lexical analysis is the process of converting characters into text. In addition

to the recognition of spaces, this process must decide how to handle numbers, hyphens, punctuation marks, and the case of letters. Strict character removal is not always the solution; these operations are simple to implement, but must be done with care (Fox, 1992).

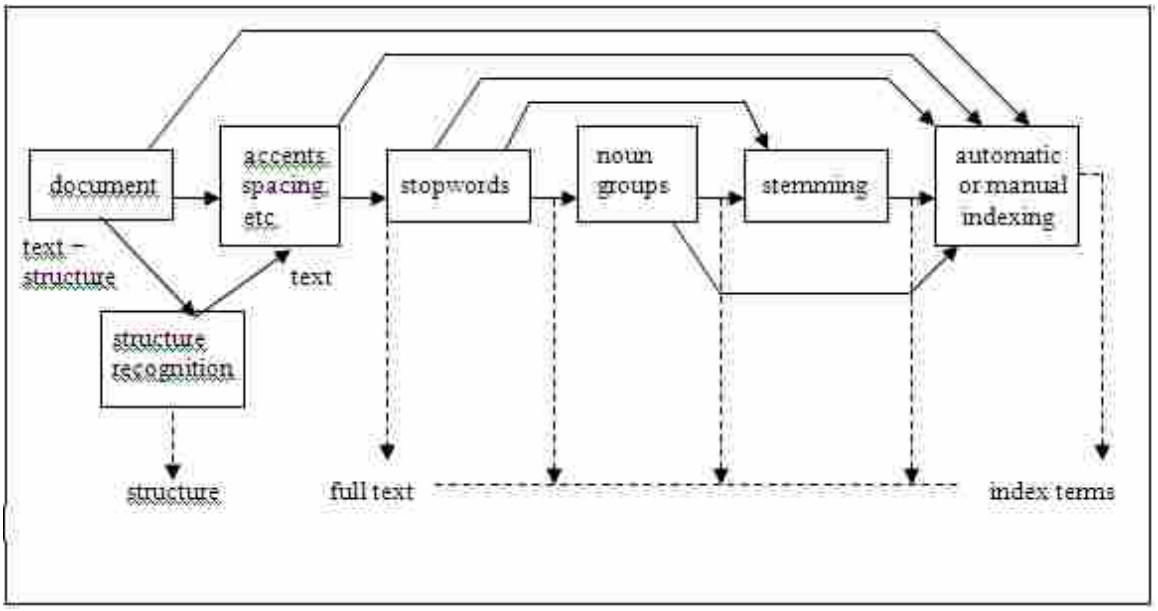

Figure 2.1. Logical View of the Indexing Process. Adapted from (Baeza-Yates, 1999).

The elimination of stopwords—like "the" and "is"— can reduce an index's size by 40% or more. Words that occur in 80% of objects in a collection are considered of no value for differentiating between objects and are also eliminated. Stopword elimination; however, introduces a new problem. The phrase "To be or not to be," for example, would be reduced to "be be" or, in some cases, eliminated all together. A search for this phrase would return no matches. As with lexical analysis, stopword removal requires thoughtful consideration before implementation (Baeza, 1999).

The creation of noun groups attempts to index only the most important words in an object. The theory behind noun groups is that nouns generally carry most of the semantic meaning of a sentence. Sometimes, as is the case with the term "information retrieval," a noun is actually a group of nouns. Because of this, noun groups are created when multiple nouns are syntactically close to each other.

Stemming is the process of replacing syntactical variations of a word with its stem, or root. Index size is mitigated because the word "run" would represent "runner" and "running." Frakes (1992) performed several experiments on various stemming implementations and; although expected to reduce index size and increase search performance, results regarding the benefits of stemming were inconclusive. The drawbacks have been valid enough to prevent most Web search engines from adopting stemming algorithms (Baeza, 1999).

Following the completion of the text operations, the actual indexing occurs. This indexing may be done manually or automatically. Manual indexing—popular in the 1960s—required trained indexers to assign keywords to an object. Today, for the most part, these same tasks are performed by machines.

Where index size is not an issue, thesauri matching can be used. Historically, keywords were part of a controlled vocabulary—a user needed to be familiar with the terminology in order to perform effective searches. As a result, searching for information was a highly skilled occupation. A thesaurus simplifies the searching task and allows a user to identify appropriate search terms from an established vocabulary set. A thesaurus may also be used during indexing. The idea is that synonyms and similar phrases would retrieve the desired objects; no controlled vocabulary would be needed. Using thesauri

matching during the indexing process is still a source of debate among retrieval researchers—and its lack of implementation is proof that developers aren't convinced (Baeza, 1999).

2.1.4.2 Query

The techniques employed during query processing are usually dependant upon and identical to those implemented during the indexing process. As a result, the techniques are the same: stopword elimination, noun group creation, stemming, and the use of synonyms. Similarly, the types of queries available are also defined by the retrieval model. The three types of queries include keyword-based, pattern matching, and structural queries. Keyword-based queries are intuitive, easy to use, and include single word, context (phrase and proximity), Boolean and natural language queries. Pattern matching queries search for objects containing a specific pattern (i.e., a DNA sequence). Structural queries allow for searches of context in relation to object structure (i.e., the location of words or phrases in relation to chapters, paragraphs, etc).

2.1.4.3 Match

When a query is submitted, the matching process searches for documents containing relevant content. Relevancy is defined by the retrieval model and, as a result, it varies among retrieval solutions. The Boolean model, for example, is based on Set Theory; a relevant object must lie within the intersection of the sets defined by the query. The boundaries of the sets are rigid. Therefore, an object containing 99 of 100 of the query terms will not be counted as relevant. Other models relax this rigidity. The vector space model, as explained by Rilloff (1994), "views each document and query as a vector

in an *N*-dimensional space, where *N* is the number of relevant terms in the database. The query vector is compared to all of the document vectors using a similarity metric." In this way, partial matches are included in the answer set. Partial matching creates varying degrees of relevance and is often accompanied by the calculation of a document weight. These values are then used to present matched documents to the user in order beginning with the most relevant.

In addition to content, a retrieval solution may also consider meta-data. For example, an object published within the last ten years may be deemed more relevant than one published fifty years ago. In an online environment, an object's popularity may be considered when calculating its relevance, i.e., pages with higher traffic are considered more relevant. This is a key component of the matching algorithm employed by Google and other web-based search engines (Brin, 1998). Ranking algorithms are especially critical in an environment as large and unknowable as the web. Because this research deals with specialized document collections, page ranking capabilities are not included in the list of measured characteristics.

2.2 Information Retrieval Evaluation

There are various measures that aid in measuring the performance of an information retrieval solution. These measures may be categorized as subjective, objective, or user-oriented measures. Subjective measures to establish the definition of a good result set and then use that definition to evaluate the actual results. The definition is usually created as experts review the contents of an object collection and classify each object as relevant or not. Throughout this research, subjective measures are those that require a degree of human interaction in order to have an "answer key." User-oriented

measures evaluate the user's experience while interacting with a system. Objective measures are used to determine more concrete statistics: speed, size, etc.

2.2.1 Subjective Measures

Of the many measures published today, precision and recall are the most well-known. It has been shown; however, that these measures have inherent weaknesses. As a result, other measures have been developed and include the precision histogram, summary table statistics, mean-average precision, F-measure, and E-measure. Still, these measures rely on the calculation of Precision and Recall. As will be explained, this is problematic when dealing with an unknown data collection. A list of subjective measures and their ideal values are displayed in Table 2.1.

Table 2.1. Subjective Measures.

Desirable Characteristics	**IDEAL**
Precision	1
Recall	1
Precision Histogram	N/A
Table Summary Statistics	N/A
Mean-Average Precision	1
F-Measure	1
E-Measure	1

2.2.1.1 Precision

To calculate Precision, an absolute truth must be known—a user must know which documents are relevant and which documents are not. However, the absolute truth is a moving target and will be different based on the retrieval strategies implemented by the IR solution. For example, if a user searches for the word "hat" and a lexical analysis

is not employed, the system will retrieve all objects containing the word hat. However, if a lexical analysis is enabled, the system will also retrieve all objects containing the words "hats," "that" and "shatter." Whether or not the retrieved objects are indeed relevant will depend upon the application. A fashion designer my not be interested in the "shatter" objects, however, a legal compliance officer may be interested in all of the retrieved objects. The inability to accurately calculate a Precision score impacts the calculation of other subjective- and user-oriented-measures that include this measure in their calculations.

Precision (P) measures the rate at which a system retrieves non-relevant objects from a collection. It is measured as follows:

$$P = \frac{\textit{number of relevant documents retrieved}}{\textit{number of documents retrieved}} \qquad (2.1)$$

A result set of ten objects that contains only three relevant objects would have a precision of .3. Ideal precision is 1. Low precision scores are prevalent among retrieval solutions because of the ambiguity of natural language. For example, most retrieval solutions are unable to successfully index words that have multiple meanings.

2.2.1.2 Recall

Recall is of particular interest to applications that need to comply with federal regulations. A search must return all relevant objects. To accurately measure recall, it is necessary to have an excellent familiarity of the object collection. As the size of a collection grows, this becomes increasingly difficult. The task becomes near-impossible

when dealing with an unknown data collection, like the internet. The inability to accurately calculate a Recall score impacts the calculation of other subjective- and user-oriented-measures that include this measure in their calculations.

Recall (R) measures the rate at which a system retrieves all relevant objects from a collection. It is measured as follows:

$$R = \frac{\textit{number of relevant documents retrieved}}{\textit{number of relevant documents}} \qquad (2.2)$$

A system whose result contained three relevant objects from a collection of 10 relevant objects would have a recall of .3. Ideal recall is 1.

2.2.1.3 Precision Histogram

Precision histograms are useful for comparing search performance of two competing algorithms. The data is computed by finding the difference between the two precision scores. A score of zero indicates the algorithms were identical. Any other score indicates a winner.

2.2.1.4 Summary Table Statistics

A summary table could be used to store the details of a given query task including the number of queries, number of objects retrieved by all queries, total number of relevant objects retrieved by all queries, the number of relevant objects that *could* have been retrieved, etc.

2.2.1.5 Mean Average Precision

Mean average precision, or average precision, averages the precision values measured as new relevant objects are retrieved. This measure ranks systems that return more relevant objects sooner higher than systems that don't. Two systems, for example each return answer sets of 100 objects and both contain 10 relevant objects. According to the mean average precision, the system that found the relevant objects sooner would have a higher score.

2.2.1.6 F-Measure

The F-measure, or harmonic mean, represents a weighted harmonic mean of precision and recall. It is measured as follows:

$$F = 2 / (1 / precision) + (1 / recall) \quad (2.3)$$

The F-Measure gives an equal weighting to precision and recall.

2.2.1.7 E-Measure

Van Rijsbergen proposed another measure, the E-measure, which allows differential weighting of recall and precision. It is measured as follows:

$$E = 1 - ((1 + b^2) / ((b^2 / recall) + (1 / precision))) \quad (2.4)$$

b is a user-specified parameter to assign greater weight to one of the two values. A *b* value greater than 1 implies that precision is more important; a value less than one

implies that recall is more important. This equation is particularly valuable to companies, for example, interested in legal compliance where recall is far more critical than precision.

2.2.2 User-oriented measures

One of the weaknesses of the recall and precision is that they require a thorough knowledge of the collection being searched. Especially in filtering systems—where the collection changes constantly—this is a daunting and near-impossible task. As a result, user-oriented measures were developed and include coverage, novelty, relative recall, and recall effort (Korfhage, 1997).

Table 2.2. User-Oriented Measures.

Desirable Characteristics	**Ideal**
Coverage	1
Novelty	Varies
Relative Recall	Varies
Recall Effort	Varies
Normalized Recall	1
Satisfaction	High
Frustration	0

2.2.2.1 Coverage

Coverage represents the number of retrieved relevant objects compared to number of relevant objects the user knows to be in the collection. It is measured as follows:

$$C = \frac{\textit{number of relevant documents retrieved}}{\textit{number of relevant documents known to be in collection}} \quad (2.5)$$

2.2.2.2 Novelty

Novelty represents the number of retrieved relevant objects that were previously unknown to the user. It is measured as follows:

$$N = \frac{\text{number of unknown relevant documents retrieved}}{\text{number of relevant documents retrieved}} \qquad (2.6)$$

Novelty measures the exact property that makes recall and precision inaccurate. Where novelty is high, recall and precision, as defined above, will be hard to calculate. As such, novelty is more a measure of a user's familiarity with a collection rather than of actual retrieval performance.

2.2.2.3 Relative Recall

Relative recall represents the number of relevant objects found in relation to the number of objects the user expected to find. The number of objects expected to be found will vary from user to user. In some cases, 100 objects will be too few, however in others, 100 will be too many.

2.2.2.4 Recall Effort

Recall effort represents the number of relative objects the user expected to find in relation to the number of objects searched in order to find as many. Like novelty and relative recall, recall effort is not so much a measure of retrieval performance as it is a reflection of the user's familiarity with the object collection. Recall effort will be higher for a large diversified collection than for a small restricted collection.

2.2.2.5 Satisfaction

Satisfaction represents a solution's ability to present relevant objects first. Ideally, each of the first objects presented to a user are relevant.

2.2.2.6 Frustration

Frustration represents the opposite of satisfaction: it represents a solution's inability to present relevant objects first. Users who find that the answer set contains irrelevant objects will either attempt a new search or become frustrated and stop using the system. A common measure is equal to the combination of satisfaction and frustration.

2.2.2.7 Normalized Recall

Normalized recall represents how well a retrieval solution find and returns relevant objects from a collection. It is measured as follows:

$$Normalized\ recall = 1 - \frac{differential\ area}{(O_R * (C * O_R))} \qquad (2.7)$$

where O_R represents the number of relevant objects in the collection and C is the size of the collection. An ideal retrieval solution retrieves all relevant objects and lists those objects before listing non-relevant objects. Thus, if a collection contains ten relevant objects, the first ten objects in an answer set would comprise the relevant objects. Normalized recall requires an extensive knowledge of the object collection and places significant importance on the last relevant object listed.

2.2.3 Objective measures

Like subjective measures, user-oriented measures require a thorough knowledge of the collection being searched. Even more than subjective measures, user-oriented measures struggle to compare retrieval solutions due to their dependence on a user's behavior and knowledge. For these reasons, objective measures are superior where the overall performance of a retrieval solution is more important than the answer set. This is true for web search engines: retrieving relevant information is critical; however, a slow product will not be used. Just as CPUs are compared by cycles per second and network connections by bits per second, retrieval solutions have objective characteristics that can be measured and used to compare competing retrieval solutions, including the time

Table 2.3. Objective Measures.

Desirable Characteristics	**IDEAL**
Time to index repository	Fast
Time to update repository	Fast
Time to execute queries	Fast
CPU Utilization	Low
Memory Utilization	Low
Hard disk space required	Low

required to index an object collection, the resources required to keep an index up to date, and the time required to execute a set of queries. Beyond measuring the time required to perform a number of tasks, objective measures are used to predict the scalability of a retrieval solution as well as retrieval accuracy, and robustness.

2.2.3.1 Time to Index, Update and Query

A review of retrieval models makes it clear that the time to index an object collection will vary greatly. As a standalone measure, this characteristic can be misleading. For example, a more complex index may require more time to create, but may also be able to process more complex queries and offer more functionality to users. Additionally, static object collections will generally be indexed just one time; the time required to generate an index alone should not lead to a premature conclusion.

Where dynamic object collections are concerned, the ability to quickly generate and maintain an index is more critical. Specifically, this process involves reviewing the object collection for the appearance new or modified objects and, when any are discovered, analyzing the changes and updating the index.

Similarly, the ability to process queries quickly is a key in retrieval performance. And like the time required to index, if used as a standalone measure, it may be misleading. For example, a complex queries may require more time to execute, but may offer more powerful search criteria.

2.2.3.2 Scalability

Tracking the objective performance of a retrieval solution through a series of tests involving different sized collections will help predict the scalability of a retrieval solution. Performance trends can be identified as constant, linear, exponential, etc.

2.2.3.3 Accuracy

Subjective measures calculate retrieval accuracy in a number of ways, but most require knowing how many relevant objects exist in the collection. Contrastingly, the

objective accuracy is more a measure of the ability to correctly interpret object content. For example, a hyphen located at the end of a sentence should be handled differently than a hyphen located in the middle of a line.

2.2.3.4 Robustness

The type and quantity of data in the collection will greatly influence the results. A retrieval solution may perform well in one environment while performing poorly in another. While a good retrieval solution may perform well whether dealing with medical records, journal articles, or e-mail, the best solution may be specifically optimized for one or the other. Indexing techniques, for example, should be different when indexing electronic books as opposed to Instant Message conversations. For example, Lee (1994) showed that signature files outperformed an inverted index system when manipulating a collection comprising e-mail-sized objects.

2.3 Review of Literature Conclusions

The demand for high performance retrieval solutions is growing at a phenomenal rate. Several models and strategies have been developed to improve the basic retrieval techniques introduced over fifty years ago. As the amount of information continues to increase, retrieval solution performance will become more critical to user-satisfaction. The ability of a retrieval solution to execute commands quickly is critical. If overall performance is slow, it will most likely not be used.

Before implementing an IR solution in any environment, its performance must be evaluated and compared to other IR solutions. One of the weaknesses of popular subjective measures is that they require a thorough knowledge of the information

collection being searched. Especially in a filtered environment—where the collection changes constantly—this is a near-impossible task. Additionally, experts often disagree on object's relevance; defining an ideal answer set is often a debated process. Furthermore, the usefulness of user-oriented measures is determined by the application and environment wherein a retrieval solution is implemented.

There is need for an infrastructure designed to objectively measure the performance of competing retrieval solutions. The ability to quickly index a document collection, to keep the index as current as possible, and to retrieve relevant information are important characteristics to evaluate. Other important factors that contribute to a proper evaluation model include retrieval accuracy, legal compliance, resource utilization, and scalability. Such a utility will aide developers in the pursuit of better retrieval strategies and assist IT professionals in the selection of a retrieval solution that best satisfies a customer's needs.

Chapter 3

3 RESEARCH PROCEDURES

This research comprises two main components: the evaluation and comparison of the performance of information retrieval solutions and the infrastructure responsible for the administration of that analysis. This chapter describes the details of both. The first section specifies the hardware and software configuration required for the testing environment. Specifically, it details the operating environment of the evaluation infrastructure, the creation of the object collections, the installation and modification of retrieval solutions, and the design of the run control. The second section describes the tests that are performed, including the characteristics to be evaluated and how measurements are taken. An early version of this research was reported in 2006 (Broadbent, 2006).

3.1 Software and Hardware Configuration

The testing infrastructure was designed to be platform independent. Although specific operating systems and hardware components are set forth below, these details should not be seen as restrictions imposed by the infrastructure. Three components are necessary to accurately measure retrieval performance: a run control, information retrieval software, and one or more object collections. The run control is responsible for pre-test initialization, log creation and post-test clean-up. The information retrieval

software is configured to execute retrieval processes. Those processes are executed against a set of object collections. An illustration of the interaction between these components is displayed in Figure 3.1.

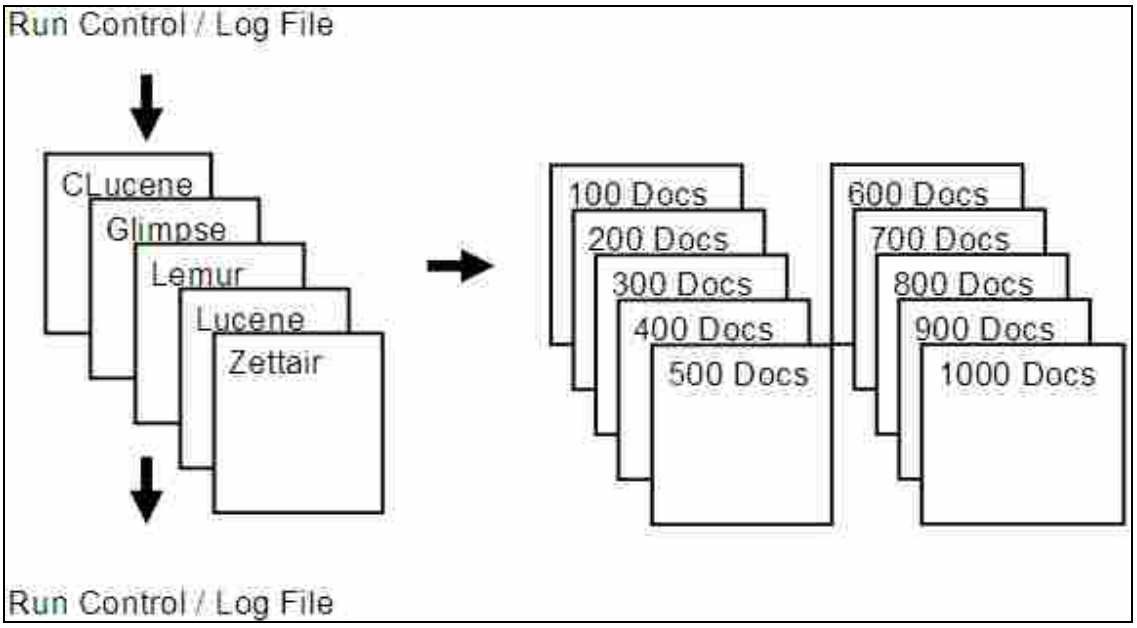

Figure 3.1. Evaluation Model.

3.1.1 Hardware Configuration

To measure the performance of retrieval solutions, it is possible to use any computer capable of running the retrieval software. All of the research for this thesis was conducted on a Toshiba Satellite featuring an Intel T2300 1.66GHz processor and .99GB of RAM. The hard drive, a Toshiba MK1032GSX, has a formatted capacity of 100GB on two disks which spin at 5400 rpm. The internal data transfer rate ranges between 236 and 456MB/s; the buffer-to-host transfer rate is 150MB/s. The average seek time is 12ms.

3.1.2 Operating System Configuration

The operating system is Microsoft Windows XP (version 5.1, build 2600.xpsp_sp2_qfe.070227-2300: Service Pack 2). VMWare Player (version 2.0.0, build-45731) was installed. A Virtual Machine was built utilizing the Debian- Etch 4.0-r0 operating system including the Full Desktop Gnome System. The file system is ext3. All of the information retrieval software and object collections were then installed on the Virtual Machine. A standard installation was performed. To accommodate the variety of programming languages used to author the retrieval solutions, a number of compilers are installed on the server, including GCC version 4.0.1 for C++ and JDK version 1.5.0 for Java. Sun's JVM is installed with default parameters left unchanged.

3.1.3 Object Collections

Eleven unique object collections of varying sizes were created for this research. Retrieval performance measured against each of ten primary object collections will demonstrate the scalability of a retrieval solution. An eleventh secondary collection consisted of a single document designed to allow the testing of specific problematic language patterns. Details pertaining to the eleventh object collection are explained in section 3.2.7.

To populate the ten primary object collections, a mirror of Project Gutenberg was established. In 1971, with $100,000,000 of computer time on the Xerox Sigma V mainframe at the University of Illinois, Hart concluded that "the greatest value created by computers would not be computing, but would be the storage, retrieval, and searching of what was stored in our libraries" (Hart, 1992). Project Gutenberg has since become the

largest single collection of free electronic books (e-Books) comprising over 100,000 files, more than 250 GB of disk space, and growing daily.

Each file in Project Gutenberg is assigned a unique e-Book number and archived accordingly. For example, e-Book 12,345 is named 12345.txt and located in the folder structure: /1/2/3/4/12345/. The project supports several file formats (i.e., plain text, MP3, HTML, PDF, RTF, XML, ZIP, etc.). For this research, sub-collections were extracted from the project. These smaller collections contained a varying number of randomly selected plain text files—all other file formats were ignored. To minimize performance differences due to traversing the hierarchical folder structure of the Gutenberg Project, the files were stored in the local file system in a single folder named for the number of files contained therein. The details of the sub-collections are available in Table 3.1. Once created, the collections remained unchanged throughout this research.

Table 3.1. Collection Details.

Name	Number of Files	Disk Space (MB)	Average File Size (KB)	Median File Size (KB)
100Docs	100	27.736	224.205	277.032
200Docs	200	97.8	360.661	494.162
300Docs	300	132.024	348.330	444.690
400Docs	400	165.420	328.116	417.198
500Docs	500	199.864	256.245	402.947
600Docs	600	248.368	350.297	417.683
700Docs	700	286.608	342.998	410.053
800Docs	800	311.952	315.502	393.134
900Docs	900	334.952	298.324	374.995
1000Docs	1000	382.244	312.805	385.280

3.1.4 Information Retrieval Solutions

There are several retrieval solutions available today. For this research, only open-source solutions were considered. Five retrieval solutions were selected for evaluation: CLucene version 0.9.16a, Glimpse version 4.18.5, Lemur version 4.5, Lucene version 2.1.0, and Zettair version 0.9.3. CLucene is a C++ port of Lucene; it is essentially a C++ translation of Lucene-1.4.3. Glimpse is a highly extensible retrieval solution authored in C++. Unlike other retrieval solutions, Glimpse offers the ability to designate different levels of indexing. A user may select to build a tiny index, a small index, or a medium index. The default level, the tiny index, was used in this research. Lemur is a self-described information retrieval toolkit authored in C and C++. Lucene is a popular, open source, customizable, java-based full-text retrieval system. Authored in C, Zettair—formerly named Lucy—is the final retrieval solution evaluated in this research. Except for small modifications—explained below—each retrieval solution was downloaded and compiled in accordance with accompanying documentation.

The first step of the evaluation process is to identify the functionality required by an application. Once a set of desired features is defined, the retrieval solutions containing matching that set of characteristics should be subjected to the tests outlined in this research. Significant differences in performances can indicate differences in the functional characteristics of retrieval solutions. The retrieval solutions in this research exhibit these functional differences and, as a result, are expected to reflect the differences in the performance evaluation.

As delivered, the retrieval solutions varied in the tasks performed during the execution of the retrieval processes targeted by the evaluation environment. For this

research, efforts were taken to minimize performance variations due to the handling of input, output, and log messages. To this end, the source for each retrieval solution was modified as follows.

3.1.4.1 CLucene

The delivered Main.cpp file was significantly modified. The original file presented a series of prompts, requiring the user to enter the location of the object collection, the location wherein to build the index, etc. The modified file allowed for these parameters to be passed in automatically. Additional code designed to record the time spent creating the index or executing a query was commented out. The modified file is included in Appendix A.

The delivered IndexFiles.cpp file was modified and saved as Index.cpp. All commands to print to the screen were commented out. Code designed to measure the time required to index the collection was also commented out. The resulting program created a new instance of an IndexWriter, created an index for an object collection, and terminated. The modified file is included in Appendix B.

The delivered SearchFiles.cpp file was modified. All of the code designed to format the results of the query was commented out. Additionally, code designed to measure and the display the time required to execute the query was disabled. The resulting program created a new instance of an IndexSearcher, created a query, printed the query parameters, executed the query, printed the number of matching documents, and terminated. The modified file, named SearchFiles.cpp, is included in Appendix C.

3.1.4.2 Glimpse

Like the other retrieval solutions, Glimpse required some customization. Throughout the indexing and searching processes, there were a number of prompts that were eliminated. Glimpse also performed a number of calculation before indexing a collection, including counting the number of files in the collection and calculating the size of the collection. These operations were bypassed to create an operational environment similar to the other retrieval solution mentioned above. Records of the modified files are included in Appendices D and E.

3.1.4.3 Lemur

Like the other retrieval solutions, Lemur required some customization. Two files were modified to normalize behavior: IndriBuildIndex.cpp and IndriRunQuery.cpp. These modified files are included in Appendices F and G.

3.1.4.4 Lucene

Index.java was copied from the delivered IndexFiles.java file. It was modified to accept two input parameters: the location of the object collection to be indexed and the location of the directory wherein to build the index. All commands to print to the screen were commented out. The resulting program created a new instance of an IndexWriter, created an index for an object collection, and terminated. By design, Lucene only indexes the first five kB of data in each file. This limit was removed so that the entire file would be indexed. Index.java was compiled and added to the delivered lucene-demos-2.1.0.jar file. The modified file is included in Appendix H.

Search.java was copied from the delivered SearchFiles.java file. It was modified to accept two input parameters: the location of the index and the query parameters. With the exception of two, all commands to print to the screen were commented out. The program still prints the query parameters and the number of matching documents found. Code designed to format the result set was also commented out. The resulting program created a new instance of an IndexReader, created a query, printed the query parameters, executed the query, printed the number of matching documents, and terminated. Search.java was compiled and added to the delivered lucene-demos-2.1.0.jar file. The modified file is included in Appendix I.

3.1.4.4 Zettair

Zettair was installed and run as delivered. In this configuration, output to the screen is minimal and was ignored. Like the other modified solutions, Zettair displayed the number of documents for which a match was identified.

3.1.5 Run Control

The Run Control is a PERL script designed to uniformly and accurately administer all of the aspects of the tests. It is written in PERL to minimize resource consumption during the execution of tests. PERL script execution is available in the standard installation of Debian-Etch. For each test, the Run Control is responsible for system initialization, recording all of performance characteristics, and deleting temporary files created during execution.

The Run Control initiates each test run. The initialization process is identical for each test run. In every case, the Run Control initiates a simple command for both the

Java and C++ run time environments. The command passes control from the Run Control to the respective compiler for execution of the command. Once complete, control returns to the Run Control. The duration of that sequence is recorded for each command. The process is repeated ten times. A copy of the run control is included in Appendix J.

The Run Control is responsible for measuring and recording all monitored characteristics. For each test, the controller will record the start and finish times for the execution of the given command. Additionally, the size of the index file(s) is recorded whenever an index is created or updated and, during searches, the Run Control records the number of matching documents returned by the retrieval solution. After each iteration, the Run Control deletes any and all files created during execution. All statistics for the test are committed to log files.

3.2 Measured Characteristics

This research focuses on the evaluation and comparison of retrieval performance. The evaluation is performed by subjecting each retrieval solution to a number of tests. Multiple iterations of each test are executed and a 20%-trimmed mean is calculated. Specifically, the evaluation includes the following six measures:

1) The time required to index an object collection; calculated as the time that elapses during the creation of an index for an object collection,

2) The time required to execute a set of queries against an object collection; calculated as the time that elapses during the execution of a set of queries against an object collection,

3) The time required to maintain an index; calculated as the time required to rebuild an existing index based on updates to objects or the addition of new objects to an object collection,

4) The size of the index file(s) created for an object collection,

5) The number of matching documents retrieved, and

6) The ability to successfully interpret problematic language patterns.

Table 3.2. Characteristics to Measure.

Type		Name	Target
Objective Measures	Time-Based Measures	Index Time	time elapsed during creation of an index for a collection
		Query Time	time elapsed during execution of a query or set of queries
		Maintenance Time	time elapsed during recreation of index after changes are introduced into the collection
		Measured Scalability	rate of performance increase due to increased collection size
	Quantitative Measures	Disk Space	size of a system's index created for a collection
		Matches	the number of matching documents retrieved
	Qualitative Measures	Retrieval Accuracy	Number of problematic patterns successfully retrieved

These may be further categorized as time-based, quantitative, and qualitative measures. An eighth measure, predicted scalability, may be inferred based on performance trends observed during the execution of retrieval processes in relation to the

increase in size of the object collection. Finally, a method for calculating the precision and recall values is set forth.

To compare retrieval performance, the results for each performance area are ordered from best to worst. Points are assigned based on rank and the total number of retrieval solutions being tested. Each characteristic is assigned a characteristic weight. This value allows more or less emphasis to be placed on a retrieval characteristic. For example, if an environment features a relatively static document collection, the ability to quickly maintain an index will be less important than the ability to execute queries. In this instance, index maintenance could be assigned a characteristic weight of zero and query execution could be assigned a characteristic weight of two. A characteristic weight can be any value. This research assigns a value of one to each characteristic. This research compares five retrieval solutions; first place is awarded five points, second place is awarded four points, and so on. The awarding of points is expressed as:

$$\text{Points Awarded} = \left(\sum_{i=1}^{i+d-1} ((c - i + 1))/d\right)w \tag{3.1}$$

where c is the number of points awarded, d is the number of retrieval solutions being evaluated, and w is the characteristic weight.

3.2.1 *n%*-Trimmed Mean Calculation

Depending on age, installed software, and other factors, a computer may take anywhere from thirty seconds to several minutes to "boot-up." Similarly, software applications in various programming languages have significant start-up delays. The

same is true for programming languages. Every programming language has a distinct initialization requirement. During this time, libraries are loaded, objects are initialized, etc. Higher-level languages, like Java, require more overhead than a lower-level language, like C, and will therefore require more time to "start-up." Because of the approach to time measurement adopted by this research, longer start up times have the potential to significantly degrade a retrieval solution's performance. However, if a program is executed a second time, most of the libraries, objects, etc. will still be cached—the effects of the start-up time the second time will be less pronounced. If a program is executed a third time, the start-up time is even shorter. Consequently, it is anticipated that measured performance will improve with each iteration of a given test. To compensate, an *n%*-trimmed mean is calculated instead of a mean. The trimmed mean is less sensitive to extreme observations, which are expected to be found during the first few test runs for each retrieval solution.

For a set of sorted numbers, an *n%*-trimmed mean removes—or trims—a percentage of both the smallest and largest numbers from the set; the remaining values are averaged together. For example, to calculate a *20%*-trimmed mean on the data set in Table 3.3, the values are first sorted from smallest to largest—or from slowest to fastest, etc. The top *20%* and the bottom *20%* are discarded. In the example there are ten data values; the two smallest and the two largest values are discarded. A mean is calculated for the remaining values. For this research, each test was run ten times. For each set, the two best times and the two worst times were discarded; *n* was 20%. Table 3.3 shows the difference between the mean and *20%*-trimmed mean for the same data set.

Table 3.3. Calculation of 20%-Trimmed Mean.

Original Data	Data After Sort	Data After Discard	20%-Trimmed Mean	Mean
89	20		37.8	42.7
68	23			
49	23	23		
23	32	32		
45	35	35		
20	43	43		
43	45	45		
32	49	49		
35	68			
23	89			

3.2.2 Index Creation

The time to index includes all initialization procedures, the analysis and parsing of data objects, and the creating and writing of the index file(s). An arbitrary timeout equal to the number of documents in the collection is observed. For example, the timeout for the 100-document collection is 100 minutes; 200 minutes for the 200-document collection, and so forth. If the timeout is reached, current processing is halted, no time is reported for the current iteration, and the run control proceeds to the next step.

3.2.3 Query Execution

This research defines the time to query to include all initialization procedures and the time required to return an answer to a query. Query execution occurs twice during the evaluation: once after the creation of a new index, and again after the maintenance of an existing index. For each query, the number of matching objects was recorded.

3.2.4 Index Maintenance

Incremental indexing is more common than complete indexing. When incremental indexing is employed, an object collection needs to be completely indexed just one time. Thereafter, only changes to the object collection are re-indexed. This research defines the time to maintain an index as the time required to re-index an object collection after changes are introduced. To calculate this, an object collection is indexed. Then, a new set of objects is inserted into the object collection. The start time is recorded. The maintenance time includes all initialization procedures, the analysis and parsing of data objects, and the creating and writing of the new index file(s). As with the indexing process, an arbitrary timeout equal to the number of documents in the collection is observed. If the timeout is reached, current processing is halted, no time is reported for the current iteration, and the run control proceeds to the next step.

3.2.5 Index Size

The index size is recorded after index creation and index maintenance. While disk space is less of an issue today than it was fifty years ago, the size of an index may provide interesting insight into retrieval performance. For example, larger indices may require more time during creation or query execution. A larger index may contain more useful data, and may grow proportionally—or disproportionately—to the size of the object collection being indexed. A copy of the index is stored for future reference.

3.2.6 Matches

The number of documents retrieved during the search process is recorded. This measure is classified as a quantitative measure because the infrastructure has no way to judge whether the documents retrieved are relevant to the search criteria.

3.2.7 Qualitative Assessment

This qualitative assessment is critical to the analysis of information retrieval solutions. While it is important to develop an infrastructure to measure and compare how quickly information retrieval tasks are performed, the number one priority in Information Retrieval is the quality of results. No matter how fast an IR solution may be, it will not be used if it cannot produce accurate results.

One of the challenges facing the ongoing development of retrieval strategies is the handling of problematic language patterns. The list of such patterns is lengthy and includes hyphenations, acronyms, abbreviations, numbers and capitalizations. The measurement of retrieval accuracy will assess how well a retrieval solution is able to index, search for, and process problematic language patterns. To do this, an eleventh document collection was created. The collection contained a single document. The document was intentionally loaded with specific examples of problematic language patterns. Then, identical queries were created and executed by each IR solution. For example, to test for the ability to find abbreviations, the abbreviation "IR" was inserted into the document. A query was defined with search criteria equal to the same abbreviation. A copy of this document is included in Appendix K.

Because there is just one document in the collection, a match indicates the pattern is detected in the special document. No match indicates the pattern was not detected.

Additionally, this approach enables the establishment of an absolute truth. It is possible to know how many relevant documents are contained within the information collection and it's possible to know if the document retrieved is relevant to the information need. More complicated measures, like Precision and Recall, may then be calculated.

3.2.8 Scalability

The results of each time-based test are plotted on a graph where the horizontal axis represents the size of the object collection and the vertical axis represents the time required to perform the given operation. Once all of the results are plotted, the performance of each retrieval solution will display as a line on the graph. It is anticipated that each line will have a positive slope, as shown in Figure 3.2.

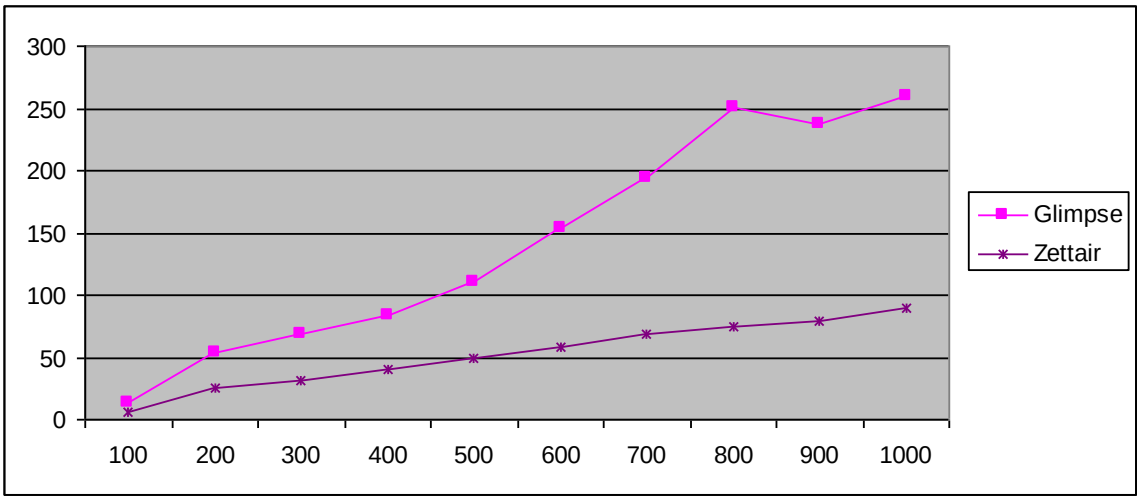

Figure 3.2. Measured Scalability

The increase in processing time in relation to the size of the object collection—or slope—will indicate the performance trend—or measured scalability—for the retrieval solution. From this, a performance estimate may be inferred for larger collections. In terms of

scalability, the smaller the vertical gain, the better. In Figure 3.2, Zettair is a more scalable solution than Glimpse.

3.2.9 Adjustments

The time-based measures measure the time elapsed during the execution of a set of retrieval process including the creation of an index for an object collection, execution of queries against an index, and the maintenance of an existing index. However, the actual measurement comprises three time periods summed together. During any given test, the Run Control records the current time and initiates the retrieval process. At that point, the retrieval solution begins executing the command. The end time is recorded when control is passed back to the Run Control. Hence, the time to index represents the time for control to be passed from the Run Control to the retrieval solution, the execution of the command, and the returning of control back to the Run Control (See Table 3.4).

Table 3.4. Definition of Elapsed Time.

Total Time Elapsed		
Control is passed to Retrieval Solution	**Execution of Command**	**Control is passed to Run Control**

A number of steps are taken before each test to achieve a consistent testing environment. The system is rebooted before each test. And no other programs are started after log in. A terminal session is opened and the command is given. Automatic software updates are disabled to maintain consistency throughout the testing.

3.3 Research Procedure Conclusions

An infrastructure for the performance evaluation and comparison of information retrieval solutions has been established. A series of programmatically-controlled tests will be conducted on two information retrieval solutions. The tests are configured in an effort to mitigate variation due to programming language and the handling of input and output. Extraneous operations to those of indexing, searching, and updating have been eliminated to ensure that performance differences may most likely be attributed to the execution of one or more information retrieval processes. The results are then used to compare competing retrieval solutions. This infrastructure is an effective tool for the comparison retrieval solutions, strategies, and custom enhancements. It is a universal infrastructure; able to accurately measure the retrieval performance regardless of model or language.

Chapter 4

4 RESULTS AND DATA ANALYSIS

This chapter presents the results from the tests described in chapter three. The characteristics are presented individually, however, it is important to note that an accurate evaluation of a single retrieval solution—or the comparison of two or more retrieval solutions—will often require taking all test results into consideration.

4.1 Index Creation Performance

Because the operating environments of retrieval solutions vary from application to application, it is incorrect to say that the goal during the indexing process is to create an index as quickly as possible. While some real-time applications may require near-instantaneous performance, other systems may have an entire night or weekend to perform indexing. Indeed, it may be more correct to assert that the goal during the indexing process is to create as thorough an index as is necessary in as reasonable an amount of time as is possible. The definition of reasonable would be established on a case-by-case basis.

To measure index creation performance, each retrieval solution created an index for each of ten object collections. This was repeated ten times. The total time, as described in chapter three, for each object collection can be seen in Table 4.1. The results are displayed in Figure 4.1. Complete results, including the times for each of the ten iterations, are included in Appendix L.

Table 4.1. Time to Index a Collection

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Time	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**
100	21.45	3	13.65	4	60.74	1	51.38	2	6.50	5
200	130.81	3	53.95	4			177.89	2	25.36	5
300	144.32	3	69.04	4			219.49	2	31.64	5
400	184.77	3	83.33	4			273.79	2	39.90	5
500			110.49	4					49.97	5
600	306.17	3	153.72	4			418.71	2	58.67	5
700	283.85	3	194.54	4			475.01	2	69.00	5
800	318.70	3	250.39	4			526.30	2	75.01	5
900	341.91	3	236.68	4			558.84	2	78.45	5
1000	466.67	3	259.05	4			694.90	2	88.88	5
Points		**27**		**40**		**1**		**18**		**50**

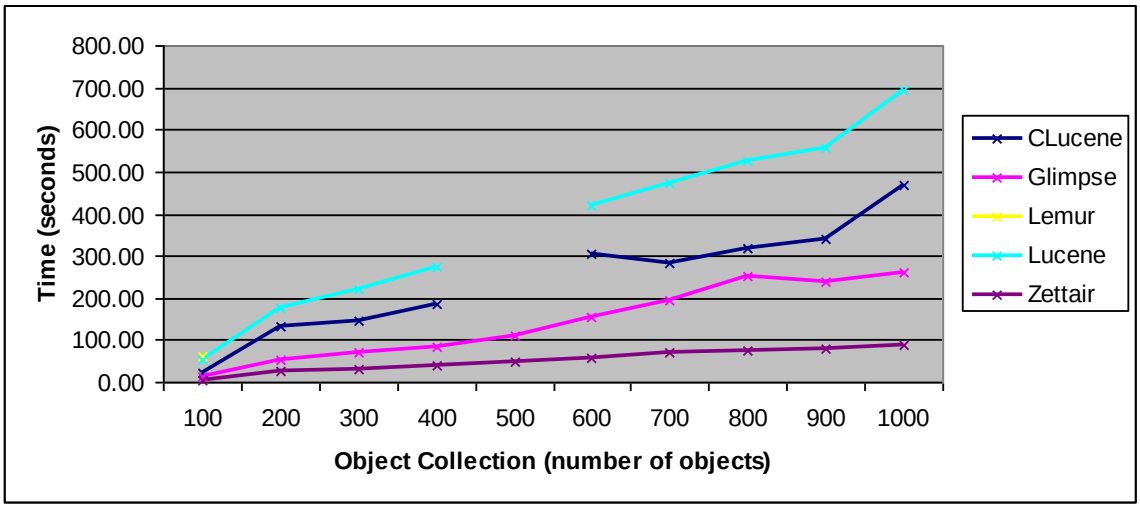

Figure 4.1. Time to Index a Collection

Table 4.1 indicates that both CLucene and Lucene failed to index the 500-document collection before the timeout limit was reached. Likewise, Lemur failed to index all but the first collection before the allowed time expired. Because no changes were made to the software before or after the test, the results of the subsequent tests were

incalculable. The inability to index the collection affected the tests that followed: because the index creation process failed, the search also failed. Similarly, because the initial index failed, the index maintenance process also failed, as will be seen in the sections to follow. For CLucene and Lucene, the failure was attributed to the large size of file 11800-8.txt. At 30MB, the file was nearly five times larger than the next largest file in all of the collections. CLucene and Lucene have built-in limits controlling the size of and the number of keywords in an index. These limits were exceeded, causing the processes to fail.

The Lemur solution did not complete the indexing process before the predetermined timeout. This failure was apparently due to the availability of system resources. While able to index all objects in all collections individually—as verified through later testing—the application was unable to index large quantities of objects at the same time.

The Glimpse solution also encountered files that it could not process correctly. However, in contrast to CLucene, Lemur and Lucene, these files were ignored, and processing continued as normal for the remaining files in the collection. Glimpse indexed all but two of the documents in the 500-document collection; all but one in the 600-document collection; all but two in the 700-document collection; all but two in the 900-document collection; and all but twelve documents in the 1000-document collection;. These files were excluded during the indexing process (and consequently, from all subsequent tests).

4.2 Query Execution Performance

Like the index creation process, the query execution process is typically focused on the quality of results—the duration of the process being a secondary concern. However, all else being equal, the fastest retrieval solution wins.

To measure query execution performance, each retrieval solution executed an identical query against each object collection, as described in chapter three. This was repeated ten times. This test was performed twice during the evaluation: once after index creation and again after index maintenance. The total time, representing a 20%-Trimmed Mean, for each object collection can be seen in Tables 4.2 and 4.3 and Figures 4.2 and 4.3. The number of matching documents for each query was also recorded. Complete results, including the times for each of the ten iterations, are included in Appendix L.

Table 4.4 and Figure 4.4 display the absolute value of the difference between the times recording during each search. A larger value in the figure represents that the search times were significantly different before and after index maintenance.

Table 4.2. Time to Perform First Search

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Time	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**
100	0.98	3	0.14	5	1.51	2	1.60	1	0.51	4
200	0.81	3	0.45	5			1.75	2	0.79	4
300	0.91	3	0.34	5			1.80	2	0.73	4
400	1.10	3	0.30	5			2.08	2	0.77	4
500			0.17	5					0.91	4
600	0.85	3	0.20	5			2.60	2	0.76	4
700	0.77	3	0.18	5			2.21	2	0.71	4
800	0.72	3	0.23	5			3.06	2	0.69	4
900	0.84	4	0.37	5			2.88	2	0.85	3
1000	0.77	4	0.28	5			1.93	2	0.82	3
Points		**29**		**50**		**2**		**17**		**38**

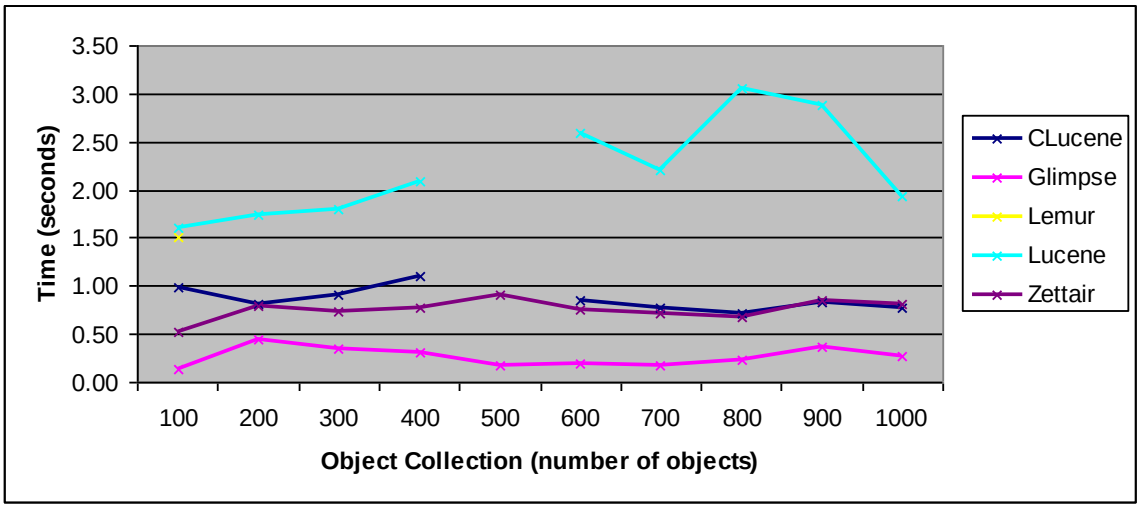

Figure 4.2. Time to Perform First Search.

Table 4.3. Time to Perform Second Search.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Time	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**
100										
200	0.73	4	0.50	5			1.82	2	0.93	3
300	0.82	3	0.36	5			1.96	2	0.73	4
400	0.88	3	0.29	5			2.11	2	0.66	4
500			0.19	5					0.74	4
600	0.79	3	0.19	5			2.89	2	0.69	4
700	0.84	3	0.20	5			2.64	2	0.71	4
800	0.83	3	0.27	5			2.55	2	0.74	4
900	0.92	3	0.25	5			1.91	2	0.70	4
1000	0.73	4	0.23	5			2.00	2	0.82	3
Points		**26**		**45**		**0**		**16**		**34**

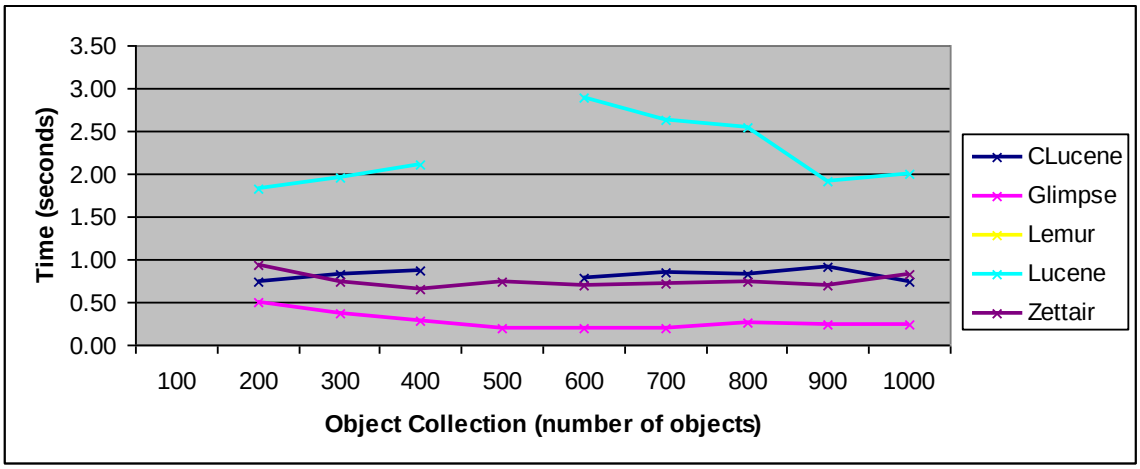

Figure 4.3. Time to Perform Second Search

Table 4.4. Changes in Search Times.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Time	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**
100										
200	0.08	3	0.05	5			0.07	4	0.13	2
300	0.09	3	0.02	4			0.16	2	0.00	5
400	0.22	2	0.01	5			0.03	4	0.11	3
500			0.02	5					0.18	4
600	0.06	4	0.01	5			0.29	2	0.07	3
700	0.07	3	0.02	4			0.43	2	0.00	5
800	0.11	3	0.04	5			0.51	2	0.05	4
900	0.08	5	0.12	4			0.97	2	0.15	3
1000	0.04	4	0.05	3			0.07	2	0.00	5
Points		**27**		**40**		**0**		**20**		**34**

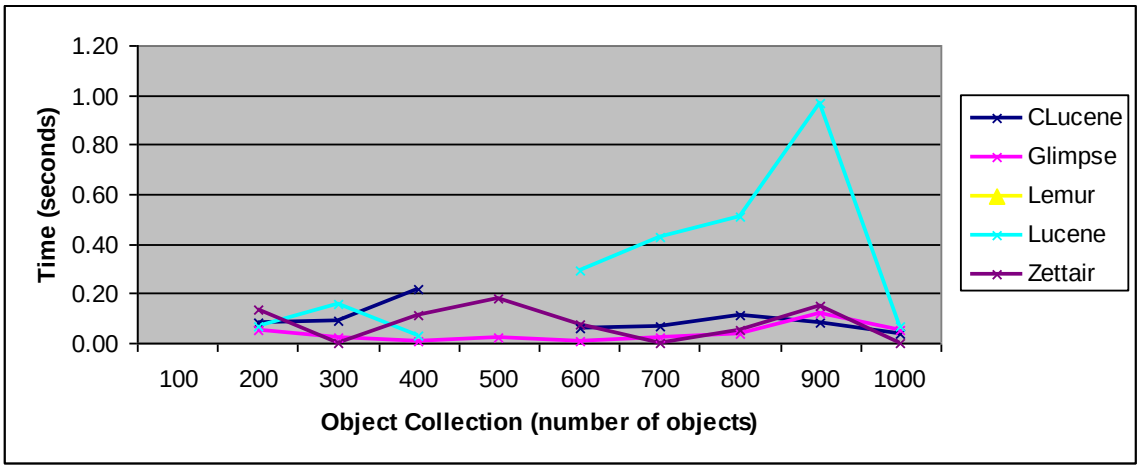

Figure 4.4. Changes in Search Times.

4.3 Index Maintenance Performance

Index maintenance falls into the same category as index creation: speed matters only if a thorough index can be created. And again, the importance of a retrieval solution's ability to update an index may be minimal to a special collections library.

Contrastingly, it may be the most important characteristics for use with an online news feed service.

To measure index maintenance performance, each retrieval solution created an index for an object collection. The object was changed, and the update command was issued. This was repeated ten times. The total time required to update the existing index, representing a 20%-Trimmed Mean, for each object collection can be seen in Table 4.5 and Figure 4.5. Complete results, including the times for each of the ten iterations, are included in Appendix L.

Table 4.5. Time to Maintain Index.

	CLucene		**Glimpse**		**Lemur**		**Lucene**		**Zettair**	
	Time	**Pts**	**Time**		**Time**	**Pts**	**Time**	**Pts**	**Time**	**Pts**
100										
200	126.71	3	75.84	4			214.42	2	29.26	5
300	154.10	3	91.08	4			260.85	2	37.03	5
400	187.85	3	105.80	4			315.75	2	44.96	5
500			138.63	4					55.74	5
600	270.79	3	180.19	4			465.70	2	61.34	5
700	314.21	3	222.83	4			517.81	2	70.83	5
800	353.50	3	277.77	4			568.87	2	79.02	5
900	371.52	3	272.54	4			655.08	2	81.39	5
1000	496.59	3	292.03	4			731.57	2	91.76	5
Points		**24**		**36**		**0**		**16**		**45**

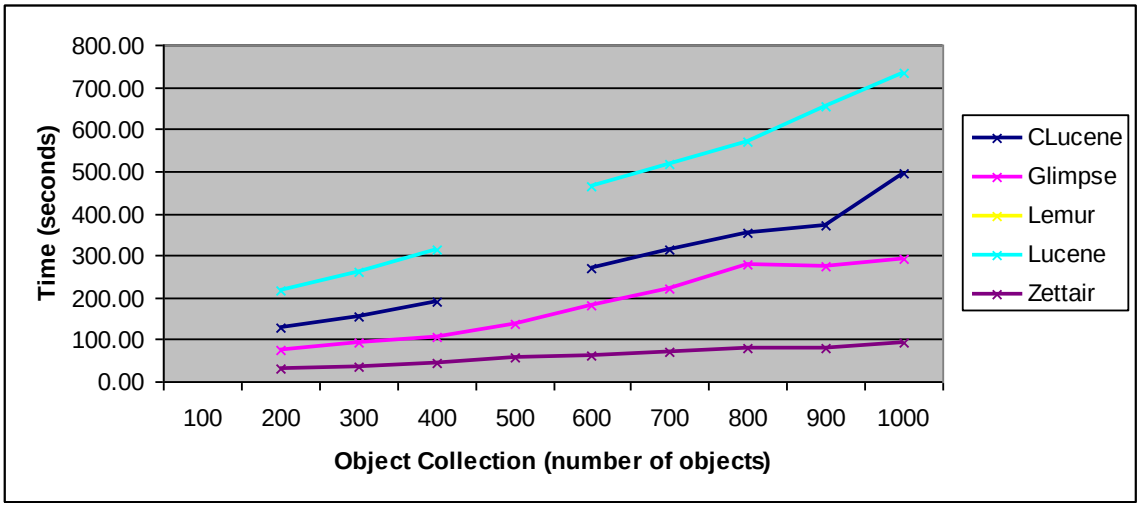

Figure 4.5. Time to Maintain Index.

4.4 Index Size

For applications where size is a concern, the ideal retrieval solution is the one that can produce the best results and consume the fewest resources. Tables 4.6 and 4.7 below record the size of the indexes after the index creation and index maintenance processes, respectively. The results are displayed in Figures 4.6 and 4.7.

Table 4.6. Index Size After Indexing.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Size	**Pts**	**Size**	**Pts**	**Size**	**Pts**	**Size**	**Pts**	**Size**	**Pts**
100	29.0	2	1.7	5	42.9	1	7.6	4	10.3	3
200	99.8	2	4.7	5			27.2	4	35.6	3
300	134.4	2	5.4	5			34.7	4	45.9	3
400	168.8	2	5.9	5			43.0	4	55.8	3
500			5.6	5					74.6	3
600	254.0	2	8.4	5			66.1	4	84.9	3
700	291.4	2	9.6	5			76.9	4	97.0	3
800	321.2	2	10.8	5			87.9	4	106.8	3
900	344.9	2	10.3	5			90.0	4	113.4	3
1000	393.9	2	11.2	5			96.9	4	127.9	3
Points		**18**		**50**		**1**		**36**		**30**

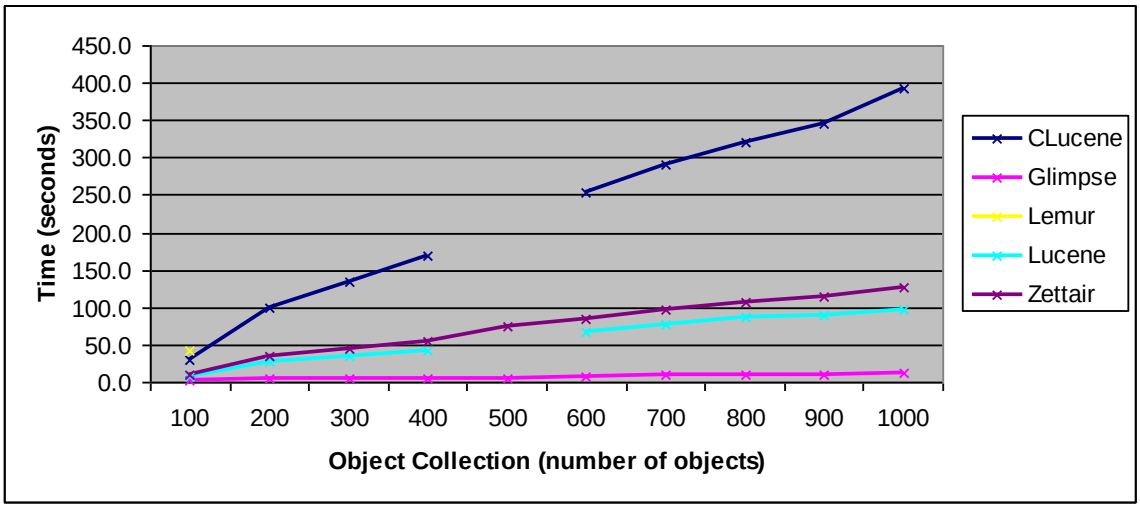

Figure 4.6. Index Size After Indexing.

Table 4.7. Index Size After Maintenance.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Size	**Pts**	**Size**	**Pts**	**Size**	**Pts**	**Size**	**Pts**	**Size**	**Pts**
100										
200	128.6	2	5.5	5			35.6	4	45.1	3
300	163.1	2	6.3	5			42.4	4	55.2	3
400	197.5	2	6.6	5			50.7	4	65.0	3
500			7.5	5					83.8	3
600	282.7	2	9.0	5			73.9	4	94.0	3
700	320.1	2	10.2	5			84.7	4	106.1	3
800	349.9	2	11.5	5			95.6	4	115.9	3
900	373.6	2	10.9	5			91.9	4	122.5	3
1000	422.6	2	11.8	5			104.7	4	136.9	3
Points		**16**		**45**		**0**		**32**		**27**

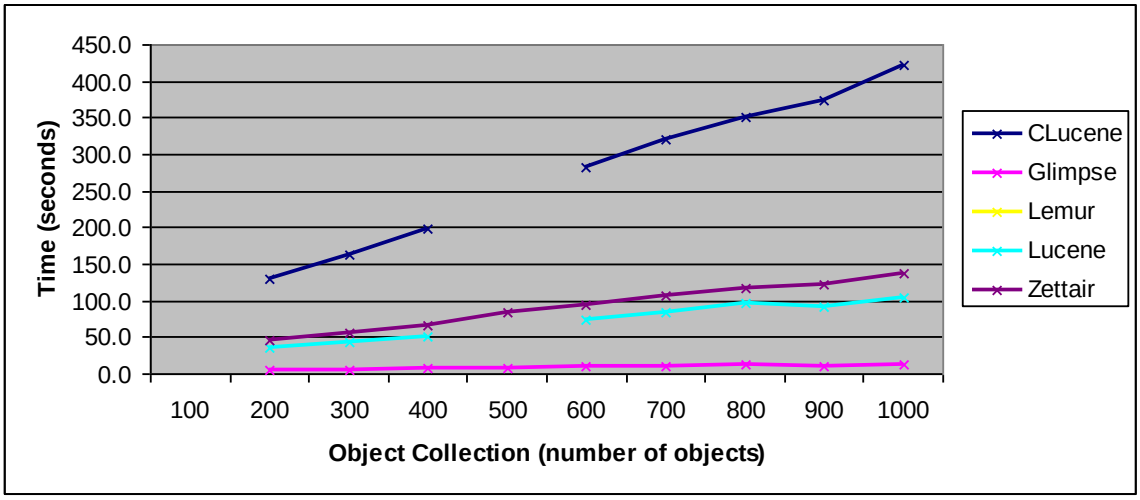

Figure 4.7. Index Size After Maintenance.

4.5 Matches

Tables 4.8 and 4.9 below record the number of matches during each search.

Specifically, Table 4.8 shows the number of matches found after the initial creation of the

index. Table 4.9 shows the number of matches after index maintenance was performed. The same results are displayed in Figures 4.8 and 4.9, respectively.

Table 4.8. Matches Found During First Search.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Match	**Pts**	**Match**	**Pts**	**Match**	**Pts**	**Match**	**Pts**	**Match**	**Pts**
100	52	2	100	5	50	1	65	3	83	4
200	90	2	200	5			131	3	180	4
300	108	2	300	5			175	3	239	4
400	164	2	400	5			244	3	318	4
500			498	5					375	4
600	250	2	599	5			365	3	470	4
700	280	2	698	5			427	3	549	4
800	294	2	800	5			423	3	581	4
900	339	2	898	5			524	3	680	4
1000	381	2	988	5			575	3	743	4
Points		**18**		**50**		**1**		**27**		**40**

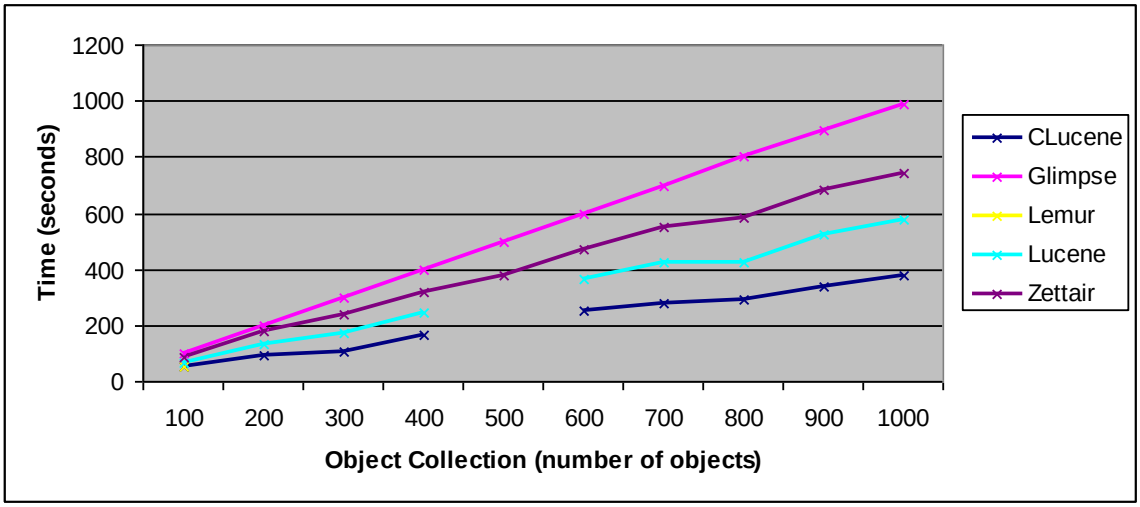

Figure 4.8. Matches Found During First Search.

Table 4.9. Matches Found During Second Search.

	CLucene		Glimpse		Lemur		Lucene		Zettair	
	Match	**Pts**	**Match**	**Pts**	**Match**	**Pts**	**Match**	**Pts**	**Match**	**Pts**
100										
200	142	2	300	5			196	3	263	4
300	160	2	400	5			240	3	322	4
400	216	2	500	5			309	3	401	4
500			598	5					458	4
600	302	2	699	5			430	3	553	4
700	332	2	798	5			492	3	632	4
800	346	2	900	5			488	3	664	4
900	391	2	998	5			589	3	763	4
1000	433	2	1088	5			640	3	826	4
Points		**16**		**45**		**0**		**24**		**36**

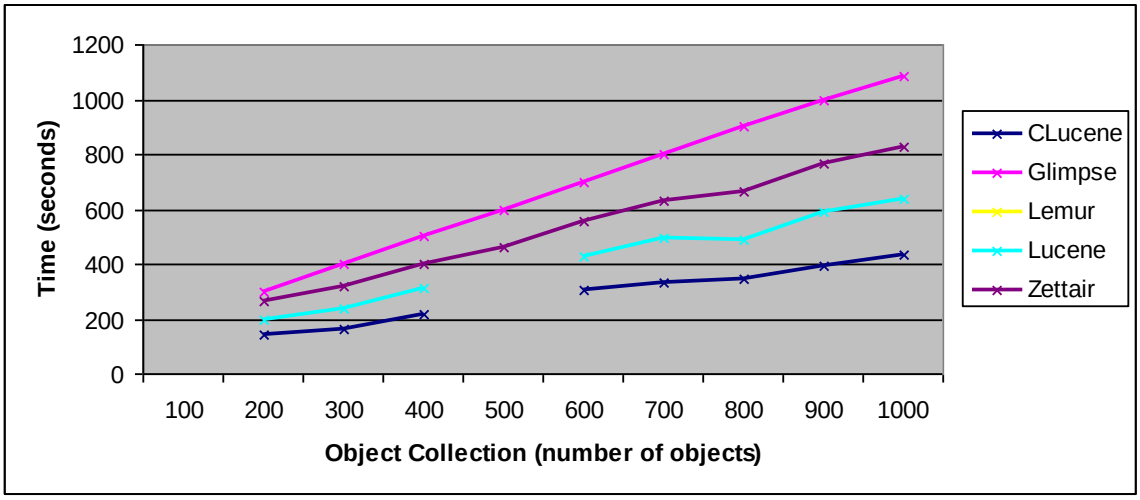

Figure 4.9. Matches Found During Second Search.

4.6 Problematic Pattern Resolution

Table 4.10 contains the results of the problematic pattern resolution scenarios. For each type of problematic pattern, a "Y" indicates that the pattern was handled successfully. A blank field indicates that the retrieval solution unsuccessfully handled

the pattern. For this research, and for the first section—Stopword Elimination—the retrieval solution correctly handled the pattern by not returning any matches. For all of the other patterns, a "Y" represents that a match was made. Of the nineteen different tests, Zettair correctly handled fifteen (79%); Lucene correctly handled eleven (58%); Glimpse and Lemur nine (47%); CLucene only eight (42%). Only one retrieval solution, Zettair, successfully retrieved the hyphenated words contained in the document. The points awarded for this test are included at the bottom of the table.

Table 4.10. Problematic Pattern Resolution.

	CLucene	**Glimpse**	**Lemur**	**Lucene**	**Zettair**
Stopword					
The	Y			Y	
Is	Y			Y	
It	Y			Y	
Hyphenation					
documents					Y
doc-uments		Y		Y	Y
e-mail		Y		Y	Y
email					Y
concerned					Y
con-cerned		Y		Y	Y
Numbers					
100			Y		
1979	Y		Y	Y	Y
Acronym					
IR	Y	Y	Y	Y	Y
IT		Y	Y		Y
Abbreviation					
Mr.	Y	Y		Y	Y
Capitalization					
Rijsbergen	Y	Y	Y	Y	Y
rijsbergen	Y	Y	Y	Y	Y
IT		Y	Y		Y
It			Y		Y
Formula					
Y * 100			Y		Y
Points	**10**	**25**	**25**	**40**	**50**

4.7 Summary of Points Awarded

Table 4.11 contains a summary of the points awarded for each test and a total of points awarded to each retrieval solution. Point calculations were calculated using a characteristic weight equal to one: each test was valued equally.

Table 4.11. Summary of Points Awarded.

	CLucene	**Glimpse**	**Lemur**	**Lucene**	**Zettair**
	Pts	**Pts**	**Pts**	**Pts**	**Pts**
Index	27	40	1	18	50
Index Size	18	50	1	36	30
Search	29	50	2	17	38
Match	18	50	1	27	40
Update	24	36	0	16	45
Update Size	16	45	0	32	27
Search	26	45	0	16	34
Match	16	45	0	24	36
Search Diff	27	40	0	20	34
Problematic Patterns	10	25	25	40	50
Total	**211**	**426**	**30**	**246**	**384**

4.8 Conclusions

Based on the test results, it is possible to reach powerful objective conclusions regarding the performance of the information retrieval solutions. For the test cases, the infrastructure showed the following characteristics for each retrieval solution:

- Based on the points awarded, Glimpse was the best IR solution of those evaluated for the given operating environments and IR solutions.
- The Zettair solution was able to index object collections faster than all of the other retrieval solutions. Based on trends seen during testing, Zettair was also the most scalable solution.
- CLucene and Lucene failed to process one file and crashed when they encountered the file. Glimpse failed to process eighteen files, however, unlike CLucene and Lucene, Glimpse was able to skip such files and continue processing. The implementations of both CLucene and Lucene were demonstration environments; Glimpse was a delivered product. CLucene and Lucene might have lacked attention to parameterization and error recovery that accompany a delivered product.
- CLucene created the largest index. The size of the CLucene index was almost equal to the size of the object collection. The Glimpse index was the smallest—typically one-tenth the size of the CLucene index. The difference between the CLucene and Lucene indices are worth noting. These two retrieval solution can create identical indices for the same object collection. The differences seen during these tests are attributed to different default parameters employed by each.
- For the collections containing up to 400 objects, the Lucene and CLucene solutions retrieved the same number of results. For the larger collections and despite its smaller index, the Lucene solution retrieved more results.
- Glimpse retrieved the most matches—one per document. It is apparent that Glimpse, by default, executes a pattern matting matching query. Where the other

retrieval solutions only matched documents containing the word “hat”, Glimpse reported a match for documents containing “that”, “chat”, “hatch”, etc.

- Glimpse executed queries much faster than the Lucene and CLucene solutions. Surprisingly, it was also able to retrieve significantly more results in considerably less time, sometimes executing the query in half the time of CLucene, and one-fourth the time of Lucene. Again, this I attributed to the smaller index size.
- Glimpse featured the most consistent search times. This was apparent when comparing the differences between the first and second searches. The absence of steep spikes for Glimpse in Figure 4.4 reflects the minimal effects the index maintenance had on Glimpse’s performance.
- Zettair outperformed the competition by nearly 20% during the Problematic Patterns Test. Zettair was the only retrieval solution that successfully handled hyphenated words.
- An intriguing result of this research was the discovery of the inability of the retrieval solutions to handle certain problematic patterns. The failure to recognize standard hyphenation was especially surprising. Also puzzling was the correct indexing of the number 1979, and the failure to match the number 100. The resolution of these failures will surely increase the quality and dependability of query results.

Chapter 5

5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Research Summary

The creation of an infrastructure capable of measuring the performance of information retrieval solutions enables an effective comparison between information retrieval solutions. A number of objective characteristics, including the time required to index an object collection, the time required to execute queries against an index, and the time required maintain an index can be evaluated together with the ability to retrieve relevant data to determine which retrieval solution best meets an operating environment's need.

Eleven unique object collections were created for this research. The objects were text files, ranging in size from 4KB to 30MB. Five information retrieval solutions were used as test cases. They were: CLucene version .9.1.0, Glimpse version 4.18.5, Lemur version 4.5, Lucene version 2.1.0 and Zettair version 0.9.3. Each solution was modified to normalize processing behavior. A control script was used to initiate the tests and record measurements. The results from the tests were compiled and a 20%-trimmed mean was calculated.

The tests that were executed measured how quickly each retrieval solution could create a new index, maintain (update) the index, and perform searches against the index. The size of the index was also recorded. Two other measures provided an idea of the

quality of the results: the number of matches found and the ability of the retrieval solution to correctly resolve problematic language patterns.

Finally, the results were presented for review. The infrastructure was designed to aid developers, researchers and IT professionals in the identification of the best retrieval solution based on the characteristic prioritization. Once all tests have completed and all points have been awarded, the best retrieval solution will be the one with the most points.

5.2 Contributions

This research provides approaches to be considered when measuring and comparing the performance of IR solutions. By design, typical evaluation infrastructures fail to take into account factors beyond the IR solution and do not provide a platform-independent environment where many IR solutions may be evaluated. This research addresses those weaknesses. Specifically, the contributions of this research include:

- A platform-independent infrastructure capable of measuring IR performance.
- A method for developers, researchers, and professionals to identify those performance characteristics that are most important given the condition of the operating environment.
- A method for preparing IR solutions for evaluation and comparison.
- The creation of randomly seeded information collections for the measurement of non-specific IR tasks.
- The creation of a known information collection for the measure of specific IR characteristics.

- The establishment of a method for measuring Precision and Recall through the use of specific, well-known collections.
- A method for analyzing the results obtained through the application of the infrastructure in order to identify the best IR solution.
- The presentation of errors that may be encountered during evaluation of IR solutions, their sources, and how to account for them.

5.3 Lessons Learned and Implications for Future Applications

An evaluation infrastructure was designed and built to measure a finite set of key information retrieval characteristics. The measurements are taken as different software solutions execute retrieval processes against specific object collections. Additional characteristics may be measured in the future. The infrastructure was designed to allow for bolt-on enhancements to be added on later. This research limited the set of measured values to five basic retrieval characteristics. These five were chosen as a starting point for future research efforts. The conclusions drawn from applications of this infrastructure will vary as different information retrieval solutions are tested and different characteristic priorities are selected. Because of these variables, it is not possible to draw the conclusion that Retrieval Solution A is, and always will be, better than Retrieval Solution B. It is, however, possible to reach objective conclusions regarding the performance of the information retrieval solutions under controlled conditions.

An accurate evaluation of IR solutions must take into account a number of variables. These variables proved to have a significant impact on IR performance. Additionally, certain methods were established to allow for the calculation of complicated measures and the testing of specific problematic language patterns. As

research in the area of IR performance evaluation moves forward, the following observations will be of value to developers, researchers, and IT professionals.

- The infrastructure prioritizes and assigns a weight to each performance characteristic, submits one or more IR solutions to a series of tests, awards points to each IR solution, adjusts the points awarded by the characteristic weight, and, finally, identifies the best IR solution based on the adjusted point totals.
- The infrastructure is capable of measuring a number of retrieval characteristics. Before performing the tests, all of the characteristics to be evaluated should be prioritized and assigned a weight to be used in the awarding of points. It can be seen that given the same test results, different IR solutions may be identified as "best" based on the prioritized characteristics.
- The infrastructure requires the IR solution to be executable via a single statement issued through the command line. Where this configuration does not exist, appropriate software for integration can be created. For example, before indexing an information collection, CLucene prompted for the location of the collection and the location to store the index. The software was modified to accept a single command, with these additional values passed as parameters. As another example, if one needed to evaluate a web-based service, a simple script could be constructed to perform the service requests.
- The Run Control is a script that issues all statements to the command line. This configuration allows for single-point modification. To add or change an IR solution, or to add or change document collection, the commands contained within this file should be modified.

- This research set forth a method for testing special conditions and calculating IR performance measures like Precision and Recall. The difficulty in calculating these measures is due to the difficulty of dealing with unknown information collections. To overcome this, a known information collections may be introduced. To illustrate the approach a special collection was created. The collection contained one document. This allowed for the establishment of an absolute test: there was one relevant document, and the IR solution either retrieved it, or didn't. This could then be used to test the specific desired characteristics.
- Ten unique information collections were created to measure the time required to complete IR tasks. These collections were randomly seeded with plain text files. The contents of the collections should be similar to those of the intended operating environment.
- The internal settings of IR solutions can produce results that are deceiving. For example, during initial testing, the CLucene index was the largest index and the Lucene index was one of the smallest. In previous tests, the opposite had been true. Further investigation revealed the version of Lucene used in this research had a default setting that limited how much of each document was indexed. With that setting left as delivered, Lucene was indexing only the beginning of each document while CLucene was indexing the entire document. Once the setting was altered to allow for full indexing, the results came into agreement with prior expectations. This example emphasizes the importance of normalization as it

pertains to IR performance. Without a thorough effort to normalize the test cases, one cannot be sure that the comparison performed is accurate.

- The implementation of different IR strategies also has a significant impact on IR performance. For example, with stemming enabled, the index will contain spelling variations of words contained within the document. With stemming disabled, only the exact words contained within the document will be indexed. This means that the same IR solution, with different operating options selected, can and should be evaluated as if it were two distinct IR solutions.
- As seen during this research, file size impacts IR performance. One of the collections contained a 31MB document. The document was a table of contents. It comprised a list of literary titles followed a series of numeric codes. The numeric codes were interpreted by some of the IR solutions as words. Thus, the file contained thousands of unique words. Some of the IR solutions were unable to process such a large quantity of unique terms and crashed before the indexing process completed. Because smaller files will potentially have fewer unique terms than larger files, IR performance against larger documents of larger sizes does not provide an accurate estimate of IR performance against documents of smaller sizes.
- This research limited the contents of the information collections to data stored in the plain text file format. Following the same pattern established for testing the handling of problematic language patterns, copies of the same document, stored in a variety of file formats, could be added to an information collection. If three files were added, then the IR solution would be expected to find three relevant

documents. This test would help identify weaknesses in the parsing strategy for different file types.

- This research limited the type of query used to one: the Boolean query. As explained in earlier chapters, several other query types exist. While the Boolean query is the most popular, it lacks certain functionality. And the need to execute different types of queries would eliminate some IR solutions from even qualifying for evaluation. Zettair, for example, only supports Boolean including phrase queries. In a situation where partial matching was important, Zettair would fail to meet the functional requirements.

In summary, care must be taken in the setup of the tests and analysis of the results. As IR continues to evolve, the approaches included in this infrastructure can also evolve to continue to provide meaningful performance evaluations.

BIBLIOGRAPHY

Baeza-Yates, Navarro, G., (1996). Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, Vol 25, Iss 1, 67-79.

Baeza-Yates, R., Neto, B., (1999). *Modern Information Retrieval*. New York, Addison Wesley, Reading, MA: ACM Press.

Brin, S., Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30, 1-7 (Apr. 1998), 107-117. DOI= http://dx.doi.org/10.1016/S0169-7552(98)00110-X.

Broadbent, R. E., Saunders, G., Ekstrom, J. J., (2006). An infrastructure for the evaluation and comparison of information retrieval systems. In Proceedings of the 7th Conference on information Technology Education (Minneapolis, Minnesota, USA, October 19 - 21, 2006). SIGITE '06. ACM, New York, NY, 123-127. DOI= http://doi.acm.org/10.1145/1168812.1168842.

Burkowski, F., (1992). An algebra for hierarchically organizing text-dominated databases. *Information Processing & Management*, Vol 28, Iss 3, 333-348.

Burkowski, F., (1992). Retrieval activities in a database consisting of heterogeneous collections of structured text. *Proceedings of the 15th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, 112-125.

Cahoon, B., McKinley, K. S., Lu, Z., (2000). Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems,* Vol 18, Iss 1, 1-43. Retrieved February 3, 2006 from http://doi.acm.org/10.1145/333135.333136.

Croft, W. B., Turtle, H. R., (1992). Text Retrieval and Inference. In P. S. Jacobs, editor, *Text-based Intelligent Systems: current research and practice in information extraction and retrieval*. Lawrence Erlbaum Associates, Inc, New Jersey, 102-130.

Fox, C., (1992). Lexical analysis and stoplists. In W. Frakes and R. Baeza-yates, editors, *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 102-130.

Frakes, W., Baeza, R., (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ.

Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A., Lochbaum, K. E., (1988). Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 465-480.

Gao, X., Murugesan, S., Lo, B., (2004). Multi-Dimensional Evaluation of Information Retrieval Results. *Proceedings of the IEEE/WIC/ACM International Conference*, 192-198. Retrieved on January 27, 2006 from http://doi.ieeecomputersociety.org/10.1109/WI.2004.10160

Gull, C. D. (1987). Historical note: Information science and technology: From coordinate indexing to the global brain. *Journal of the American Society for Information Science*, 38(5), 338-366.

Hart, M. S., (1992). History and Philosophy of Project Gutenberg. Retrieved on April 20, 2007 from http://www.gutenberg.org/about/history

Hessel, A., (1955). *A history of Libraries*. The Scarecrow Press, New Brunswick, NJ. Translated by Reuben Peiss.

Korfhage, R., (1997). *Information Storage and Retrieval.* John Wiley & Sons, Inc.

Landoni, M., Bell, S., (2000). Information retrieval techniques for evaluating search engines: a critical overview. *Aslib Proceedings*, 52, 3, 124-129. Retrieved January 27, 2006 from http://www.emeraldinsight.com/10.1108/EUM0000000007006.

Navarro, G., Baeza, R., (1997). Proximal Nodes: A model to query document databases by content and structure. *ACM Transactions on Office and Information Systems*, 15(4), 401-435.

van Rijsbergen, C. J., (1979). Information Retrieval (2nd Ed.). Butterworths, London.

Riloff, E., Hollaar, L., (1996). Text databases and information retrieval. *ACM Computing Surveys*, Vol. 28, Iss. 1, 133-135. Retrieved January 27, 2006 from http://doi.acm.org/10.1145/234313.234371

Robertson, S. E., Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Sciences*, 27(3), 129-146.

Salton, G., ED., (1971). *The SMART Retrieval solution: Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ.

Salton, G., Fox, E. A., Wu, H., (1983). Extended Boolean information retrieval. *Communications of the ACM*, Vol 26, Iss 11, 1022-1036.

Salton, G., McGill, M. J., (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

APPENDICES

APPENDIX A: CLUCENE – MAIN.CPP

The delivered Main.cpp file was significantly modified. The original file presented a series of prompts, requiring the user to enter the location of the object collection, the location wherein to build the index, etc. The modified file allowed for these parameters to be passed in automatically. Additional code designed to record the time spent creating the index or executing a query was commented out.

```
/*------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------*/
#include "stdafx.h"
#include "CLucene.h"

#ifdef _CLCOMPILER_MSVC
#ifdef _DEBUG
	#define CRTDBG_MAP_ALLOC
	#include <stdlib.h>
	#include <crtdbg.h>
#endif
#endif

#include <iostream>

using namespace std;

void DeleteFiles(const char* dir);
void IndexFiles(char* path, char* target, const bool clearIndex);
void SearchFiles(const char* index, char* search_str);
void getStats(const char* directory);

int main(int argc,char *argv[])
{
	string com	= argv[1];
	char*  source_dir    = argv[2];
	char*  path_to_index = argv[3];
	char*  search_str    = argv[4];

	if (com=="-i"){
```

```
            IndexFiles(source_dir,path_to_index,true);
        } else if (com=="-s"){
            SearchFiles(path_to_index,search_str);
        }
        return 0;
}
```

APPENDIX B: CLUCENE – INDEX.CPP

The delivered IndexFiles.cpp file was modified and saved as Index.cpp. All commands to print to the screen were commented out. Code designed to measure the time required to index the collection was also commented out. The resulting program created a new instance of an IndexWriter, created an index for an object collection, and terminated.

```
/*------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------*/
#include "stdafx.h"

#include "CLucene.h"
#include "CLucene/util/Reader.h"
#include "CLucene/util/Misc.h"
#include "CLucene/util/dirent.h"
#include <iostream>
#include <fstream>

using namespace std;
using namespace lucene::index;
using namespace lucene::analysis;
using namespace lucene::util;
using namespace lucene::store;
using namespace lucene::document;

Document* FileDocument(const char* f){
        // make a new, empty document
        Document* doc = _CLNEW Document();

        // Add the path of the file as a field named "path".  Use a Tex t field, so
        // that the index stores the path, and so that the path is searchable
  TCHAR tf[CL_MAX_DIR];
  STRCPY_AtoT(tf,f,CL_MAX_DIR);
  doc->add( *Field::Text(_T("path"), tf ) );

        // Add the last modified date of the file a field named "modified".  Use a
        // Keyword field, so that it's searchable, but so that no attempt is made
        // to tokenize the field into words.
```

```
        //doc->add( *Field.Keyword("modified", DateField.timeToString(f-
>lastModified())));

        // Add the contents of the file a field named "contents".  Use a Text
        // field, specifying a Reader, so that the text of the file is tokenized.

   //read the data without any encoding. if you want to use special encoding
   //see the contrib/jstreams - they contain various types of stream readers
   FILE* fh = fopen(f,"r");
        if ( fh != NULL ){
                StringBuffer str;
                // use fstat for portability
                int fn = fileno(fh);
                struct stat filestat;
                fstat(fn, &filestat);
                str.reserve(filestat.st_size);
                //str.reserve(fileSize(fh->_file));
                char abuf[1024];
                TCHAR tbuf[1024];
                size_t r;
                do{
                        r = fread(abuf,1,1023,fh);
                        abuf[r]=0;
                        STRCPY_AtoT(tbuf,abuf,r);
                        tbuf[r]=0;
                        str.append(tbuf);
                }while(r>0);
                fclose(fh);

                doc->add( *Field::Text(_T("contents"),str.getBuffer()) );
        }

        // return the document
        return doc;
}

void indexDocs(IndexWriter* writer, char* directory) {
        DIR* dir = opendir(directory);
        if ( dir != NULL ){
                struct dirent* fl;

                struct fileStat buf;

                char path[CL_MAX_DIR];
                strcpy(path,directory);
                strcat(path,PATH_DELIMITERA);
```

```
                char* pathP = path + strlen(path);
                fl = readdir(dir);
                while ( fl != NULL ){
                        if ( (strcmp(fl->d_name, ".")) && (strcmp(fl->d_name, "..")) ) {
                        pathP[0]=0;
                        strcat(pathP,fl->d_name);
                        int32_t ret = fileStat(path,&buf);
                        if ( buf.st_mode & S_IFDIR ) {
                                indexDocs(writer, path );
                        }else{
                                Document* doc = FileDocument( path );
                                writer->addDocument( doc );
                                _CLDELETE(doc);
                        }
                }
                fl = readdir(dir);
                }
                closedir(dir);
        }else{
                printf( "adding: %s\n", directory);

                Document* doc = FileDocument( directory );
                writer->addDocument( doc );
                _CLDELETE(doc);
        }
}
void IndexFiles(char* path, char* target, const bool clearIndex){
        IndexWriter* writer = NULL;
        lucene::analysis::standard::StandardAnalyzer an;

        if ( !clearIndex && IndexReader::indexExists(target) ){
                if ( IndexReader::isLocked(target) ){
                        printf("Index was locked... unlocking it.\n");
                        IndexReader::unlock(target);
                }

                writer = _CLNEW IndexWriter( target, &an, false);
        }else{
                writer = _CLNEW IndexWriter( target ,&an, true);
        }
        writer->setMaxFieldLength(IndexWriter::DEFAULT_MAX_FIELD_LENGTH);
        indexDocs(writer, path);
        writer->optimize();
        writer->close();
        _CLDELETE(writer);
}
```

APPENDIX C: CLUCENE – SEARCHFILES.CPP

The delivered SearchFiles.cpp file was modified. All of the code designed to format the results of the query was commented out. Additionally, code designed to measure and the display the time required to execute the query was disabled. The resulting program created a new instance of an IndexSearcher, created a query, printed the query parameters, executed the query, printed the number of matching documents, and terminated.

```
/*------------------------------------------------------------------------
* Copyright (C) 2003-2006 Ben van Klinken and the CLucene Team
*
* Distributable under the terms of either the Apache License (Version 2.0) or
* the GNU Lesser General Public License, as specified in the COPYING file.
------------------------------------------------------------------------*/
#include "stdafx.h"

#include "CLucene.h"
#include <iostream>

using namespace std;
using namespace lucene::analysis;
using namespace lucene::index;
using namespace lucene::util;
using namespace lucene::queryParser;
using namespace lucene::document;
using namespace lucene::search;

void SearchFiles(const char* index, char* line){
   //Searcher searcher(index);
         standard::StandardAnalyzer analyzer;
         TCHAR tline[80];
         const TCHAR* buf;

         IndexSearcher s(index);
           STRCPY_AtoT(tline,line,80);
                Query* q = QueryParser::parse(tline,_T("contents"),&analyzer);
                buf = q->toString(_T("contents"));
                _tprintf(_T("Searching for: %s\n"), buf);
                _CLDELETE_CARRAY(buf);
                Hits* h = s.search(q);
                printf("%d Total Matching Documents.\n", h->length());
```

```
            _CLDELETE(h);
            _CLDELETE(q);
        s.close();
        //delete line;
}
```

APPENDIX D: GLIMPSE – MAIN.C

Throughout the indexing and searching processes, there were a number of prompts that were eliminated. Glimpse also performed a number of calculation before indexing a collection, including counting the number of files in the collection and calculating the size of the collection. These operations were bypassed to create an operational environment similar to the other retrieval solution mentioned above.

```
/* Copyright (c) 1994 Sun Wu, Udi Manber, Burra Gopal.  All Rights Reserved. */
/* bgopal: (1993-4) redesigned/rewritten using agrep's library interface */
#include <autoconf.h>
#include <sys/param.h>
#include <errno.h>
#include "glimpse.h"
#include "defs.h"
#include <fcntl.h>
#include "checkfile.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/file.h>  /* for flock definition */
#if       ISO_CHAR_SET
#include <locale.h>    /* support for 8bit character set */
#endif

#define CLIENTSERVER      1
#define USE_MSGHDR        0
#define USE_UNIXDOMAIN          0
#define DEBUG         0

#define DEF_SERV_PORT   2001
#define MIN_SERV_PORT   1024
#define MAX_SERV_PORT 30000
#define SERVER_QUEUE_SIZE      10       /* number of requests to buffer up while
processing one request = 5 */

/* Borrowed from C-Lib */
extern char **environ;
extern int errno;

#if       CLIENTSERVER
#include "communicate.c"
```

```
#endif  /*CLIENTSERVER*/

/* For client-server protocol */
CHAR *SERV_HOST = NULL;
int     SERV_PORT;
char    glimpse_reqbuf[MAX_ARGS*MAX_NAME_LEN];
extern int glimpse_clientdied;/* set if signal received about dead socket: need agrep
variable so that exec() can return quickly */
int     glimpse_reinitialize = 0;

/* Borrowed from agrep.c */
extern int D_length;            /* global variable in agrep */
extern int D;                   /* global variable in agrep */
extern int pattern_index;
/* These are used for byte level index search */
extern CHAR CurrentFileName[MAX_LINE_LEN];
extern int SetCurrentFileName;
extern int CurrentByteOffset;
extern int SetCurrentByteOffset;
extern long CurrentFileTime;
extern int SetCurrentFileTime;
extern int execfd;
extern int  agrep_initialfd;
extern CHAR *agrep_inbuffer;
extern int  agrep_inlen;
extern int  agrep_inpointer;
extern FILE *agrep_finalfp;
extern CHAR *agrep_outbuffer;
extern int  agrep_outlen;
extern int  agrep_outpointer;
extern int glimpse_call;        /* prevent agrep from printing out its usage */
extern int glimpse_isserver;    /* prevent agrep from asking for user input */
int     first_search = 1;       /* intra/interaction in process_query() and glimpse_search()
*/
#if     ISSERVER
int     RemoteFiles = 0;        /* Are the files present locally or remotely? If on, then -NQ
is automatically added to all search options for each query */
#endif

/* Borrowed from index/io.c */
extern int InfoAfterFilename;
extern int OneFilePerBlock;
extern int StructuredIndex;
extern unsigned int *dest_index_set;
extern unsigned char *dest_index_buf;
extern unsigned int *src_index_set;
```

```
extern unsigned char *src_index_buf;
extern unsigned char *merge_index_buf;
extern int mask_int[32];
extern int indexable_char[256];
int test_indexable_char[256];
extern int p_table[MAX_PARTITION];
extern int GMAX_WORD_SIZE;
extern int IndexNumber;                 /* used in getword() */
extern int InterpretSpecial;    /* used to "not-split" agrep-regexps */
extern int UseFilters;          /* defined in build_in.c, used for filtering routines in io.c */
extern int ByteLevelIndex;
extern int RecordLevelIndex;
extern int rdelim_len;
extern char rdelim[MAX_LINE_LEN];
extern char old_rdelim[MAX_LINE_LEN];
extern int file_num;
extern int REAL_PARTITION, REAL_INDEX_BUF, MAX_ALL_INDEX,
FILEMASK_SIZE;

/* Borrowed from get_filename.c */
extern int bigbuffer_size;
extern char *bigbuffer;
extern char *outputbuffer;

/* OPTIONS/FLAGS */
int     veryfast = 0;
int     CONTACT_SERVER = 0;     /* Should client try to call server at all or just
process query on its own? */
int     NOBYTELEVEL = 0;        /* Some cases where we cannot do byte level fast-
search: ALWAYS 0 if !ByteLevelIndex */
int     OPTIMIZEBYTELEVEL = 0;          /* Some cases where we don't want to do
byte level search since number of files is small */
int     GCONSTANT = 0;          /* should pattern be taken as-is or parsed? */
int     GLIMITOUTPUT = 0;       /* max no. of output lines:
0=>infinity=default=nolimit */
int     GLIMITTOTALFILE = 0;    /* max no. of files to match:
0=>infinity=default=nolimit */
int     GLIMITPERFILE = 0;      /* not used in glimpse */
int     GBESTMATCH = 0;         /* Should I change -B to -# where # = no. of errors?
*/
int     GRECURSIVE = 0;
int     GNOPROMPT = 1;
int     GBYTECOUNT = 0;
int     GPRINTFILENUMBER = 0;
int     GPRINTFILETIME = 0;
int     GOUTTAIL = 0;
```

```
int	GFILENAMEONLY = 0;	/* how to do it if it is an and expression in
structured queries */
int	GNOFILENAME=0;
int	GPRINTNONEXISTENTFILE = 0; /* if filename is not there in index, then at
least let user know its name */
int	MATCHFILE = 0;
int	PRINTATTR = 0;
int	PRINTINDEXLINE = 0;
int	Pat_as_is=0;
int	Only_first=1;		/* Do index search only */
int	PRINTAPPXFILEMATCH=1;	/* Print places in file where match occurs:
useful with -b only to analyse the index */
int	GCOUNT=1;		/* print number of matches rather than actual matches: used
only when PRINTAPPX = 1 */
int	HINTSFROMUSER=0;	/* The user gives the hints about where we should
search (result of adding -EQNgy) */
int	WHOLEFILESCOPE=0;	/* used only when foundattr is NOT set: otherwise,
scope is whole file anyway */
int	foundattr=0;		/* set in split.c -- != 0 only when StructuredIndex AND
query is structured */
int	foundnot=0;		/* set in split.c -- != 0 only when the not operator (~) is
present in the pattern */
int	FILENAMESINFILE=0;	/* whether the user is providing an explicit list of
filenames to be searched for pattern (if absent, then means all files) */
int	BITFIELDFILE=0;		/* Based on contribution From ada@mail2.umu.se
Fri Jul 12 01:56 MST 1996; Christer Holgersson, Sen. SysNet Mgr, Umea
University/SUNET, Sweden */
int	BITFIELDOFFSET=0;
int	BITFIELDLENGTH=0;
int	BITFIELDENDIAN=0;
int	GNumDays = 0;		/* whether the user wants files modified within
these many days before creating the index: only >0 makes sense */

/* structured queries */
CHAR ***attr_vals;		/* matrix of char pointers: row=max #of attributes,
col=max possible values */
CHAR **attr_found;		/* did the expression corr. to each value in attr_vals match?
*/
ParseTree *GParse;		/* what kind of expression corr. to attr are we looking for */

/* arbitrary booleans */
ParseTree terminals[MAXNUM_PAT];	/* parse tree's terminal node pointers pt. to
elements of this array; also used outside */
char	matched_terminals[MAXNUM_PAT];	/* ...[i] is 1 if i'th terminal matched:
used in filter_output and eval_tree */
int	num_terminals;		/* number of terminal patterns */
```

```
int     ComplexBoolean=0;  /* 1 if we need to use parse trees and the eval function */

/* index search */
CHAR *pat_list[MAXNUM_PAT];  /* complete words within global pattern */
int     pat_lens[MAXNUM_PAT];  /* their lengths */
int     pat_attr[MAXNUM_PAT];   /* set of attributes */
int     is_mgrep_pat[MAXNUM_PAT];
int     mgrep_pat_index[MAXNUM_PAT];
int     num_mgrep_pat;
CHAR pat_buf[(MAXNUM_PAT + 2)*MAXPAT];
int     pat_ptr = 0;
extern char INDEX_DIR[MAX_LINE_LEN];
char    *TEMP_DIR = NULL;       /* directory to store glimpse temporary files, usually
/tmp unless -T is specified */
char    indexnumberbuf[256];        /* to read in first few lines of the index */
char    *index_argv[MAX_ARGS];
int     index_argc = 0;
int     bestmatcherrors=0;    /* set during index search, used later on */
int     patindex;
int     patbufpos = -1;
char    tempfile[MAX_NAME_LEN];
char    *filenames_file = NULL;
char    *bitfield_file = NULL;

/* agrep search */
char    *agrep_argv[MAX_ARGS];
int     agrep_argc = 0;
CHAR *FileOpt;              /* the option list after -F */
int     fileopt_length;
CHAR GPattern[MAXPAT];
int     GM;
CHAR APattern[MAXPAT];
int     AM;
CHAR GD_pattern[MAXPAT];
int     GD_length;
CHAR **GTextfiles;
CHAR **GTextfilenames;
int     *GFileIndex;
int     GNumfiles;
int     GNumpartitions;
CHAR GProgname[MAXNAME];

/* persistent file descriptors */
#if     BG_DEBUG
FILE *debug;                /* file descriptor for debugging output */
#endif  /*BG_DEBUG*/
```

```
FILE   *timesfp = NULL;
FILE   *timesindexfp = NULL;
FILE   *indexfp = NULL;     /* glimpse index */
FILE   *partfp = NULL;              /* glimpse partitions */
FILE   *minifp = NULL;              /* glimpse turbo */
FILE   *nullfp = NULL;              /* to discard output: agrep -s doesn't work properly
*/
int     svstdin = 0, svstdout = 1, svstderr = 2;
static int one = 1;             /* to set socket option so that glimpseserver releases socket
after death */

/* Index manipulation */
struct offsets **src_offset_table;
struct offsets **multi_dest_offset_table[MAXNUM_PAT];
unsigned int *multi_dest_index_set[MAXNUM_PAT];
extern free_list();
struct stat index_stat_buf, file_stat_buf;
int timesindexsize = 0;
int last_Y_filenumber = 0;

/* Direct agrep access for bytelevel-indices */
extern int COUNT, INVERSE, TCOMPRESSED, NOFILENAME, POST_FILTER,
OUTTAIL, BYTECOUNT, SILENT, NEW_FILE,
        LIMITOUTPUT, LIMITPERFILE, LIMITTOTALFILE, PRINTRECORD,
DELIMITER, SILENT, FILENAMEONLY, num_of_matched, prev_num_of_matched,
FILEOUT;
CHAR matched_region[MAX_REGION_LIMIT*2 + MAXPATT*2];
int     RegionLimit=DEFAULT_REGION_LIMIT;

/* Returns number of matched records/lines. Uses agrep's options to output stuff nicely;
never called with RecordLevelIndex set */
int
glimpse_search(AM, APattern, GD_length, GD_pattern, realfilename, filename,
fileindex, src_offset_table, outfp)
        int             AM;
        unsigned char   APattern[];
        int             GD_length;
        unsigned char   GD_pattern[];
        char            *realfilename;
        char            *filename;
        int             fileindex;
        struct offsets  *src_offset_table[];
        FILE            *outfp;
{
        FILE            *infp;
        char            sig[SIGNATURE_LEN];
```

```
        struct offsets   **p1, *tp1;
        CHAR        *text, *curtextend, *curtextbegin, c;
        int         times;
        int         num, ret=0, totalret = 0;
        int         prevoffset = 0, begininterval = 0, endinterval = -1;
        CHAR        *beginregionptr = 0, *endregionptr = 0;
        int         beginpage = 0, endpage = -1;
        static int  MAXTIMES, MAXPGTIMES, pagesize;
        static int  first_time = 1;

        /*
         * If can't open file for read, quit
         * For each offset for that file:
         *      seek to that point
         *      go back until delimiter, go forward until delimiter, output it:
MAX_REGION_LIMIT is 16K on either side.
         *      read in units of RegionLimit
         *      before outputting matched record, use options to put prefixes (or use
memagrep which does everything?)
         * Algorithm changed: don't read same page in twice.
         */

        if (first_time) {
                pagesize = DISKBLOCKSIZE;
                MAXTIMES = ((MAX_REGION_LIMIT / RegionLimit) > 1) ?
(MAX_REGION_LIMIT / RegionLimit) : 1;
                MAXPGTIMES = ((MAX_REGION_LIMIT / pagesize) > 1) ?
(MAX_REGION_LIMIT / pagesize) : 1;
                first_time = 0;
        }
        /* Safety: must end/begin with delim */
        memcpy(matched_region, GD_pattern, GD_length);
        memcpy(matched_region+MAXPATT+2*MAX_REGION_LIMIT, GD_pattern,
GD_length);
        text = &matched_region[MAX_REGION_LIMIT+MAXPATT];

        if ((infp = my_fopen(filename, "r")) == NULL) return 0;
        NEW_FILE = ON;
#if     0
        /* Cannot search in .CZ files since offset computations will be incorrect */
        TCOMPRESSED = ON;
        if (!tuncompressible_filename(file_list[i], strlen(file_list[i]))) TCOMPRESSED =
OFF;
        num_read = fread(sig, 1, SIGNATURE_LEN, infp);
        if ((TCOMPRESSED == ON) && tuncompressible(sig, num_read)) {
                EASYSEARCH = sig[SIGNATURE_LEN-1];
```

```
                if (!EASYSEARCH) {
                        fprintf(stderr, "not compressed for easy-search: can miss some
matches in: %s\n", CurrentFileName);          /* not filename!!! */
                }
        }
        else TCOMPRESSED = OFF;
#endif  /*0*/

        p1 = &src_offset_table[fileindex];
        while (*p1 != NULL) {
                if ( (begininterval <= (*p1)->offset) && (endinterval > (*p1)->offset) ) {
        /* already covered this area */
#if     DEBUG
                        printf("ignoring %d in [%d,%d]\n", (*p1)->offset, begininterval,
endinterval);
#endif  /*DEBUG*/
                        tp1 = *p1;
                        *p1 = (*p1)->next;
                        my_free(tp1, sizeof(struct offsets));
                        continue;
                }

                TCOMPRESSED = OFF;
#if     1
                if ( (beginpage <= (*p1)->offset) && (endpage >= (*p1)->offset) &&
(text + ((*p1)->offset - prevoffset) + GD_length < endregionptr)) {
                        /* beginregionptr = curtextend - GD_length; /* prevent next
curtextbegin to go behind previous curtextend (!) */
                        text += ((*p1)->offset - prevoffset);
                        prevoffset = (*p1)->offset;
                        if (!((curtextend = forward_delimiter(text, endregionptr,
GD_pattern, GD_length, 1)) < endregionptr))
                                goto fresh_read;
                        if (!((curtextbegin = backward_delimiter(text, beginregionptr,
GD_pattern, GD_length, 0)) > beginregionptr))
                                goto fresh_read;
                }
                else { /* NOT within an area already read: must read another page: if
record overlapps page, might read page twice: no time to fix */
                fresh_read:
                        prevoffset = (*p1)->offset;
                        text = &matched_region[MAX_REGION_LIMIT+MAXPATT];
        /* middle: points to occurrence of pattern */
                        endpage = beginpage = ((*p1)->offset / pagesize) * pagesize;
                        /* endpage = (((*p1)->offset + pagesize) / pagesize) * pagesize */
```

```
                endregionptr = beginregionptr = text - ((*p1)->offset - beginpage);
        /* overlay physical place starting from this logical point */
                /* endregionptr = text + (endpage - (*p1)->offset); */
                curtextbegin = curtextend = text;
                times = 0;
                while (times < MAXPGTIMES) {
                        fseek(infp, endpage, 0);
                        num =
(&matched_region[MAX_REGION_LIMIT*2+MAXPATT] - endregionptr < pagesize) ?
(&matched_region[MAX_REGION_LIMIT*2+MAXPATT] - endregionptr) : pagesize;
                        if ((num = fread(endregionptr, 1, num, infp)) <= 0) break;
                        endpage += num;
                        endregionptr += num;
                        if (endregionptr <= text) {
                                curtextend = text;      /* error in value of offset: file
was modified and offsets no longer true: your RISK! */
                                break;
                        }
                        if (((curtextend = forward_delimiter(text, endregionptr,
GD_pattern, GD_length, 1)) < endregionptr) ||
                                (endregionptr >=
&matched_region[MAX_REGION_LIMIT*2 + MAXPATT])) break;
                        times ++;
                }
                times = 0;
                while (times < MAXPGTIMES) {    /* I have already read the
initial page since endpage is beginpage initially */
                        if ((curtextbegin = backward_delimiter(text,
beginregionptr, GD_pattern, GD_length, 0)) > beginregionptr) break;
                        if (beginpage > 0) {
                                if (beginregionptr - pagesize <
&matched_region[MAXPATT]) {
                                        if ((num = beginregionptr -
&matched_region[MAXPATT]) <= 0) break;
                                }
                                else num = pagesize;
                                beginpage -= num;
                                beginregionptr -= num;
                        }
                        else break;
                        times ++;
                        fseek(infp, beginpage, 0);
                        fread(beginregionptr, 1, num, infp);
                }
        }
#else   /*1*/
```

```
		/* Find forward delimiter (including delimiter) */
		times = 0;
		fseek(infp, (*p1)->offset, 0);
		while (times < MAXTIMES) {
			if ((num = fread(text+RegionLimit*times, 1, RegionLimit, infp)) >
0)
				curtextend = forward_delimiter(text,
text+RegionLimit*times+num, GD_pattern, GD_length, 1);
			if ((curtextend < text+RegionLimit*times+num) || (num <
RegionLimit)) break;
			times ++;
		}
		/* Find backward delimiter (including delimiter) */
		times = 0;
		while (times < MAXTIMES) {
			num = ((*p1)->offset - RegionLimit*(times+1)) > 0 ? ((*p1)-
>offset - RegionLimit*(times+1)) : 0;
			fseek(infp, num, 0);
			if (num > 0) {
				fread(text-RegionLimit*(times+1), 1, RegionLimit, infp);
				curtextbegin = backward_delimiter(text, text-
RegionLimit*(times+1), GD_pattern, GD_length, 0);
			}
			else {
				fread(text-RegionLimit*times-(*p1)->offset, 1, (*p1)-
>offset, infp);
				curtextbegin = backward_delimiter(text, text-
RegionLimit*times-(*p1)->offset, GD_pattern, GD_length, 0);
			}
			if ((num <= 0) || (curtextbegin > text-RegionLimit*(times+1)))
break;
			times ++;
		}
#endif  /*1*/

		/* set interval and delete the entry */
		begininterval = (*p1)->offset - (text - curtextbegin);
		endinterval = (*p1)->offset + (curtextend - text);

		if (strncmp(curtextbegin, GD_pattern, GD_length)) {
			/* always pass enclosing delimiters to agrep; since we have seen
text before curtextbegin + we have space, we can overwrite */
			memcpy(curtextbegin - GD_length, GD_pattern, GD_length);
			curtextbegin -= GD_length;
		}
#if	DEBUG
```

```
		c = *curtextend;
		*curtextend = '\0';
		printf("%s [%d < %d < %d], text = %d: %s\n", CurrentFileName,
begininterval, (*p1)->offset, endinterval, text, curtextbegin);
		*curtextend = c;
#endif  /*DEBUG*/

		tp1 = *p1;
		*p1 = (*p1)->next;
		my_free(tp1, sizeof(struct offsets));
		if (curtextend <= curtextbegin) continue;	/* error in offsets/delims */

		/*
		 * Don't call memagrep since that is heavy weight. Call exec
		 * directly after doing agrep_search()'s preprocessing here.
		 * PS: can add agrep variable not to do delim search if called from here
		 * since that prevents unnecessarily scanning the buffer for the 2nd time.
		 */
		CurrentByteOffset = begininterval+1;
		SetCurrentByteOffset = 1;
		first_search = 1;
		if (first_search) {
			if ((ret = memagrep_search(AM, APattern, curtextend-
curtextbegin, curtextbegin, 0, outfp)) > 0)
				totalret ++; /* += ret */
			else if ((ret < 0) && (errno == AGREP_ERROR)) {
				fclose(infp);
				return -1;
			}
			first_search = 0;
		}
		else {	/* All agrep globals are properly set: has a bug because agrep's
globals aren't properly reinitialized without agrep_search :-( */
			agrep_finalfp = (FILE *)outfp;
			agrep_outlen = 0;
			agrep_outbuffer = NULL;
			agrep_outpointer = 0;
			execfd = agrep_initialfd = -1;
			agrep_inbuffer = curtextbegin;
			agrep_inlen = curtextend - curtextbegin;
			agrep_inpointer = 0;
			if ((ret = exec(-1, NULL)) > 0)
				totalret ++; /* += ret; */
			else if ((ret < 0) && (errno == AGREP_ERROR)) {
				fclose(infp);
				return -1;
```

```
                        }
                }

                if (((LIMITOUTPUT > 0) && (LIMITOUTPUT <= num_of_matched)) ||
                   ((LIMITPERFILE > 0) && (LIMITPERFILE <= num_of_matched -
prev_num_of_matched))) break;       /* done */
                if ((totalret > 0) && FILENAMEONLY) break;
        } /* while *p1 != NULL */

        SetCurrentByteOffset = 0;
        fclose(infp);
        if (totalret > 0) {         /* dirty solution: must handle part of agrep here */
                if (COUNT && !FILEOUT && !SILENT) {
                        if(!NOFILENAME) fprintf(outfp, "%s: %d\n", CurrentFileName,
totalret);
                        else fprintf(outfp, "%d\n", totalret);
                }
                else if (FILEOUT) {
                        file_out(realfilename);
                }
        }
        return totalret;
}

/* Sets lastfilenumber that needs to be searched: rest must be discarded */
int
process_Y_option(num_files, num_days, fp)
        int     num_files, num_days;
        FILE    *fp;
{
        CHAR arrayend[4];

        last_Y_filenumber = 0;
        if ((num_days <= 0) || (fp == NULL) || (timesindexsize <= 0)) return 0;
        last_Y_filenumber = num_files;
        if (num_days * sizeof(int) >= timesindexsize) return 0;       /* everything will be
within so many days */
        if (fseek(fp, num_days*sizeof(int), 0) == -1) return -1;
        fread(arrayend, 1, 4, fp);
        if ((last_Y_filenumber = (arrayend[0] << 24) | (arrayend[1] << 16) | (arrayend[2]
<< 8) | arrayend[3]) > num_files) last_Y_filenumber = num_files;
        if (last_Y_filenumber == 0) {
                last_Y_filenumber = 1;
                printf("Warning: no files modified in the last %d days were found in the
index.\nSearching only the most recently modified file...\n", num_days);
        }
```

```
        return 0;
}

read_index(indexdir)
char    indexdir[MAXNAME];
{
        char    *home;
        char    s[MAXNAME];
        int     ret;

        if (indexdir[0] == '\0') {
                if ((home = (char *)getenv("HOME")) == NULL) {
                        getcwd(indexdir, MAXNAME-1);
                        fprintf(stderr, "using working-directory '%s' to locate index\n",
indexdir);
                }
                else strncpy(indexdir, home, MAXNAME);
        }
        ret = chdir(indexdir);
        if (getcwd(INDEX_DIR, MAXNAME-1) == NULL) strcpy(INDEX_DIR,
indexdir);
        if (ret < 0) {
                fprintf(stderr, "using working-directory '%s' to locate index\n",
INDEX_DIR);
        }

        sprintf(s, "%s", INDEX_FILE);
        indexfp = fopen(s, "r");
        if(indexfp == NULL) {
                fprintf(stderr, "can't open glimpse index-file %s/%s\n", INDEX_DIR,
INDEX_FILE);
                fprintf(stderr, "(use -H to give an index-directory or run 'glimpseindex' to
make an index)\n");
                return -1;
        }
        if (stat(s, &index_stat_buf) == -1) {
                fprintf(stderr, "can't stat %s/%s\n", INDEX_DIR, s);
                fclose(indexfp);
                return -1;
        }

        sprintf(s, "%s", P_TABLE);
        partfp = fopen(s, "r");
        if(partfp == NULL) {
                fprintf(stderr, "can't open glimpse partition-table %s/%s\n", INDEX_DIR,
P_TABLE);
```

```
                fprintf(stderr, "(use -H to specify an index-directory or run glimpseindex
to make an index)\n");
                fclose(indexfp);
                return -1;
        }

        sprintf(s, "%s", DEF_TIME_FILE);
        timesfp = fopen(s, "r");

        sprintf(s, "%s.index", DEF_TIME_FILE);
        timesindexfp = fopen(s, "r");
        if (timesindexfp != NULL) {
                struct stat st;
                fstat(fileno(timesindexfp), &st);
                timesindexsize = st.st_size;
        }

        /* Get options */
#if     BG_DEBUG
        debug = fopen(DEBUG_FILE, "w+");
        if(debug == NULL) {
                fprintf(stderr, "can't open file %s/%s, errno=%d\n", INDEX_DIR,
DEBUG_FILE, errno);
                return(-1);
        }
#endif  /*BG_DEBUG*/
        fgets(indexnumberbuf, 256, indexfp);
        if(strstr(indexnumberbuf, "1234567890")) IndexNumber = ON;
        else IndexNumber = OFF;
        fscanf(indexfp, "%%%d\n", &OneFilePerBlock);
        if (OneFilePerBlock < 0) {
                ByteLevelIndex = ON;
                OneFilePerBlock = -OneFilePerBlock;
        }
        else if (OneFilePerBlock == 0) {
                GNumpartitions = get_table(P_TABLE, p_table, MAX_PARTITION, 0);
        }
        fscanf(indexfp, "%%%d%s\n", &StructuredIndex, old_rdelim);
        /* Set WHOLEFILESCOPE for do-it-yourself request processing at client */
        WHOLEFILESCOPE = 1;
        if (StructuredIndex <= 0) {
                if (StructuredIndex == -2) {
                        RecordLevelIndex = 1;
                        strcpy(rdelim, old_rdelim);
                        rdelim_len = strlen(rdelim);
                        preprocess_delimiter(rdelim, rdelim_len, rdelim, &rdelim_len);
```

```
                }
                WHOLEFILESCOPE = 0;
                StructuredIndex = 0;
                PRINTATTR = 0;      /* doesn't make sense: must not go into filter_output
*/
        }
        else if (-1 == (StructuredIndex = attr_load_names(ATTRIBUTE_FILE))) {
                fprintf(stderr, "error in reading attribute file %s/%s\n", INDEX_DIR,
ATTRIBUTE_FILE);
                return(-1);
        }
#if     BG_DEBUG
        fprintf(debug, "buf = %s OneFilePerBlock=%d StructuredIndex=%d\n",
indexnumberbuf, OneFilePerBlock, StructuredIndex);
#endif  /*BG_DEBUG*/
        sprintf(s, "%s", MINI_FILE);
        minifp = fopen(s, "r");
        /* if (minifp==NULL && OneFilePerBlock) fprintf(stderr, "Can't open for
reading: %s/%s --- cannot do very fast search\n", INDEX_DIR, MINI_FILE); */
        if (OneFilePerBlock && glimpse_isserver && (minifp != NULL))
read_mini(indexfp, minifp);
        read_filenames();

        /* Once IndexNumber info is available */
        set_indexable_char(indexable_char);
        set_indexable_char(test_indexable_char);
        set_special_char(indexable_char);
        return 0;
}

#define CLEANUP \
{\
        int     q, k;\
        if (timesfp != NULL) fclose(timesfp);\
        if (timesindexfp != NULL) fclose(timesindexfp);\
        if (indexfp != NULL) fclose(indexfp);\
        if (partfp != NULL) fclose(partfp);\
        if (minifp != NULL) fclose(minifp);\
        if (nullfp != NULL) fclose(nullfp);\
        indexfp = partfp = minifp = nullfp = NULL;\
        if (ByteLevelIndex) {\
                if (src_offset_table != NULL) for (k=0; k<OneFilePerBlock; k++) {\
                        free_list(&src_offset_table[k]);\
                }\
                for (q=0; q<MAXNUM_PAT; q++) {\
```

```
                    if (multi_dest_offset_table[q] != NULL) for (k=0; k<OneFilePerBlock;
k++) {\
                        free_list(&multi_dest_offset_table[q][k]);\
                    }\
                }\
        }\
        if (StructuredIndex) {\
                attr_free_table();\
        }\
        destroy_filename_hashtable();\
        my_free(SERV_HOST, MAXNAME);\
}

/* Called whenever we get SIGUSR2/SIGHUP (at the end of process_query()) */
reinitialize_server(argc, argv)
        int     argc;
        char    **argv;
{
        int     i, fd;
        CLEANUP;
#if     0
        init_filename_hashtable();
        region_initialize();
        indexfp = partfp = minifp = nullfp = NULL;
        if ((nullfp = fopen("/dev/null", "w")) == NULL) {
                return(-1);
        }
        src_offset_table = NULL;
        for (i=0; i<MAXNUM_PAT; i++) multi_dest_offset_table[i] = NULL;
        if (-1 == read_index(INDEX_DIR)) return(-1);
#if     0
#ifndef LOCK_UN
#define LOCK_UN    8
#endif
        if ((fd = open(INDEX_DIR, O_RDONLY)) == -1) return -1;
        flock(fd, LOCK_UN);
        close(fd);
#endif
        return 0;
#else
        return execve(argv[0], argv, environ);
#endif
}

/* MUST CARE IF PIPE/SOCKET IS BROKEN! ALSO SIGUSR1
(hardy@cs.colorado.edu) => QUIT CURRENT REQUEST. */
```

```
int ignore_signal[32] = {       0,
                        0, 0, 1, 1, 1, 1, 1, 1,     /* all the tracing stuff: since default action is
to dump core */
                        0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 1, 0, 0 };      /* resource lost: since default action is to
dump core */

/* S.t. sockets don't persist: they sometimes have a bad habit of doing so */
void
cleanup()
{
        int     i;

        /* ^C in the middle of a client call */
        if (svstderr != 2) {
                close(2);
                dup(svstderr);
        }
        fprintf(stderr, "server cleaning up...\n");
        CLEANUP;
        for (i=0; i<64; i++) close(i);
        exit(3);
}

void reinitialize(s)
int s;
{
        /* To force main-while loop call reinitialize_server() after do_select() */
        glimpse_reinitialize = 1;
#ifdef __svr4__
        /* Solaris 2.3 insists that you reset the signal handler */
        (void)signal(s, reinitialize);
#endif
}

#define QUITREQUESTMSG "glimpseserver: aborting request...\n"
/* S.t. one request doesn't keep server occupied too long, when client already quits */
void quitrequest(s)
int s;
{
        /*
         * Don't write onto stderr, since 2 is duped to sockfd => can cause recursive
signal!
         * Also, don't print error message more than once for quitting one request. The
```

```
         * server receives signals for EVERY write it attempts when it finds a match: I
could
         * not find a way to prevent it, but agrep/bitap.c/fill_buf() was fixed to limit it.
         * -- bg on 16th Feb 1995
         */
        if (!glimpse_clientdied && (s != SIGUSR1))          /* USR1 is a "friendly"
cleanup message */
                write(svstderr, QUITREQUESTMSG, strlen(QUITREQUESTMSG));

        glimpse_clientdied = 1;
#ifdef __svr4__
        /* Solaris 2.3 insists that you reset the signal handler */
        (void)signal(s, quitrequest);
#endif
}

/* The client receives this signal when an output/input pipe is broken, etc. It simply exits
from the current request */
void exitrequest()
{
        glimpse_clientdied = 1;
}

main(argc, argv)
int argc;
char *argv[];
{
        int     ret = 0, tried = 0;
        char    indexdir[MAXNAME];
        char    **oldargv = argv;
        int     oldargc = argc;
#if     CLIENTSERVER
        int     sockfd, newsockfd, clilen, len, clpid;
        int     clout;
#if     USE_UNIXDOMAIN
        struct sockaddr_un cli_addr, serv_addr;
#else   /*USE_UNIXDOMAIN*/
        struct sockaddr_in cli_addr, serv_addr;
        struct hostent *hp;
#endif  /*USE_UNIXDOMAIN*/
        int     cli_len;
        int     clargc;
        char    **clargv;
        int     clstdin, clstdout, clstderr;
        int     i;
        char    array[4];
```

```
        char    *p, c;
#endif  /*CLIENTSERVER*/
        int     quitwhile;

#if     ISO_CHAR_SET
        setlocale(LC_ALL,"");       /* support for 8bit character set: ew@senate.be,
Henrik.Martin@eua.ericsson.se */
#endif
#if     CLIENTSERVER && ISSERVER
        glimpse_isserver = 1;  /* I am the server */
#else   /*CLIENTSERVER && ISSERVER*/
        if (argc <= 1) {
                usage();          /* Client nees at least 1 argument */
                exit(1);
        }
#endif  /*CLIENTSERVER && ISSERVER*/

#define RETURNMAIN(val)\
{\
      CLEANUP;\
      if (val < 0) exit (2);\
          else if (val == 0) exit (1);\
      else exit (0);\
}

        SERV_HOST = (CHAR *)my_malloc(MAXNAME);
#if     !SYSCALLTESTING
        /* once-only initialization */
        init_filename_hashtable();
        src_offset_table = NULL;
        for (i=0; i<MAXNUM_PAT; i++) multi_dest_offset_table[i] = NULL;
#endif
        gethostname(SERV_HOST, MAXNAME - 2);
        SERV_PORT = DEF_SERV_PORT;
        srand(getpid());
        umask(077);
        strcpy(&GProgname[0], argv[0]);
#if     !SYSCALLTESTING
        region_initialize();
#endif
        indexfp = partfp = minifp = nullfp = NULL;
        if ((nullfp = fopen("/dev/null", "w")) == NULL) {
                fprintf(stderr, "%s: cannot open for writing: /dev/null, errno=%d\n",
argv[0], errno);
                RETURNMAIN(-1);
        }
```

```
        InterpretSpecial = ON;
        GMAX_WORD_SIZE = MAXPAT;

#if     CLIENTSERVER
#if     !ISSERVER
        /* Install signal handlers so that glimpse doesn't continue to run when pipes get
broken, etc. */
        if (((void (*)())-1 == signal(SIGPIPE, exitrequest))
#ifndef SCO
            || ((void (*)())-1 == signal(SIGURG, exitrequest))
#endif
        )
        {
                /* Check for return values here since they ensure reliability */
                fprintf(stderr, "glimpse: Unable to install signal-handlers.\n");
                RETURNMAIN(-1);
        }

        /* Check if client has too many arguments: then it is surely running as agrep since
I have < half those options! */
        if (argc > MAX_ARGS) goto doityourself;
#endif  /*!ISSERVER*/

#if     !SYSCALLTESTING
        while((--argc > 0) && (*++argv)[0] == '-' ) {
                p = argv[0] + 1;        /* ptr to first character after '-' */
                c = *(argv[0]+1);
                if (*p == '-') {  /* cheesy hack to support --version and --help options */
                        if (*(p+1) == 'v') {
                                c = 'V';
                        } else if (*(p+1) == 'h') {
                                c = '?';
                        }
                }
                quitwhile = OFF;
                while (!quitwhile && (*p != '\0')) {
                        switch(c) {
                        /* Look for -H option at server (only one that makes sense); if
client has a -H, then it goes to doityourself */
                        case 'H' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: a directory name must
follow the -H option\n", GProgname);
                                                RETURNMAIN(usageS());
                                        }
```

```
                                        argv ++;
                                        strcpy(indexdir, argv[0]);
                                        argc --;
                                }
                                else {
                                        strcpy(indexdir, p+1);
                                }
                                quitwhile = ON;
                                break;

                        /* Recognized by both client and server */
                        case 'J' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the server host name
must follow the -J option\n", GProgname);
#if     ISSERVER
                                                RETURNMAIN(usageS());
#else   /*ISSERVER*/
                                                RETURNMAIN(usage());
#endif  /*ISSERVER*/
                                        }
                                        argv ++;
                                        strcpy(SERV_HOST, argv[0]);
                                        argc --;
                                }
                                else {
                                        strcpy(SERV_HOST, p+1);
                                }
                                quitwhile = ON;
                                break;

                        /* Recognized by both client and server */
                        case 'K' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the server port must
follow the -C option\n", GProgname);
#if     ISSERVER
                                                RETURNMAIN(usageS());
#else   /*ISSERVER*/
                                                RETURNMAIN(usage());
#endif  /*ISSERVER*/
                                        }
                                        argv ++;
                                        SERV_PORT = atoi(argv[0]);
```

```
                                argc --;
                        }
                        else {
                                SERV_PORT = atoi(p+1);
                        }
                        if ((SERV_PORT < MIN_SERV_PORT) || (SERV_PORT
> MAX_SERV_PORT)) {
                                fprintf(stderr, "Bad server port %d: must be in [%d,
%d]: using default %d\n",
                                        SERV_PORT, MIN_SERV_PORT,
MAX_SERV_PORT, DEF_SERV_PORT);
                                SERV_PORT = DEF_SERV_PORT;
                        }
                        quitwhile = ON;
                        break;

#if     ISSERVER
#if     SFS_COMPAT
                case 'R' :
                        RemoteFiles = ON;
                        break;

                case 'Z' :
                        /* No op */
                        break;
#endif

                case 'V' :
                        printf("\nThis is glimpseindex version %s, %s.\n\n",
GLIMPSE_VERSION, GLIMPSE_DATE);
                        RETURNMAIN(1);

                case '?' :
                        RETURNMAIN(usageS());

                /* server cannot recognize any other option */
                default :
                        fprintf(stderr, "%s: server cannot recognize option: '%s'\n",
GProgname, p);
                        RETURNMAIN(usageS());
#else   /*ISSERVER*/

                /* These have 1 argument each, so must do quitwhile */
                case 'd' :
                case 'e' :
                case 'f' :
```

```
                case 'k' :
                case 'D' :
                case 'F' :
                case 'I' :
                case 'L' :
                case 'R' :
                case 'S' :
                case 'T' :
                case 'Y' :
                case 'p' :
                        if (argv[0][2] == '\0') {/* space after - option */
                                if(argc <= 1) {
                                        fprintf(stderr, "%s: the '-%c' option must
have an argument\n", GProgname, c);
                                        RETURNMAIN(usage());
                                }
                                argv++;
                                argc--;
                        }
                        quitwhile = ON;
                        break;

                /* These are illegal */
                case 'm' :
                case 'v' :
                        fprintf(stderr, "%s: illegal option: '-%c'\n", GProgname, c);
                        RETURNMAIN(usage());

                /* They can't be patterns and filenames since they start with a -,
these don't have arguments */
                case '!' :
                case 'a' :
                case 'b' :
                case 'c' :
                case 'h' :
                case 'i' :
                case 'j' :
                case 'l' :
                case 'n' :
                case 'o' :
                case 'q' :
                case 'r' :
                case 's' :
                case 't' :
                case 'u' :
                case 'g' :
```

```
                        case 'w' :
                        case 'x' :
                        case 'y' :
                        case 'z' :
                        case 'A' :
                        case 'B' :
                        case 'E' :
                        case 'G' :
                        case 'M' :
                        case 'N' :
                        case 'O' :
                        case 'P' :
                        case 'Q' :
                        case 'U' :
                        case 'W' :
                        case 'X' :
                        case 'Z' :
                                break;

                        case 'C':
                                CONTACT_SERVER = 1;
                                break;
                        case 'V' :
                                printf("\nThis is glimpse version %s, %s.\n\n",
GLIMPSE_VERSION, GLIMPSE_DATE);
                                RETURNMAIN(1);

                        case '?':
                                RETURNMAIN(usage());
                        default :
                                if (isdigit(c)) quitwhile = ON;
                                else {
                                        fprintf(stderr, "%s: illegal option: '-%c'\n",
GProgname, c);
                                        RETURNMAIN(usage());
                                }
                                break;
#endif  /*ISSERVER*/
                        } /* switch(c) */
                        p ++;
                        c = *p;
                }
        }
#else
        CONTACT_SERVER = 1;
        argc=0;
```

```
#endif

#if	!ISSERVER
	/* Next arg must be the pattern: Check if the user wants to run the client as agrep,
or doesn't want to contact the server */
	if ((argc > 1) || (!CONTACT_SERVER)) goto doityourself;
#endif  /*!ISSERVER*/

	argv = oldargv;
	argc = oldargc;
#endif  /*CLIENTSERVER*/

#if	ISSERVER && CLIENTSERVER
	if (-1 == read_index(indexdir)) RETURNMAIN(ret);

	/* Install signal handlers so that glimpseserver doesn't continue to run when
sockets get broken, etc. */
	for (i=0; i<32; i++)
		if (ignore_signal[i]) signal(i, SIG_IGN);
	signal(SIGHUP, cleanup);
	signal(SIGINT, cleanup);
	if (((void (*)())-1 == signal(SIGPIPE, quitrequest)) ||
	   ((void (*)())-1 == signal(SIGUSR1, quitrequest)) ||
#ifndef SCO
	   ((void (*)())-1 == signal(SIGURG, quitrequest)) ||
#endif
	   ((void (*)())-1 == signal(SIGUSR2, reinitialize)) ||
	   ((void (*)())-1 == signal(SIGHUP, reinitialize))) {
		/* Check for return values here since they ensure reliability */
		fprintf(stderr, "glimpseserver: Unable to install signal-handlers.\n");
		RETURNMAIN(-1);
	}

#if	USE_UNIXDOMAIN
	if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
		fprintf(stderr, "server cannot open socket for communication.\n");
		RETURNMAIN(-1);
	}
	char TMP_FILE_NAME[256];
	strcpy(TMP_FILE_NAME,TEMP_DIR) ;
	strcat(TMP_FILE_NAME,"/.glimpse_server");
	unlink(TMP_FILE_NAME);
	memset((char *)&serv_addr, '\0', sizeof(serv_addr));
	serv_addr.sun_family = AF_UNIX;
	strcpy(serv_addr.sun_path, TMP_FILE_NAME);     /* < 108 ! */
	len = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);
```

```
#else   /*USE_UNIXDOMAIN*/
        if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("glimpseserver: Cannot create socket");
                RETURNMAIN(-1);
        }
        memset((char *)&serv_addr, '\0', sizeof(serv_addr));
      serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(SERV_PORT);
#if     0
        /* use host-names not internet style d.d.d.d notation */
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
#else
        /*
         * We only want to accept connections from glimpse clients
         * on the SERV_HOST, do not use INADDR_ANY!
         */
        if ((hp = gethostbyname(SERV_HOST)) == NULL) {
                perror("glimpseserver: Cannot resolve host");
                RETURNMAIN(-1);
        }
        memcpy((caddr_t)&serv_addr.sin_addr, hp->h_addr, hp->h_length);
#endif  /*0*/
        len = sizeof(serv_addr);
#endif  /*USE_UNIXDOMAIN*/
        /* test code for glimpse server, get it to realse socket when it dies: contribution by
Sheldon Smoker <sheldon@thunder.tig.com> */
        if((setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,(char
*)&one,sizeof(one))) == -1) {
                fprintf(stderr,"glimpseserver: could not set socket option\n");
                perror("setsockopt");
                exit(1);
        }
        /* end test code */

        if (bind(sockfd, (struct sockaddr *)&serv_addr, len) < 0) {
                perror("glimpseserver: Cannot bind to socket");
                RETURNMAIN(-1);
        }
        listen(sockfd, SERVER_QUEUE_SIZE);

        printf("glimpseserver: On-line (pid = %d, port = %d) waiting for request...\n",
getpid(), SERV_PORT);
        fflush(stdout); /* must fflush to print on server stdout */
        while (1) {
                /*
                 *  Spin until sockfd is ready to do a non-blocking accept(2).
```

```
                 *  We only wait for 15 seconds, because SunOS may
                 *  swap us out if we block for 20 seconds or more.
                 *  -- Courtesy: Darren Hardy, hardy@cs.colorado.edu
                 */
                if ((ret = do_select(sockfd, 15)) == 0) {
                        if ((errno == EINTR) && glimpse_reinitialize) {
                                glimpse_reinitialize = 0;
                                CLEANUP;
                                close(sockfd);
                                sleep(IC_PORTRELEASE);
                                reinitialize_server(oldargc, oldargv);
                        }
                        continue;
                }
                else if (ret != 1) continue;

                /* get parameters */
                ret = 0;
                clargc = 0;
                clargv = NULL;
                cli_len = sizeof(cli_addr);
                if ((newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &cli_len))
< 0) continue;
                if (getreq(newsockfd, glimpse_reqbuf, &clstdin, &clstdout, &clstderr,
&clargc, &clargv, &clpid) < 0) {
                        ret = -1;
#if      DEBUG
                        printf("getreq errno: %d\n", errno);
#endif  /*DEBUG*/
                        goto end_process;
                }

#if      DEBUG
                printf("server processing request on %x\n", newsockfd);
#endif  /*DEBUG*/
                /*
                 * Server doesn't wait for response, no point using
                svstdin = dup(0);
                close(0);
                dup(clstdin);
                close(clstdin);
                 */
                /*
                 * This is wrong since clstderr == clstdout!
                svstdout = dup(1);
                close(1);
```

```
dup(clstdout);
close(clstdout);
svstderr = dup(2);
close(2);
dup(clstderr);
close(clstderr);
*/
svstdout = dup(1);
svstderr = dup(2);
close(1);
close(2);
dup(clstdout);
dup(clstderr);
close(clstdout);
close(clstderr);

/*
        * IMPORTANT: Unbuffered I/O to the client!
 * Done for Harvest since partial results might be
 * needed and fflush will not flush partial results
 * to the client if we type ^C and kill it: it puts
 * them into /dev/null. This way, output is unbuffered
 * and the client sees at least some results if killed.
 */
setbuf(stdout, NULL);
setbuf(stderr, NULL);

        glimpse_call = 0;
        glimpse_clientdied = 0;
        ret = process_query(clargc, clargv, newsockfd);
/*
 * Server doesn't wait for response, no point using
close(0);
dup(svstdin);
close(svstdin);
svstdin = 0;
 */
if (glimpse_clientdied) {
        /*
         * This code is *ONLY* used as a safety net now.
         * The old problem was that users would see portions
         * of previous (and usually) unrelated queries!
         * glimpseserver now uses unbuffered I/O to the
         * client so all previous fwrite's to now are
         * gone.  But since this is such a nasty problem
         * we flush stdout to /dev/null just in case.
```

```
                         */
                        clout = open("/dev/null", O_WRONLY);
                        close(1);
                        dup(clout);
                        close(clout);
                        fflush(stdout);
                }

                /* Restore svstdout and svstdout to stdout/stderr */
                close(1);
                dup(svstdout);
                close(svstdout);
                svstdout = 1;
                close(2);
                dup(svstderr);
                close(svstderr);
                svstderr = 2;

        end_process:
#if     USE_MSGHDR
                /* send reply and cleanup */
                array[0] = (ret & 0xff000000) >> 24;
                array[1] = (ret & 0xff0000) >> 16;
                array[2] = (ret & 0xff00) >> 8;
                array[3] = (ret & 0xff);
                writen(newsockfd, array, 4);
#endif  /*USE_MSGHDR*/
#if     DEBUG
                write(1, "done\n", 5);
#endif  /*DEBUG*/
                for (i=0; i<clargc; i++)
                        if (clargv[i] != NULL) my_free(clargv[i], 0);
                if (clargv != NULL) my_free(clargv, 0);
                close(newsockfd);       /* if !USE_MSGHDR, client directly reads from
socket and writes onto stdout until EOF */
        }
#else   /*ISSERVER && CLIENTSERVER*/

#if     CLIENTSERVER
trynewsocket:
#if     USE_UNIXDOMAIN
        if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
                perror("socket");
                goto doityourself;
        }
        memset((char *)&serv_addr, '\0', sizeof(serv_addr));
```

```
        serv_addr.sun_family = AF_UNIX;
        char TMP_FILE_NAME[256];
        strcpy(TMP_FILE_NAME,TEMP_DIR) ;
        strcat(TMP_FILE_NAME,"/.glimpse_server");
        strcpy(serv_addr.sun_path, TMP_FILE_NAME);     /* < 108 ! */
        len = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);
#else   /*USE_UNIXDOMAIN*/
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                perror("socket");
                goto doityourself;
        }
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(SERV_PORT);
#if     0
        /* use host-names not internet style d.d.d.d notation */
        serv_addr.sin__addr.s_addr = inet_addr(SERV_HOST);
#else   /*0*/
        if ((hp = gethostbyname(SERV_HOST)) == NULL) {
                fprintf(stderr, "gethostbyname (%s) failed\n", SERV_HOST);
                goto doityourself;
        }
        memcpy((caddr_t)&serv_addr.sin_addr, hp->h_addr, hp->h_length);
#endif  /*0*/
        len = sizeof(serv_addr);
#endif  /*USE_UNIXDOMAIN*/

        if (connect(sockfd, (struct sockaddr *)&serv_addr, len) < 0) {
           char errbuf[4096];
           sprintf(errbuf, "glimpse: Cannot contact glimpseserver: %s, port %d:",
SERV_HOST, SERV_PORT);
           perror(errbuf);
                /* perror(SERV_HOST); */
#if     DEBUG
                printf("connect errno: %d\n", errno);
#endif  /*DEBUG*/
                close(sockfd);
                if ((errno == ECONNREFUSED) && (tried < 4)) {
                        tried ++;
                        goto trynewsocket;
                }
                goto doityourself;
        }

        if (sendreq(sockfd, glimpse_reqbuf, fileno(stdin), fileno(stdout), fileno(stderr),
argc, argv, getpid()) < 0) {
                perror("sendreq");
```

```
#if	DEBUG
		printf("sendreq errno: %d\n", errno);
#endif	/*DEBUG*/
		close(sockfd);
		goto doityourself;
	}

#if	USE_MSGHDR
	if (readn(sockfd, array, 4) != 4) {
		close(sockfd);
		goto doityourself;
	}
	ret = (array[0] << 24) + (array[1] << 16) + (array[2] << 8) + array[3];
#else	/*USE_MSGHDR*/
{
	/*
	 *  Dump everything the server writes into the socket onto
	 *  stdout until EOF/error.  Do this in a way so that *everything*
	 *  the server sends is dumped to stdout by the client.  The
	 *  client might die suddenly via ^C or SIGTERM, but we still
	 *  want the results.
	 */
	char tmpbuf[1024];
	int n;

	while ((n = read(sockfd, tmpbuf, 1024)) > 0) {
		write(fileno(stdout), tmpbuf, n);
	}
}
#endif	/*USE_MSGHDR*/

	close(sockfd);
	RETURNMAIN(ret);

doityourself:
#if	DEBUG
	printf("doing it myself :-(\n");
#endif	/*DEBUG*/
#endif	/*CLIENTSERVER*/
	setbuf(stdout, NULL);/* Unbuffered I/O to always get every result */
	setbuf(stderr, NULL);
	glimpse_call = 0;
	glimpse_clientdied = 0;
	ret = process_query(oldargc, oldargv, fileno(stdin));
	RETURNMAIN(ret);
#endif	/*ISSERVER && CLIENTSERVER*/
```

```
}

process_query(argc, argv, newsockfd)
int argc;
char *argv[];
int newsockfd;
{
        int     searchpercent;
        int     num_blocks;
        int     num_read;
        int     i, j;
        int     iii; /* Udi */
        int     jjj;
        char    c;
        char    *p;
        int     ret;
        int     jj;
        int     quitwhile;
        char    indexdir[MAX_LINE_LEN];
        char    temp_filenames_file[MAX_LINE_LEN];
        char    temp_bitfield_file[MAX_LINE_LEN];
        char    TEMP_FILE[MAX_LINE_LEN];
        char    temp_file[MAX_LINE_LEN];
        int     oldargc = argc;
        char    **oldargv = argv;
        CHAR dummypat[MAX_PAT];
        int     dummylen=0;
        int     my_M_index, my_P_index, my_b_index, my_A_index, my_l_index = -1,
my_B_index = -1;
        char    **outname;
        int     gnum_of_matched = 0;
        int     gprev_num_of_matched = 0;
        int     gfiles_matched = 0;
        int     foundpat = 0;
        int     wholefilescope=0;
        int     nobytelevelmustbeon=0;
        long    get_file_time();

        if ((argc <= 0) || (argv == NULL)) {
                errno = EINVAL;
                return -1;
        }
/*
 * Macro to destroy EVERYTHING before return since we might want to make this a
 * library function later on: convention is that after destroy, objects are made
 * NULL throughout the source code, and are all set to NULL at initialization time.
```

```
 * DO agrep_argv, index_argv and FileOpt my_malloc/my_free optimizations later.
 * my_free calls have 2nd parameter = 0 if the size is not easily determinable.
 */
#define RETURN(val) \
{\
        int     q,k;\
\
        first_search = 0;\
        for (k=0; k<MAX_ARGS; k++) {\
                if (agrep_argv[k] != NULL) my_free(agrep_argv[k], 0);\
                if (index_argv[k] != NULL) my_free(index_argv[k], 0);\
                agrep_argv[k] = index_argv[k] = NULL;\
        }\
        if (FileOpt != NULL) my_free(FileOpt, MAXFILEOPT);\
        FileOpt = NULL;\
        for (k=0; k<MAXNUM_PAT; k++) {\
                if (pat_list[k] != NULL) my_free(pat_list[k], 0);\
                pat_list[k] = NULL;\
        }\
        sprintf(tempfile, "%s/.glimpse_tmp.%d", TEMP_DIR, getpid());\
        unlink(tempfile);\
        sprintf(outname[0], "%s/.glimpse_apply.%d", TEMP_DIR, getpid());\
        unlink(outname[0]);\
        my_free(outname[0], 0);\
        my_free(outname, 0);\
        my_free(TEMP_DIR, MAX_LINE_LEN);\
        my_free(filenames_file, MAX_LINE_LEN);\
        my_free(bitfield_file, MAX_LINE_LEN);\
\
        if (ByteLevelIndex) {\
                if (src_offset_table != NULL) for (k=0; k<OneFilePerBlock; k++) {\
                        free_list(&src_offset_table[k]);\
                }\
                /* Don't make src_offset_table itself NULL: it will be bzero-d below if
!NULL */\
                for (q=0; q<MAXNUM_PAT; q++) {\
                   if (multi_dest_offset_table[q] != NULL) for (k=0; k<OneFilePerBlock;
k++) {\
                        free_list(&multi_dest_offset_table[q][k]);\
                   }\
                   /* Don't make multi_dest_offset_table[q] itself NULL: it will be bzero-
d below if !NULL */\
                }\
        }\
        for (k=0; k<num_terminals;k++)\
                free(terminals[k].data.leaf.value);\
```

```
	if (ComplexBoolean) destroy_tree(&GParse);\
	for (k=0; k<GNumfiles; k++) {\
		my_free(GTextfiles[k], 0);\
		GTextfiles[k] = NULL;\
	}\
	/* Don't free the GTextfiles buffer itself since it is allocated once in
get_filename.c */\
	return (val);\
}

	/*
	 * Initialize
	 */
	strcpy(&GProgname[0], argv[0]);
	if (argc <= 1) return(usage());
	filenames_file = (char *)my_malloc(MAX_LINE_LEN);
	bitfield_file = (char *)my_malloc(MAX_LINE_LEN);
	TEMP_DIR = (char *)my_malloc(MAX_LINE_LEN);
	strcpy(TEMP_DIR, "/tmp");
	D_length = 0;
	D = 0;
	pattern_index = 0;
	first_search = 1;
	outname  = (char **)my_malloc(sizeof(char *));
	outname[0] = (char *)my_malloc(MAX_LINE_LEN);
	NOBYTELEVEL = 0;
	OPTIMIZEBYTELEVEL = 0;
	GCONSTANT = 0;
	GLIMITOUTPUT = 0;
	GLIMITTOTALFILE = 0;
	GBESTMATCH = 0;
	GRECURSIVE = 0;
	GNOPROMPT = 1;
	GBYTECOUNT = 0;
	GPRINTFILENUMBER = 0;
	GPRINTFILETIME = 0;
	GOUTTAIL = 2;		/* stupid fix, but works */
	GFILENAMEONLY = 0;
	GNOFILENAME = 0;
	GPRINTNONEXISTENTFILE = 0;
	MATCHFILE = 0;
	PRINTATTR = 0;
	PRINTINDEXLINE = 0;
	Pat_as_is=0;
	Only_first = 1;
	PRINTAPPXFILEMATCH = 1;
```

```
        GCOUNT = 1;
        HINTSFROMUSER = 0;
        FILENAMESINFILE=0;
        BITFIELDFILE=0;
        BITFIELDOFFSET=0;
        BITFIELDLENGTH=0;
        BITFIELDENDIAN=0;
        GNumDays = 0;
        foundattr = 0;
        foundnot = 0;
        ComplexBoolean = 0;
        bestmatcherrors = 0;
        patbufpos = -1;
        RegionLimit=DEFAULT_REGION_LIMIT;
        strcpy(GD_pattern, "\n");
        GD_length = strlen(GD_pattern);
        indexdir[0] = '\0';
        memset(index_argv, '\0', sizeof(char *) * MAX_ARGS);
        index_argc = 0;
        memset(agrep_argv, '\0', sizeof(char *) * MAX_ARGS);
        agrep_argc = 0;
        FileOpt = NULL;
        fileopt_length = 0;
        memset(pat_list, '\0', sizeof(char *) * MAXNUM_PAT);
        memset(pat_attr, '\0', sizeof(int) * MAXNUM_PAT);
        for (i=0; i<MAX_ARGS; i++)
                index_argv[i] = (char *)my_malloc(MaxNameLength + 2);
        memset(is_mgrep_pat, '\0', sizeof(int) * MAXNUM_PAT);
        memset(mgrep_pat_index, '\0', sizeof(int) *MAXNUM_PAT);
        num_mgrep_pat = 0;
        memset(pat_buf, '\0', (MAXNUM_PAT + 2)*MAXPAT);
        pat_ptr = 0;
        sprintf(tempfile, "%s/.glimpse_tmp.%d", TEMP_DIR, getpid());
        /* Set WHOLEFILESCOPE for per-request processing at server */
        if (StructuredIndex) WHOLEFILESCOPE = 1;
        else WHOLEFILESCOPE = 0;
        last_Y_filenumber = 0;

        if (argc > MAX_ARGS) {
#if     ISSERVER
        fprintf(stderr, "too many arguments %d obtained on server!\n", argc);
#endif  /*ISSERVER*/
                i = fileagrep(oldargc, oldargv, 0, stdout);
                RETURN(i);
        }
```

```
        /*
         * Process what options you can, then call fileagrep_init() to set
         * options in agrep and get the pattern. Then, call fileagrep_search().
         * Begin by copying options into agrep_argv assuming glimpse was not
         * called as agrep (optimistic :-).
         */

        agrep_argc = 0;
        for (i=0; i<MAX_ARGS; i++) agrep_argv[i] = NULL;
        agrep_argv[agrep_argc] = (char *)my_malloc(strlen(argv[0]) + 2);
        strcpy(agrep_argv[agrep_argc], argv[0]);      /* copy the name of the program
anyway */
        agrep_argc ++;

        /* In glimpse, you should never output filenames with zero matches */
        if (agrep_argc + 1 >= MAX_ARGS) {
                fprintf(stderr, "%s: too many options!\n", GProgname);
                RETURN(usage());
        }
        agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
        agrep_argv[agrep_argc][0] = '-';
        agrep_argv[agrep_argc][1] = 'z';
        agrep_argv[agrep_argc][2] = '\0';
        agrep_argc ++;

        /* In glimpse, you should always print pattern when using mgrep (user can't do -f
or -m)! */
        if (agrep_argc + 1 >= MAX_ARGS) {
                fprintf(stderr, "%s: too many options!\n", GProgname);
                RETURN(usage());
        }
        agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
        agrep_argv[agrep_argc][0] = '-';
        agrep_argv[agrep_argc][1] = 'P';
        agrep_argv[agrep_argc][2] = '\0';
        my_P_index = agrep_argc;
        agrep_argc ++;

        /* In glimpse, you should always output multiple when doing mgrep */
        if (agrep_argc + 1 >= MAX_ARGS) {
                fprintf(stderr, "%s: too many options!\n", GProgname);
                RETURN(usage());
        }
        agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
        agrep_argv[agrep_argc][0] = '-';
        agrep_argv[agrep_argc][1] = 'M';
```

```
agrep_argv[agrep_argc][2] = '\0';
my_M_index = agrep_argc;
agrep_argc ++;

/* In glimpse, you should print the byte offset if there is a structured query */
if (agrep_argc + 1 >= MAX_ARGS) {
        fprintf(stderr, "%s: too many options!\n", GProgname);
        RETURN(usage());
}
agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
agrep_argv[agrep_argc][0] = '-';
agrep_argv[agrep_argc][1] = 'b';
agrep_argv[agrep_argc][2] = '\0';
my_b_index = agrep_argc;
agrep_argc ++;

/* In glimpse, you should always have space for doing -m if required */
if (agrep_argc + 2 >= MAX_ARGS) {
        fprintf(stderr, "%s: too many options!\n", GProgname);
        RETURN(usage());
}
agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
agrep_argv[agrep_argc][0] = '-';
agrep_argv[agrep_argc][1] = 'm';
agrep_argv[agrep_argc][2] = '\0';
agrep_argc ++;
agrep_argv[agrep_argc] = (char *)my_malloc(2);     /* no op */
agrep_argv[agrep_argc][0] = '\0';
agrep_argc ++;

/* Add -A option to print filenames as default */
if (agrep_argc + 1 >= MAX_ARGS) {
        fprintf(stderr, "%s: too many options!\n", GProgname);
        RETURN(usage());
}
agrep_argv[agrep_argc] = (char *)my_malloc(sizeof(char *));
agrep_argv[agrep_argc][0] = '-';
agrep_argv[agrep_argc][1] = 'A';
agrep_argv[agrep_argc][2] = '\0';
my_A_index = agrep_argc;
agrep_argc ++;

while((agrep_argc < MAX_ARGS) && (--argc > 0) && (*++argv)[0] == '-' ) {
        p = argv[0] + 1;          /* ptr to first character after '-' */
        c = *(argv[0]+1);
        quitwhile = OFF;
```

```
		while (!quitwhile && (*p != '\0')) {
			c = *p;
			switch(c) {
			case 'F' :
				MATCHFILE = ON;
				FileOpt = (CHAR *)my_malloc(MAXFILEOPT);
				if (*(p + 1) == '\0') {/* space after - option */
					if(argc <= 1) {
						fprintf(stderr, "%s: a file pattern must follow
the -F option\n", GPrognname);
						RETURN(usage());
					}
					argv++;
					if ((dummylen = strlen(argv[0])) > MAXFILEOPT)
{
						fprintf(stderr, "%s: -F option list too long\n",
GPrognname);
						RETURN(usage());
					}
					strcpy(FileOpt, argv[0]);
					argc--;
				} else {
					if ((dummylen = strlen(p+1)) > MAXFILEOPT) {
						fprintf(stderr, "%s: -F option list too long\n",
GPrognname);
						RETURN(usage());
					}
					strcpy(FileOpt, p+1);
				} /* else */
				quitwhile = ON;
				break;

			/* search the index only and output the number of blocks */
			case 'N' :
				Only_first = ON;
				break ;

			/* also keep track of the matches in each file */
			case 'Q' :
				PRINTAPPXFILEMATCH = ON;
				break ;

			case 'U' :
				InfoAfterFilename = ON;
				break;
```

```
                        case '!' :
                                HINTSFROMUSER = ON;
                                break;

                        /* go to home directory to find the index: even if server overwrites
indexdir here, it won't overwrite INDEX_DIR until read_index() */
                        case 'H' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: a directory name must
follow the -H option\n", GProgname);
                                                RETURN(usage());
                                        }
                                        argv ++;
#if     !ISSERVER
                                        strcpy(indexdir, argv[0]);
#endif  /*!ISSERVER*/
                                        argc --;
                                }
#if     !ISSERVER
                                else {
                                        strcpy(indexdir, p+1);
                                }
                                agrep_argv[agrep_argc] = (char *)my_malloc(4);
                                strcpy(agrep_argv[agrep_argc], "-H");
                                agrep_argc ++;
                                agrep_argv[agrep_argc] = (char
*)my_malloc(strlen(indexdir) + 2);
                                strcpy(agrep_argv[agrep_argc], indexdir);
                                agrep_argc ++;
#endif  /*!ISSERVER*/
                                quitwhile = ON;
                                break;

#if     ISSERVER && SFS_COMPAT
                        /* INDEX_DIR will be already set since this is the server, so we
can direclty xfer the .glimpse_* files */
                        case '.' :
                                strcpy(TEMP_FILE, INDEX_DIR);
                                strcpy(temp_file, ".");
                                strcat(TEMP_FILE, "/.");
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: a file name must follow
the -. option\n", GProgname);
                                                RETURN(usage());
```

```
                                        }
                                        argv ++;
                                        strcat(TEMP_FILE, argv[0]);
                                        strcat(temp_file, argv[0]);
                                        argc --;
                                }
                                else {
                                        strcat(TEMP_FILE, p+1);
                                        strcat(temp_file, p+1);
                                }
                                if (!strcmp(temp_file, INDEX_FILE) || !strcmp(temp_file,
FILTER_FILE) ||
                                   !strcmp(temp_file, ATTRIBUTE_FILE) ||
!strcmp(temp_file, MINI_FILE) ||
                                   !strcmp(temp_file, P_TABLE) || !strcmp(temp_file,
PROHIBIT_LIST) ||
                                   !strcmp(temp_file, INCLUDE_LIST) ||
!strcmp(temp_file, NAME_LIST) ||
                                   !strcmp(temp_file, NAME_LIST_INDEX) ||
!strcmp(temp_file, NAME_HASH) ||
                                   !strcmp(temp_file, NAME_HASH_INDEX) ||
!strcmp(temp_file, DEF_STAT_FILE) ||
                                   !strcmp(temp_file, DEF_MESSAGE_FILE) ||
!strcmp(temp_file, DEF_TIME_FILE)) {
                                        if ((ret = open(TEMP_FILE, O_RDONLY, 0)) <=
0) RETURN(ret);
                                        while ((num_read = read(ret, matched_region,
MAX_REGION_LIMIT*2)) > 0) {
                                                write(1 /* NOT TO newsockfd since that
was got by a syscall!!! */, matched_region, num_read);
                                        }
                                        close(ret);
                                }
                                quitwhile = ON;
                                RETURN(0);
#endif  /* ISSERVER */

                        /* go to temp directory to create temp files */
                        case 'T' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: a directory name must
follow the -T option\n", GProgname);
                                                RETURN(usage());
                                        }
                                        argv ++;
```

```
                                        strcpy(TEMP_DIR, argv[0]);
                                        argc --;
                                }
                                else {
                                        strcpy(TEMP_DIR, p+1);
                                }
                                sprintf(tempfile, "%s/.glimpse_tmp.%d", TEMP_DIR,
getpid());
                                quitwhile = ON;
                                break;

                        /* To get files within some number of days before indexing was
done */
                        case 'Y':
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the number of days must
follow the -Y option\n", GProgname);
                                                RETURN(usage());
                                        }
                                        argv ++;
                                        GNumDays = atoi(argv[0]);
                                        argc --;
                                }
                                else {
                                        GNumDays = atoi(p+1);
                                }
                                if (GNumDays <= 0) {
                                        fprintf(stderr, "%s: the number of days %d must be
> 0\n", GProgname, GNumDays);
                                        RETURN(usage());
                                }
                                quitwhile = ON;
                                break;

                        case 'R' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the record size must
follow the -R option\n", GProgname);
                                                RETURN(usage());
                                        }
                                        argv ++;
                                        RegionLimit = atoi(argv[0]);
                                        argc --;
                                }
```

```
                                else {
                                        RegionLimit = atoi(p+1);
                                }
                                if ((RegionLimit <= 0) || (RegionLimit >
MAX_REGION_LIMIT)) {
                                        fprintf(stderr, "Bad record size %d: must be in [%d,
%d]: using default %d\n",
                                                RegionLimit, 1, MAX_REGION_LIMIT,
DEFAULT_REGION_LIMIT);
                                        RegionLimit = DEFAULT_REGION_LIMIT;
                                }
                                quitwhile = ON;
                                break;

                        /* doesn't matter if we overwrite the value in the client since the
same value would have been picked up in main() anyway */
                        case 'J' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the server host name
must follow the -J option\n", GProgname);
                                                RETURN(usageS());
                                        }
                                        argv ++;
#if     !ISSERVER
                                        strcpy(SERV_HOST, argv[0]);
#endif  /*!ISSERVER*/
                                        argc --;
                                }
#if     !ISSERVER
                                else {
                                        strcpy(SERV_HOST, p+1);
                                }
#endif  /*!ISSERVER*/
                                quitwhile = ON;
                                break;

                        /* doesn't matter if we overwrite the value in the client since the
same value would have been picked up in main() anyway */
                        case 'K' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the server port must
follow the -C option\n", GProgname);
                                                RETURN(usage());
                                        }
```

```
                                        argv ++;
#if     !ISSERVER
                                        SERV_PORT = atoi(argv[0]);
#endif  /*!ISSERVER*/
                                        argc --;
                                }
#if     !ISSERVER
                                else {
                                        SERV_PORT = atoi(p+1);
                                }
                                if ((SERV_PORT < MIN_SERV_PORT) || (SERV_PORT
> MAX_SERV_PORT)) {
                                        fprintf(stderr, "Bad server port %d: must be in [%d,
%d]: using default %d\n",
                                                SERV_PORT, MIN_SERV_PORT,
MAX_SERV_PORT, DEF_SERV_PORT);
                                        SERV_PORT = DEF_SERV_PORT;
                                }
#endif  /*!ISSERVER*/
                                quitwhile = ON;
                                break;

                        /* Based on contribution From ada@mail2.umu.se Fri Jul 12 01:56
MST 1996; Christer Holgersson, Sen. SysNet Mgr, Umea University/SUNET, Sweden */
                        /* the bit-mask corresponding to the set of filenames within which
the pattern should be searched is explicitly provided in a filename (absolute path name) */
                        case 'p' :
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if (argc <= 1) {
                                                fprintf(stderr, "%s: the bitfield file [and an
offset/length/endian separated by :] must follow the -p option\n", GProgname);
                                                RETURN(usage());
                                        }
                                        argv ++;
                                        strcpy(bitfield_file, argv[0]);
                                        argc --;
                                }
                                else {
                                        strcpy(bitfield_file, p+1);
                                }
                                /* Find offset and length into bitfield file */
                                {
                                        int iiii = 0;

                                        BITFIELDOFFSET=0;
                                        BITFIELDLENGTH=0;
```

```
                                        BITFIELDENDIAN=0;
                                        iiii = 0;
                                        while (bitfield_file[iiii] != '\0') {
                                                if (bitfield_file[iiii] == '\\') {
                                                        iiii ++;
                                                        if (bitfield_file[iiii] == '\0') break;
                                                        if (bitfield_file[iiii] == ':') {
                                                                strcpy(&bitfield_file[iiii-1],
&bitfield_file[iiii]);
                                                        }
                                                        else iiii ++;
                                                        continue;
                                                }
                                                if (bitfield_file[iiii] == ':') {
                                                        bitfield_file[iiii] = '\0';
                                                        sscanf(&bitfield_file[iiii+1],
"%d:%d:%d", &BITFIELDOFFSET, &BITFIELDLENGTH, &BITFIELDENDIAN);
                                                        if ((BITFIELDOFFSET < 0) ||
(BITFIELDLENGTH < 0) || (BITFIELDENDIAN < 0)) {
                                                                fprintf(stderr, "Wrong offset
%d or length %d or endian %d of bitfield file\n", BITFIELDOFFSET,
BITFIELDLENGTH, BITFIELDENDIAN);
                                                                RETURN(usage());
                                                        }
                                                        break;
                                                }
                                                iiii++;
                                        }
#if     BG_DEBUG
                                        fprintf(debug, "BITFIELD %s : %d : %d : %d\n",
BITFIELDFILE, BITFIELDOFFSET, BITFIELDLENGTH, BITFIELDENDIAN);
#endif
                                }
                                if (bitfield_file[0] != '/') {
                                        getcwd(temp_bitfield_file, MAX_LINE_LEN-1);
                                        strcat(temp_bitfield_file, "/");
                                        strcat(temp_bitfield_file, bitfield_file);
                                        strcpy(bitfield_file, temp_bitfield_file);
                                }
                                BITFIELDFILE = 1;
                                quitwhile = ON;
                                break;

                        /* the set of filenames within which the pattern should be searched
is explicitly provided in a filename (absolute path name) */
                        case 'f' :
```

```
                        if (*(p + 1) == '\0') {/* space after - option */
                                if (argc <= 1) {
                                        fprintf(stderr, "%s: the filenames file must
follow the -f option\n", GProgname);
                                        RETURN(usage());
                                }
                                argv ++;
                                strcpy(filenames_file, argv[0]);
                                argc --;
                        }
                        else {
                                strcpy(filenames_file, p+1);
                        }
                        if (filenames_file[0] != '/') {
                                getcwd(temp_filenames_file, MAX_LINE_LEN-1);
                                strcat(temp_filenames_file, "/");
                                strcat(temp_filenames_file, filenames_file);
                                strcpy(filenames_file, temp_filenames_file);
                        }
                        FILENAMESINFILE = 1;
                        quitwhile = ON;
                        break;

                case 'C' :
                        CONTACT_SERVER = 1;
                        break;

                case 'a' :
                        PRINTATTR = 1;
                        break;

                case 'E':
                        PRINTINDEXLINE = 1;
                        break;

                case 'W':
                        wholefilescope = 1;
                        break;

                case 'z' :
                        UseFilters = 1;
                        break;

                case 'r' :
                        GRECURSIVE = 1;
                        break;
```

```
                case 'V' :
                        printf("\nThis is glimpse version %s, %s.\n\n",
GLIMPSE_VERSION, GLIMPSE_DATE);
                        RETURN(0);

                /* Must let 'm' fall thru to default once multipatterns are done in
agrep */
                case 'm' :
                case 'v' :
                        fprintf(stderr, "%s: illegal option: '-%c'\n", GProgname, c);
                        RETURN(usage());

                case 'I' :
                case 'D' :
                case 'S' :
                        /* There is no space after these options */
                        agrep_argv[agrep_argc] = (char
*)my_malloc(strlen(argv[0]) + 2);
                        agrep_argv[agrep_argc][0] = '-';
                        strcpy(agrep_argv[agrep_argc] + 1, p);
                        agrep_argc ++;
                        quitwhile = ON;
                        break;

                case 'l':
                        GFILENAMEONLY = 1;
                        my_l_index = agrep_argc;
                        agrep_argv[agrep_argc] = (char *)my_malloc(4);
                        agrep_argv[agrep_argc][0] =  '-';
                        agrep_argv[agrep_argc][1] = c;
                        agrep_argv[agrep_argc][2] = '\0';
                        agrep_argc ++;
                        break;

                /*
                 * Copy the set of options for agrep: put them in separate argvs
                 * even if they are together after one '-' (easier to process).
                 * These are agrep options which glimpse has to peek into.
                 */
                default:
                        agrep_argv[agrep_argc] = (char *)my_malloc(16);
                        agrep_argv[agrep_argc][0] =  '-';
                        agrep_argv[agrep_argc][1] = c;
                        agrep_argv[agrep_argc][2] = '\0';
                        agrep_argc ++;
```

```
                        if (c == 'n') {
                                nobytelevelmustbeon=1;
                        }
                        else if (c == 'X') GPRINTNONEXISTENTFILE = 1;
                        else if (c == 'j') GPRINTFILETIME = 1;
                        else if (c == 'b') GBYTECOUNT = 1;
                        else if (c == 'g') GPRINTFILENUMBER = 1;
                        else if (c == 't') GOUTTAIL = 1;
                        else if (c == 'y') GNOPROMPT = 1;
                        else if (c == 'h') GNOFILENAME = 1;
                        else if (c == 'c') GCOUNT = 1;
                        else if (c == 'B') {
                                GBESTMATCH = 1;
                                my_B_index = agrep_argc - 1;
                        }
                        /* the following options are followed by a parameter */
                        else if ((c == 'e') || (c == 'd') || (c == 'L') || (c == 'k')) {
                                if (*(p + 1) == '\0') {/* space after - option */
                                        if(argc <= 1) {
                                                fprintf(stderr, "%s: the '-%c' option
must have an argument\n", GProgname, c);
                                                RETURN(usage());
                                        }
                                        argv++;
                                        if ( (c == 'd') && ((D_length =
strlen(argv[0])) > MAX_NAME_SIZE) ) {
                                                fprintf(stderr, "%s: delimiter pattern
too long (has > %d chars)\n", GProgname, MAX_NAME_SIZE);
                                                RETURN(usage());
                                                /* Should this be RegionLimit if
ByteLevelIndex? */
                                        }
                                        else if (c == 'L') {
                                                GLIMITOUTPUT =
GLIMITTOTALFILE = GLIMITPERFILE = 0;
                                                sscanf(argv[0], "%d:%d:%d",
&GLIMITOUTPUT, &GLIMITTOTALFILE, &GLIMITPERFILE);
                                                if ((GLIMITOUTPUT < 0) ||
(GLIMITTOTALFILE < 0) || (GLIMITPERFILE < 0)) {
                                                        fprintf(stderr, "%s: invalid
output limit %s\n", GProgname, argv[0]);
                                                        RETURN(usage());
                                                }
                                        }
```

```
                                        agrep_argv[agrep_argc] = (char
*)my_malloc(strlen(argv[0]) + 2);
                                        strcpy(agrep_argv[agrep_argc], argv[0]);
                                        if (c == 'd') {
                                                preprocess_delimiter(argv[0],
D_length, GD_pattern, &GD_length);
                                                if (GOUTTAIL == 2) GOUTTAIL =
0;
                                                /* Should this be RegionLimit if
ByteLevelIndex? */
                                        }
                                        if (c == 'k') GCONSTANT = 1;
                                        argc--;
                                } else {
                                        if ( (c == 'd') && ((D_length = strlen(p+1))
> MAX_NAME_SIZE) ) {
                                                fprintf(stderr, "%s: delimiter pattern
too long (has > %d chars)\n", GProgname, MAX_NAME_SIZE);
                                                RETURN(usage());
                                                /* Should this be RegionLimit if
ByteLevelIndex? */
                                        }
                                        else if (c == 'L') {
                                                GLIMITOUTPUT =
GLIMITTOTALFILE = GLIMITPERFILE = 0;
                                                sscanf(p+1, "%d:%d:%d",
&GLIMITOUTPUT, &GLIMITTOTALFILE, &GLIMITPERFILE);
                                                if ((GLIMITOUTPUT < 0) ||
(GLIMITTOTALFILE < 0) || (GLIMITPERFILE < 0)) {
                                                        fprintf(stderr, "%s: invalid
output limit %s\n", GProgname, p+1);
                                                        RETURN(usage());
                                                }
                                        }
                                        agrep_argv[agrep_argc] = (char
*)my_malloc(strlen(p+1) + 2);
                                        strcpy(agrep_argv[agrep_argc], p+1);
                                        if (c == 'd') {
                                                preprocess_delimiter(p+1, D_length-
2, GD_pattern, &GD_length);
                                                if (GOUTTAIL == 2) GOUTTAIL =
0;
                                                /* Should this be RegionLimit if
ByteLevelIndex? */
                                        }
                                        if (c == 'k') GCONSTANT = 1;
```

```
                                        }
                                        agrep_argc ++;
#if     DEBUG
                                        fprintf(stderr, "%d = %s\n", agrep_argc,
agrep_argv[agrep_argc - 1]);
#endif  /*DEBUG*/
                                        quitwhile = ON;
                                        if ((c == 'e') || (c == 'k')) foundpat = 1;
                                }
                                /* else it is something that glimpse doesn't know and agrep
needs to look at */

                                break;  /* from default: */

                        } /* switch(c) */
                        p ++;
                }
        } /* while (--argc > 0 && (*++argv)[0] == '-') */

/* exitloop: */
        if ((GBESTMATCH == ON) && (MATCHFILE == ON) && (Only_first ==
ON))
                fprintf(stderr, "%s: Warning: the number of matches may be incorrect
when -B is used with -F.\n", HARVEST_PREFIX);

        if (GOUTTAIL) GOUTTAIL = 1;

        if (GNOFILENAME) {
                agrep_argv[my_A_index][1] = 'Z';    /* ignore the -A option */
        }

#if     ISSERVER
        if (RemoteFiles) {      /* force -NQ so that won't start looking for files! */
                Only_first = ON;
                PRINTAPPXFILEMATCH = ON;
        }
#endif

        if (argc > 0) {
                /* copy the rest of the options the pattern and the filenames if any
verbatim */
                for (i=0; i<argc; i++) {
                        if (agrep_argc >= MAX_ARGS) break;
                        agrep_argv[agrep_argc] = (char *)my_malloc(strlen(argv[0]) + 2);
                        strcpy(agrep_argv[agrep_argc], argv[0]);
                        agrep_argc ++;
```

```
                        argv ++; }
                if (!foundpat) argc --;
        }

#if     0
        for (j=0; j<agrep_argc; j++) printf("agrep_argv[%d] = %s\n", j, agrep_argv[j]);
        printf("argc = %d\n", argc);
#endif  /*0*/

        /*
         * Now perform the search by first looking at the index
         * and obtaining the files to search; and then search
         * them and output the result. If argc > 0, glimpse
         * runs as agrep: otherwise, it searches index, etc.
         */

        if (argc <= 0) {
                if (RecordLevelIndex) {         /* based on work done for
robint@zedcor.com Robin Thomas, Art Today, Tucson, AZ */
                        /*
                        if ((D_length > 0) && strcmp(GD_pattern, rdelim)) {
                                fprintf(stderr, "Index created for delimiter `%s': cannot
search with delimiter `%s'\n", rdelim, GD_pattern);
                                RETURN(-1);
                        }
                        SHOULD I HAVE THIS CHECK? MAYBE GD_pattern is a
SUBSTRING OF rdelim??? But this is safest thing to do... robint@zedcor.com
                        */
                        RegionLimit = 0;        /* region is EXACTLY the same record
number, not a portion of a file within some offset+length */
                }
                glimpse_call = 1;
                /* Initialize some data structures, read the index */
                if (GRECURSIVE == 1) {
                        fprintf(stderr, "illegal option: '-r'\n");
                        RETURN(usage());
                }
                num_terminals = 0;
                GParse = NULL;
                memset(terminals, '\0', sizeof(ParseTree) * MAXNUM_PAT);
#if     !ISSERVER
                if (-1 == read_index(indexdir)) RETURN(-1);
#endif  /*!ISSERVER*/

/*
```

```
This handles the -n option with ByteLevelIndex: disabled as of now, else should go into
file search...
*/
                if (nobytelevelmustbeon && (ByteLevelIndex && !RecordLevelIndex)) {
        /* with RecordLevelIndex, we'll do search, so don't set NOBYTELEVEL */
                        /* fprintf(stderr, "Warning: -n option used with byte-level index:
must SEARCH the files\n"); */
                        NOBYTELEVEL=ON;
                }

                WHOLEFILESCOPE = (WHOLEFILESCOPE || wholefilescope);

                if (ByteLevelIndex) {
                        /* Must zero them here in addition to index search so that
RETURN macro runs correctly */
                        if ((src_offset_table == NULL) &&
                           ((src_offset_table = (struct offsets **)my_malloc(sizeof(struct
offsets *) * OneFilePerBlock)) == NULL)) exit(2);
                        memset(src_offset_table, '\0', sizeof(struct offsets *) *
OneFilePerBlock);
                        for (i=0; i<MAXNUM_PAT; i++) {
                                if ((multi_dest_offset_table[i] == NULL) &&
                                   ((multi_dest_offset_table[i] = (struct offsets
**)my_malloc(sizeof(struct offsets *) * OneFilePerBlock)) == NULL)) exit(2);
                                memset(multi_dest_offset_table[i], '\0', sizeof(struct offsets
*) * OneFilePerBlock);
                        }
                }
                read_filters(INDEX_DIR, UseFilters);

                if (glimpse_clientdied) RETURN(0);
                /* Now initialize agrep, set the options and get the actual pattern into
GPattern */
                if ((GM = fileagrep_init(agrep_argc, agrep_argv, MAXPAT, GPattern))
<= 0) {
                        /* this printf need not be there: agrep prints messages if error */
                        fprintf(stderr, "%s: error in options or arguments to `agrep'\n",
HARVEST_PREFIX);
                        RETURN(usage());
                }
                patindex = pattern_index;
                for (j=0; j<GM; j++) {
                        if (GPattern[j] == '\\') j++;
                        else if (test_indexable_char[GPattern[j]]) break;
                }
                if (j >= GM) {
```

```
                        fprintf(stderr, "%s: pattern '%s' has no characters that were
indexed: glimpse cannot search for it\n", HARVEST_PREFIX, GPattern);
                        for (j=0; j<GM; j++) {
                                if (GPattern[j] == '\\') j++;
                                else if (isdigit(GPattern[j])) break;
                        }
                        if (j < GM) {
                                fprintf(stderr, "\t(to search for numbers, make the index
using 'glimpseindex -n ...')\n");
                        }
                        RETURN(-1);
                }

                /* Split GPattern into individual boolean terms */
                if (GCONSTANT) {
                        strcpy(APattern, GPattern);
                        GParse = NULL;
                        ComplexBoolean = 0;
                        terminals[0].op = 0;
                        terminals[0].type = LEAF;
                        terminals[0].terminalindex = 0;
                        terminals[0].data.leaf.attribute = 0;
                        terminals[0].data.leaf.value = (CHAR *)malloc(GM + 1);
                        strcpy(terminals[0].data.leaf.value, GPattern);
                        num_terminals = 1;
                }
                else if (split_pattern(GPattern, GM, APattern, terminals, &num_terminals,
&GParse, StructuredIndex) <= 0) RETURN(-1);
#if     BG_DEBUG
                fprintf(debug, "GPattern = %s, APattern = %s, num_terminals = %d\n",
GPattern, APattern, num_terminals);
#endif  /*BG_DEBUG*/

                /* Set scope for booleans */
                if (foundnot && !wholefilescope) {
                        fprintf(stderr, "To use the NOT (~) operation, you must use whole
file scope (-W) for booleans.\n");
                        RETURN(usage())
                }
                if (foundattr) WHOLEFILESCOPE = 1;      /* makes no sense to search
attribute=value expressions without WHOLEFILESCOPE */
                else if (!ComplexBoolean && !PRINTATTR && !((long)GParse &
AND_EXP)) WHOLEFILESCOPE = 0;       /* ORs can be done without
WHOLEFILESCOPE */
                if (WHOLEFILESCOPE <= 0) agrep_argv[my_b_index][1] = 'Z';
/*
```

```
		if (!ComplexBoolean && ((long)GParse & AND_EXP) && (my_l_index
!= -1)) agrep_argv[my_l_index][1] = 'Z';
*/
		if ((ComplexBoolean || ((long)GParse & AND_EXP)) && (my_l_index !=
-1)) agrep_argv[my_l_index][1] = 'Z';

		/* Now re-initialize agrep_argv with APattern instead of GPattern */
		my_free(agrep_argv[patindex], 0);
		AM=strlen(APattern);
		agrep_argv[patindex] = (char *)my_malloc(AM + 2);
		strcpy(agrep_argv[patindex], APattern);

		if (HINTSFROMUSER) {
			int	num=0, x, y, i, j;
			char	temp[MAX_NAME_SIZE+2];
			struct offsets *o, *tailo, *headO;

			while(1) {
				if ((num = readline(newsockfd, dest_index_buf,
REAL_INDEX_BUF)) <= 0) {
					fprintf(stderr, "Input format error with -U
option\n");
					RETURN(-1);
				}
				dest_index_buf[num+1] = '\n';
				if (!strncmp(dest_index_buf, "BEGIN", strlen("BEGIN")))
break;
			}
			sscanf(&dest_index_buf[strlen("BEGIN")], "%d%d%d",
&bestmatcherrors, &NOBYTELEVEL, &OPTIMIZEBYTELEVEL);
			/* printf("BEGIN %d %d %d\n", bestmatcherrors,
NOBYTELEVEL, OPTIMIZEBYTELEVEL);	*/
			num = readline(newsockfd, dest_index_buf,
REAL_INDEX_BUF);
			while (num > 0) {
				dest_index_buf[num+1] = '\n';
				if (!strncmp(dest_index_buf, "END", strlen("END")))
break;
				i = j = 0;
				while ((j<MAX_NAME_SIZE) && (dest_index_buf[i] !=
FILE_END_MARK) && (dest_index_buf[i] != '[') && (dest_index_buf[i] != '\n'))
					temp[j++] = dest_index_buf[i++];
				temp[j] = '\0';
				x = atoi(temp);
				GFileIndex[GNumfiles] = x;
				if (x == file_num - 1) {
```

```
                                        bigbuffer[bigbuffer_size] = '\0';
                                        GTextfiles[GNumfiles++] = (CHAR
*)strdup(GTextfilenames[x]);
                                        bigbuffer[bigbuffer_size] = '\n';
                                }
                                else {
                                        *(GTextfilenames[x+1] - 1) = '\0';
                                        GTextfiles[GNumfiles++] = (CHAR
*)strdup(GTextfilenames[x]);
                                        *(GTextfilenames[x+1] - 1) = '\n';
                                }
                                /* printf("%d %s [", x, GTextfiles[GNumfiles-1]); */
                                src_index_set[block2index(x)] |= block2mask(x);
                                if (ByteLevelIndex && !NOBYTELEVEL) {           /*
NOBYTELEVEL is 0 here with RecordLevelIndex */
                                        heado = tailo = NULL;
                                onemorey:
                                        j = 0;
                                        while ((j<MAX_NAME_SIZE) &&
((dest_index_buf[i] == FILE_END_MARK) || (dest_index_buf[i] == '['))) i++;
                                        while ((j<MAX_NAME_SIZE) &&
(dest_index_buf[i] != FILE_END_MARK) && (dest_index_buf[i] != '\n') &&
(dest_index_buf[i] != ']'))
                                                temp[j++] = dest_index_buf[i++];
                                        temp[j] = '\0';
                                        y = atoi(temp);
                                        /* printf(" %d", y); */
                                        o = (struct offsets *)my_malloc(sizeof(struct
offsets));
                                        o->offset = y;
                                        o->next = NULL;
                                        o->sign = o->done = 0;
                                        if (heado == NULL) {
                                                heado = o;
                                                tailo = o;
                                        }
                                        else {
                                                tailo->next = o;
                                                tailo = o;
                                        }
                                        if (dest_index_buf[i] == FILE_END_MARK) goto
onemorey;
                                        src_offset_table[x] = heado;
                                }
                                /* printf("]\n"); */
```

```
                                num = readline(newsockfd, dest_index_buf,
REAL_INDEX_BUF);
                        }
                        goto search_files;
                }

                /*
                 * Copy the agrep-options that are relevant to index search into
                 * index_argv (see man-pages for which options are relevant).
                 * Also, adjust patindex whenever options are skipped over.
                 * NOTE: agrep_argv does NOT contain two options after one '-'.
                 */
                index_argc = 0;
                for (j=0; j<agrep_argc; j++) {
                        if (agrep_argv[j][0] == '-') {
                                if ((agrep_argv[j][1] == 'c') || (agrep_argv[j][1] == 'h') ||
(agrep_argv[j][1] == 'l') || (agrep_argv[j][1] == 'n') ||
                                    (agrep_argv[j][1] == 's') || (agrep_argv[j][1] == 't') ||
(agrep_argv[j][1] == 'G') || (agrep_argv[j][1] == 'O') ||
                                    (agrep_argv[j][1] == 'b') || (agrep_argv[j][1] == 'i') ||
(agrep_argv[j][1] == 'u') || (agrep_argv[j][1] == 'g') ||
                                    (agrep_argv[j][1] == 'E') || (agrep_argv[j][1] == 'Z') ||
(agrep_argv[j][1] == 'j') || (agrep_argv[j][1] == 'X')) {
                                    patindex --;
                                    continue;
                                }
                                if ((agrep_argv[j][1] == 'd') || (agrep_argv[j][1] == 'L')) {
        /* skip over the argument too */
                                    j++;
                                    patindex -= 2;
                                    continue;
                                }
                                if ((agrep_argv[j][1] == 'e') || (agrep_argv[j][1] == 'm')) {
                                        strcpy(index_argv[index_argc], agrep_argv[j]);
                                        index_argc ++; j++;
                                        strcpy(index_argv[index_argc], agrep_argv[j]);
                                        if (agrep_argv[j-1][1] == 'm') patbufpos =
index_argc;     /* where to put the patbuf if fast-boolean by mgrep() */
                                        index_argc ++;
                                }
                                else {  /* No arguments: just copy THAT option: maybe,
change some options */
                                        strcpy(index_argv[index_argc], agrep_argv[j]);
                                        if (agrep_argv[j][1] == 'A')
index_argv[index_argc][1] = 'h';
```

```
                                        else if (agrep_argv[j][1] == 'x')
index_argv[index_argc][1] = 'w';
                                        index_argc++;
                                }
                        }
                        else {  /* This is either the pattern itself or a filename */
                                strcpy(index_argv[index_argc], agrep_argv[j]);
                                index_argc++;
                        }
                }
                sprintf(index_argv[index_argc], "%s", INDEX_FILE);
                index_argc ++;
#if     0
                for (j=0; j<index_argc; j++) printf("index_argv[%d] = %s\n", j,
index_argv[j]);
                printf("patindex = %d\n", patindex);
#endif  /*0*/

                /* process -Y option BEFORE search_index() so that get_set() doesn't
even have to collect data for very old files */
                ret = process_Y_option(file_num, GNumDays, timesindexfp);

                /* Search the index and process index-search-only options; Worry about
file-pattern */
                ret = search_index(GParse);
                if ((ret <= 0) && (!OneFilePerBlock ||
(src_index_set[REAL_PARTITION - 1] != 1))) RETURN(-1);       /* previously was: if
ret < 0 then return... */
                num_blocks=0;
                if (OneFilePerBlock) {
                        if (ByteLevelIndex || RecordLevelIndex) {

                        if (src_offset_table)   /* Don't iterate if null */
                        for(iii=0; iii<OneFilePerBlock; iii++) {
                                if (src_offset_table[iii] != NULL) num_blocks ++;
                        }
                        /* printf("num_blocks = %d\n", num_blocks); */
                        }
                        if (src_index_set[REAL_PARTITION - 1] == 1) {
                                num_blocks = OneFilePerBlock;
                                for(iii=0; iii<round(OneFilePerBlock, 8*sizeof(int)) - 1;
iii++) {
                                        src_index_set[iii] = 0xffffffff;
                                }
                                src_index_set[iii] = 0;
                                for (jjj=0; jjj<8*sizeof(int); jjj++) {
```

```
                                        if (iii*8*sizeof(int) + jjj >= OneFilePerBlock)
break;
                                        src_index_set[iii] |= mask_int[jjj];
                                }
                        }
                        else for(iii=0; iii<round(OneFilePerBlock, 8*sizeof(int)); iii++) {
                                if (src_index_set[iii] == 0) continue;
                                for (jjj=0; jjj < 8*sizeof(int); jjj++)
                                        if (src_index_set[iii] & mask_int[jjj])
                                                if (!ByteLevelIndex || NOBYTELEVEL) {
                                                        num_blocks ++;
                                                }
                                                else {
                                                        if (src_offset_table[iii*8*sizeof(int)
+ jjj] != NULL)
                                                                num_blocks ++;
                                                        else src_index_set[iii] &=
~mask_int[jjj];
                                                }
                        }
                        if (num_blocks > OneFilePerBlock) num_blocks =
OneFilePerBlock;        /* roundoff */
                }
                else {
                        for (iii=0; iii<MAX_PARTITION; iii++)
                                if (src_index_set[iii]) num_blocks++;
                }
                if (num_blocks <= 0) RETURN (0);
                if ((src_index_set[REAL_PARTITION - 1] == 1) && !Only_first &&
!OPTIMIZEBYTELEVEL) {
                        fprintf(stderr, "Warning: pattern has words present in the stop-list:
must SEARCH the files\n");
                }
                if (RecordLevelIndex && GFILENAMEONLY && veryfast &&
(src_index_set[REAL_PARTITION - 1] != 1)) Only_first = 1;
                /* if just the NOBYTELEVEL flag is set, then it is an optimization which
glimpse does and user need not be warned */
#if     DEBUG
                fprintf(stderr, "--> search=%d optimize=%d times=%d all=%d blocks=%d
len=%d pat=%s scope=%d\n",
                        NOBYTELEVEL, OPTIMIZEBYTELEVEL,
src_index_set[REAL_PARTITION - 2], src_index_set[REAL_PARTITION - 1],
num_blocks, strlen(APattern), APattern, WHOLEFILESCOPE);
#endif  /*DEBUG*/
```

```
		/* Based on contribution From ada@mail2.umu.se Fri Jul 12 01:56 MST
1996; Christer Holgersson, Sen. SysNet Mgr, Umea University/SUNET, Sweden */
		if (BITFIELDFILE) {
			int i, len = -1, nextchar;
			FILE	*fp;
			fp = fopen(bitfield_file, "r");
			if (fp != NULL) {
				if (BITFIELDENDIAN >= 2) {	/* is a BIG-ENDIAN
4B integer list of indexes of files in .glimpse_filenames (sparse set) */
					if (BITFIELDLENGTH == 0) BITFIELDLENGTH
= file_num;
					if (BITFIELDOFFSET > 0) fseek(fp,
BITFIELDOFFSET, (long)0);
					if (OneFilePerBlock) {
						for (i=0; i<round(file_num, 8*sizeof(int));
i++)
							multi_dest_index_set[0][i] = 0;
					}
					else {
						for (i=0; i<MAX_PARTITION; i++)
							multi_dest_index_set[0][i] = 0;
					}
					i = -1;
					while ((nextchar = getc(fp)) != EOF) {
						nextchar = nextchar & 0xff;
						i = nextchar << 24;
						if ((nextchar = getc(fp)) == EOF) break;
						nextchar = nextchar & 0xff;
						i |= nextchar << 16;
						if ((nextchar = getc(fp)) == EOF) break;
						nextchar = nextchar & 0xff;
						i |= nextchar << 8;
						if ((nextchar = getc(fp)) == EOF) break;
						nextchar = nextchar & 0xff;
						i |= nextchar;

						if (OneFilePerBlock) {
							if (i < file_num)
multi_dest_index_set[0][block2index(i)] |= mask_int[i%(8*sizeof(int))];
						}
						else {
							if (i < MAX_PARTITION)
multi_dest_index_set[0][i] = 1;
						}
					}
					fclose(fp);
```

```
                                        if (i == -1) {
                                                fprintf(stderr, "Error in reading %d bytes
from offset %d in bitfield file %s ... ignoring it\n", BITFIELDOFFSET,
BITFIELDLENGTH, bitfield_file);
                                                /* ignore index_file */
                                        }
                                        else {  /* intersect files in index_file with those that
were obtained after pattern search */
                                                if (OneFilePerBlock) {
                                                        for (i=0; i<round(file_num,
8*sizeof(int)); i++)
                                                                src_index_set[i] &=
multi_dest_index_set[0][i];
                                                }
                                                else {
                                                        for (i=0; i<MAX_PARTITION; i++)
                                                                src_index_set[i] &=
multi_dest_index_set[0][i];
                                                }
                                        }
                                }
                                else {
                                        if (BITFIELDLENGTH == 0) BITFIELDLENGTH
= sizeof(int)*REAL_PARTITION /* sizeof(int)*FILEMASK_SIZE */;
                                        if (BITFIELDOFFSET > 0) fseek(fp,
BITFIELDOFFSET, (long)0);
                                        if (OneFilePerBlock) {
                                                for (i=0; i<round(file_num, 8*sizeof(int));
i++)
                                                        multi_dest_index_set[0][i] = 0;
                                        }
                                        else {
                                                for (i=0; i<MAX_PARTITION; i++)
                                                        multi_dest_index_set[0][i] = 0;
                                        }
                                        i = 0;
                                        while ((i < BITFIELDLENGTH) && (nextchar =
getc(fp)) != EOF) {
                                                nextchar = nextchar & 0xff;
                                                if (OneFilePerBlock) {
                                                        if (BITFIELDENDIAN == 1) {
        /* little-endian: little end of integer was dumped first in bitfield_file */
                                                                multi_dest_index_set[0][i/4]
|= (nextchar << (8*(i%4)));
                                                        }
```

```
                                                else if (BITFIELDENDIAN == 0) {
        /* big-endian: big end of integer is first was dumped first in bitfield_file */
                                                        multi_dest_index_set[0][i/4]
|= (nextchar << (8*(4-1-(i%4))));
                                                }
                                        }
                                        else {
                                                if (i < MAX_PARTITION) { /*
interpretation of "bit" changes without OneFilePerBlock */
                                                        multi_dest_index_set[0][i] =
(nextchar != 0) ? 1 : 0;
                                                }
                                                else break;     /*
BITFIELDLENGTH, by above definition, is always > MAX_PARTITION: see io.c */
                                        }
                                        i++;
                                }
                                fclose(fp);
                                if (i <= 0) {
                                        fprintf(stderr, "Error in reading %d bytes
from offset %d in bitfield file %s ... ignoring it\n", BITFIELDOFFSET,
BITFIELDLENGTH, bitfield_file);
                                        /* ignore bitfield_file */
                                }
                                else {  /* intersect files in bitfield_file with those
that were obtained after pattern search */
                                        if (OneFilePerBlock) {
                                                for (i=0; i<round(file_num,
8*sizeof(int)); i++)
                                                        src_index_set[i] &=
multi_dest_index_set[0][i];
                                        }
                                        else {
                                                for (i=0; i<MAX_PARTITION; i++)
                                                        src_index_set[i] &=
multi_dest_index_set[0][i];
                                        }
                                }
                        }
                }
        }

        if (FILENAMESINFILE) mask_filenames(src_index_set, filenames_file,
file_num, num_blocks);        /* keep only those files that are in filenames_file */
        if (BITFIELDFILE || FILENAMESINFILE) {
                num_blocks=0;
```

```
                        if (OneFilePerBlock) {
                                for(iii=0; iii<round(OneFilePerBlock, 8*sizeof(int)); iii++)
{
                                        if (src_index_set[iii] == 0) continue;
                                        for (jjj=0; jjj < 8*sizeof(int); jjj++)
                                                if (src_index_set[iii] & mask_int[jjj])
                                                        if (!ByteLevelIndex ||
NOBYTELEVEL) {
                                                                num_blocks ++;
                                                        }
                                                        else {
                                                                if
(src_offset_table[iii*8*sizeof(int) + jjj] != NULL)
                                                                        num_blocks ++;
                                                                else src_index_set[iii] &=
~mask_int[jjj];
                                                        }
                                        }
                                        if (num_blocks > OneFilePerBlock) num_blocks =
OneFilePerBlock;       /* roundoff */
                        }
                        else {
                                for (iii=0; iii<MAX_PARTITION; iii++)
                                        if (src_index_set[iii]) num_blocks++;
                        }
                        if (num_blocks <= 0) RETURN (0);
                }

                dummypat[0] = '\0';
                if (!MATCHFILE)     {         /* the argc,argv don't matter */
                        get_filenames(src_index_set, 0, NULL, dummylen, dummypat,
file_num);

                        if (Only_first) { /* search the index only */
                                /*fprintf(stderr, "There are matches to %d out of %d %s\n",
num_blocks, (OneFilePerBlock > 0) ? OneFilePerBlock : GNumpartitions,
(OneFilePerBlock > 0) ? "files" : "blocks");*/
                                if (num_blocks > 0) {
                                        char     cc[8];
                                        cc[0] = 'y';
#if       !ISSERVER
                                        if (!GNOPROMPT) {
                                                fprintf(stderr, "Do you want to see the file
names? (y/n)");
                                                fgets(cc, 4, stdin);
                                        }
```

```
#endif  /*!ISSERVER*/
                                        if (!SILENT && (cc[0] == 'y')) {
                                                if (PRINTAPPXFILEMATCH &&
Only_first && GPRINTFILENUMBER) {
                                                        printf("BEGIN %d %d %d\n",
bestmatcherrors, NOBYTELEVEL, OPTIMIZEBYTELEVEL);
                                                }
                                                for (jjj=0; jjj<GNumfiles; jjj++) {
                                                        /*printf("**********I AM HERE
FIRST**********");*/
                                                        if ((GLIMITOUTPUT > 0) && (jjj
>= GLIMITOUTPUT)) break;
                                                        if (ByteLevelIndex &&
!NOBYTELEVEL && (src_index_set[REAL_PARTITION - 1] != 1) &&
(src_offset_table[GFileIndex[jjj]] == NULL)) continue;
                                                        /*if (GPRINTFILENUMBER)
printf("%d", GFileIndex[jjj]);
                                                        else printf("%s", GTextfiles[jjj]);*/
                                                        if (PRINTAPPXFILEMATCH) {
                                                           if (GCOUNT) {
                                                                int n = 0;
                                                                /*printf(": ");*/
                                                                if (ByteLevelIndex &&
(src_offset_table != NULL)) {
                                                                        struct offsets  *p1 =
src_offset_table[GFileIndex[jjj]];
                                                                        while (p1 != NULL) {
                                                                                n ++;
                                                                                p1 = p1->next;
                                                                        }
                                                                }
                                                                else n = 1;       /* there is
atleast 1 match */
                                                                printf("%d", n);
                                                           }
                                                           else {
                                                                /*printf(" [");*/
                                                                if (ByteLevelIndex &&
(src_offset_table != NULL)) {
                                                                        struct offsets  *p1 =
src_offset_table[GFileIndex[jjj]];
                                                                        while (p1 != NULL) {
                                                                                /*printf(" %d",
p1->offset);*/
                                                                                p1 = p1->next;
                                                                        }
```

```
                                                }
                                                /*printf("]");*/
                                            }
                                        }
                                        /*printf("\n");*/
                                    }
                                    if (PRINTAPPXFILEMATCH &&
Only_first && GPRINTFILENUMBER) {
                                        printf("END\n");
                                    }
                                }
                            }
                            RETURN(0);
                        } /* end of Only_first */
                        if (!OneFilePerBlock) searchpercent =
num_blocks*100/GNumpartitions;
                        else searchpercent = num_blocks * 100 / OneFilePerBlock;
#if     BG_DEBUG
                        fprintf(debug, "searchpercent = %d, num_blocks = %d\n",
searchpercent, num_blocks);
#endif  /*BG_DEBUG*/
#if     !ISSERVER
                        if (!GNOPROMPT && (searchpercent >
MAX_SEARCH_PERCENT)) {
                            char    cc[8];
                            cc[0] = 'y';
                            fprintf(stderr, "Your first query may search about %d%%
of the total space! Continue? (y/n)", searchpercent);
                            fgets(cc, 4, stdin);
                            if (cc[0] != 'y') RETURN(0);
                        }
                        if (ByteLevelIndex && !RecordLevelIndex && (searchpercent >
DEF_MAX_INDEX_PERCENT)) NOBYTELEVEL = 1;   /* with RecordLevelIndex, I
don't just want to stop collecting offsets just because searchpercent > .... */
#endif  /*!ISSERVER*/
                } /* end of !MATCHFILE */
                else {   /* set up the right options for -F in index_argv/index_argc itself
since they will no longer be used */
                        index_argc=0;
                        strcpy(index_argv[0], GProgname);

                        /* adding the -h option, which is safer for -F */
                        index_argc ++;
                        index_argv[index_argc][0] =  '-';
                        index_argv[index_argc][1] = 'h';
                        index_argv[index_argc][2] = '\0';
```

```
                        index_argc ++;

                        /* new code: bgopal, Feb/8/94: deleted udi's code here */
                        j = 0;
                        while (FileOpt[j] == '-') {
                                j++;
                                while ((FileOpt[j] != ' ') && (FileOpt[j] != '\0') &&
(FileOpt[j] != '\n')) {
                                        if (j >= MAX_ARGS - 1) {
                                                fprintf(stderr, "%s: too many options after -
F: %s\n", GProgname, FileOpt);
                                                RETURN(usage());
                                        }
                                        index_argv[index_argc][0] =  '-';
                                        index_argv[index_argc][1] = FileOpt[j];
                                        index_argv[index_argc][2] = '\0';
                                        index_argc ++;
                                        j++;
                                }
                                if ((FileOpt[j] == '\0') || (FileOpt[j] == '\n')) break;
                                if ((FileOpt[j] == ' ') && (FileOpt[j-1] == '-')) {
                                        fprintf(stderr, "%s: illegal option: '-' after -F\n",
GProgname);
                                        RETURN(usage());
                                }
                                else if (FileOpt[j] == ' ') while(FileOpt[j] == ' ') j++;
                        }
                        while(FileOpt[j] == ' ') j++;

                        fileopt_length = strlen(FileOpt);
                        strncpy(index_argv[index_argc],FileOpt+j,fileopt_length-j);
                        index_argv[index_argc][fileopt_length-j] = '\0';
                        index_argc++;
                        my_free(FileOpt, MAXFILEOPT);
                        FileOpt = NULL;

#if     BG_DEBUG
                        fprintf(debug, "pattern to check with -F =
%s\n",index_argv[index_argc-1]);
#endif  /*BG_DEBUG*/
#if     DEBUG
                        fprintf(stderr, "-F : ");
                        for (jj=0; jj < index_argc; jj++)
                                fprintf(stderr, " %s ",index_argv[jj]);
                        fprintf(stderr, "\n");
#endif  /*DEBUG*/
```

```
                        fflush(stdout);
                        get_filenames(src_index_set, index_argc, index_argv, dummylen,
dummypat, file_num);

                        /* Assume #files per partitions is appx constant */
                        if (OneFilePerBlock) num_blocks = GNumfiles;
                        else num_blocks = GNumfiles * GNumpartitions /
p_table[GNumpartitions - 1];
                        if (Only_first) { /* search the index only */
                                fprintf(stderr, "There are matches to %d out of %d %s\n",
num_blocks, (OneFilePerBlock > 0) ? OneFilePerBlock : GNumpartitions,
(OneFilePerBlock > 0) ? "files" : "blocks");
                                if (num_blocks > 0) {
                                        char    cc[8];
                                        cc[0] = 'y';
#if      !ISSERVER
                                        if (!GNOPROMPT) {
                                                fprintf(stderr, "Do you want to see the file
names? (y/n)");
                                                fgets(cc, 4, stdin);
                                        }
#endif  /*!ISSERVER*/
                                        if (!SILENT && (cc[0] == 'y')) {
                                                if (PRINTAPPXFILEMATCH &&
Only_first && GPRINTFILENUMBER) {
                                                        printf("BEGIN %d %d %d\n",
bestmatcherrors, NOBYTELEVEL, OPTIMIZEBYTELEVEL);
                                                }
                                                for (jjj=0; jjj<GNumfiles; jjj++) {
                                                        printf("**********I AM HERE
SECOND**********");
                                                        if ((GLIMITOUTPUT > 0) && (jjj
>= GLIMITOUTPUT)) break;
                                                        if (ByteLevelIndex &&
!NOBYTELEVEL && (src_index_set[REAL_PARTITION - 1] != 1) &&
(src_offset_table[GFileIndex[jjj]] == NULL)) continue;
                                                        if (GPRINTFILENUMBER)
printf("%d", GFileIndex[jjj]);
                                                        else printf("%s", GTextfiles[jjj]);
                                                        if (PRINTAPPXFILEMATCH) {
                                                           if (GCOUNT) {
                                                                int n = 0;
                                                                printf(": ");
                                                                if (ByteLevelIndex &&
(src_offset_table != NULL)) {
```

```
                                                struct offsets  *p1 =
src_offset_table[GFileIndex[jjj]];
                                                while (p1 != NULL) {
                                                        n ++;
                                                        p1 = p1->next;
                                                }
                                        }
                                        else n = 1;     /* there is
atleast 1 match */
                                        printf("%d", n);
                                }
                                else {
                                        printf("[");
                                        if (ByteLevelIndex &&
(src_offset_table != NULL)) {
                                                struct offsets  *p1 =
src_offset_table[GFileIndex[jjj]];
                                                while (p1 != NULL) {
                                                        printf(" %d",
p1->offset);
                                                        p1 = p1->next;
                                                }
                                        }
                                        printf("]");
                                }
                        }
                        printf("\n");
                }
                if (PRINTAPPXFILEMATCH &&
Only_first && GPRINTFILENUMBER) {
                        printf("END\n");
                }
            }
        }
        RETURN(0);
    } /* end of Only_first */
    if (OneFilePerBlock) searchpercent = GNumfiles * 100 /
OneFilePerBlock;
    else searchpercent = GNumfiles * 100 / p_table[GNumpartitions -
1];
#if     BG_DEBUG
    fprintf(debug, "searchpercent = %d, num_files = %d\n",
searchpercent, p_table[GNumpartitions - 1]);
#endif  /*BG_DEBUG*/
#if     !ISSERVER
```

```
                        if (!GNOPROMPT && (searchpercent >
MAX_SEARCH_PERCENT)) {
                                char    cc[8];
                                cc[0] = 'y';
                                fprintf(stderr, "Your second query may search about
%d%% of the total space! Continue? (y/n)", searchpercent);
                                fgets(cc, 4, stdin);
                                if (cc[0] != 'y') RETURN(0);
                        }
                        if (ByteLevelIndex && !RecordLevelIndex && (searchpercent >
DEF_MAX_INDEX_PERCENT)) NOBYTELEVEL = 1;   /* with RecordLevelIndex, I
don't just want to stop collecting offsets just because searchpercent > .... */
#endif  /*!ISSERVER*/
                }

                /* At this point, I have the set of files to search */

        search_files:
                /* Replace -B by the number of errors if best-match */
                if (GBESTMATCH && (my_B_index >= 0)) {
                        sprintf(&agrep_argv[my_B_index][1], "%d", bestmatcherrors);
#if      BG_DEBUG
                        fprintf(debug, "Changing -B to -%d\n", bestmatcherrors);
#endif  /*BG_DEBUG*/
                }
                agrep_argv[my_M_index][1] = 'Z';
                agrep_argv[my_P_index][1] = 'Z';
#if      0
                for (iii=0; iii<agrep_argc; iii++) fprintf(stdout, "'%s' ", agrep_argv[iii]);
                fprintf(stdout, "\n");
#endif
/*
                if (!ComplexBoolean && ((long)GParse & AND_EXP) && (my_l_index
!= -1) && !WHOLEFILESCOPE) agrep_argv[my_l_index][1] = 'l';
*/
                if ((ComplexBoolean || ((long)GParse & AND_EXP)) && (my_l_index !=
-1) && !WHOLEFILESCOPE) agrep_argv[my_l_index][1] = 'l';

                if (GNumfiles <= 0) RETURN(0);
                if (glimpse_clientdied) RETURN(0);
                /* must reinitialize since the above agrep calls for index-search ruined the
real options: it is required EVEN IF ByteLevelIndex */
                AM = fileagrep_init(agrep_argc, agrep_argv, MAXPAT, APattern);
                /* do actual search with postfiltering if structured query */
                if (WHOLEFILESCOPE <= 0) {
                        if (!UseFilters) {
```

```
                        if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL) {
                                for (i=0; i<GNumfiles; i++) {
                                        /* if (RecordLevelIndex &&
(src_offset_table[GFileIndex[i]] == NULL)) continue; */
                                        gprev_num_of_matched =
gnum_of_matched;
                                        SetCurrentFileName = 1;
                                        if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                        else strcpy(CurrentFileName, GTextfiles[i]);
                                        if (GPRINTFILETIME) {
                                                SetCurrentFileTime = 1;
                                                CurrentFileTime =
get_file_time(timesfp, NULL, GTextfiles[i], GFileIndex[i]);
                                        }
                                        if ((ret = fileagrep_search(AM, APattern, 1,
&GTextfiles[i], 0, stdout)) > 0) {
                                                gnum_of_matched += ret;
                                                gfiles_matched ++;
                                        }
                                        SetCurrentFileName = 0;
                                        if (GPRINTFILETIME) SetCurrentFileTime
= 0;
                                        if (GLIMITOUTPUT > 0) {
                                                if (GLIMITOUTPUT <=
gnum_of_matched) break;
                                                LIMITOUTPUT =
GLIMITOUTPUT - gnum_of_matched;
                                        }
                                        if (GLIMITTOTALFILE > 0) {
                                                if (GLIMITTOTALFILE <=
gfiles_matched) break;
                                                LIMITTOTALFILE =
GLIMITTOTALFILE - gfiles_matched;
                                        }
                                        if ((ret < 0) && (errno ==
AGREP_ERROR)) break;
                                        if (glimpse_clientdied) break;
                                        fflush(stdout);
                                }
                        }
                        else {
                                for (i=0; i<GNumfiles; i++) {
                                        gprev_num_of_matched =
gnum_of_matched;
```

```
                                        SetCurrentFileName = 1;
                                        if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                        else strcpy(CurrentFileName, GTextfiles[i]);
                                        if (my_stat(GTextfiles[i], &file_stat_buf) ==
-1) {
                                                if (GPRINTNONEXISTENTFILE)
printf("%s\n", CurrentFileName);
                                                continue;
                                        }
                                        if (GPRINTFILETIME) {
                                                SetCurrentFileTime = 1;
                                                CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                        }
                                        if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
                                                /* fprintf(stderr, "Warning: file
modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                                first_search = 1;
                                                if ((ret = fileagrep_search(AM,
APattern, 1, &GTextfiles[i], 0, stdout)) > 0) {
                                                        gnum_of_matched += ret;
                                                        gfiles_matched ++;
                                                }
                                        }
                                        else if ((ret = glimpse_search(AM, APattern,
GD_length, GD_pattern, GTextfiles[i], GTextfiles[i], GFileIndex[i], src_offset_table,
stdout)) > 0) {
                                                gnum_of_matched += ret;
                                                gfiles_matched ++;
                                        }
                                        SetCurrentFileName = 0;
                                        if (GPRINTFILETIME) SetCurrentFileTime
= 0;
                                        if (GLIMITOUTPUT > 0) {
                                                if (GLIMITOUTPUT <=
gnum_of_matched) break;
                                                LIMITOUTPUT =
GLIMITOUTPUT - gnum_of_matched;
                                        }
                                        if (GLIMITTOTALFILE > 0) {
                                                if (GLIMITTOTALFILE <=
gfiles_matched) break;
```

```
                                                LIMITTOTALFILE =
GLIMITTOTALFILE - gfiles_matched;
                                        }
                                        if ((ret < 0) && (errno ==
AGREP_ERROR)) break;
                                        if (glimpse_clientdied) break;
                                        fflush(stdout);
                                }
                        }
                } /* end of !UseFilters */
                else {
                        sprintf(outname[0], "%s/.glimpse_apply.%d", TEMP_DIR,
getpid());
                        for (i=0; i<GNumfiles; i++) {
                                /* if (RecordLevelIndex &&
(src_offset_table[GFileIndex[i]] == NULL)) continue; */
                                if (apply_filter(GTextfiles[i], outname[0]) == 1) {
                                        gprev_num_of_matched =
gnum_of_matched;
                                        SetCurrentFileName = 1;
                                        if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                        else strcpy(CurrentFileName, GTextfiles[i]);
                                        if (my_stat(GTextfiles[i], &file_stat_buf) ==
-1) {
                                                if (GPRINTNONEXISTENTFILE)
printf("%s\n", CurrentFileName);
                                                continue;
                                        }
                                        if (GPRINTFILETIME) {
                                                SetCurrentFileTime = 1;
                                                CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                        }
                                        if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL || (file_stat_buf.st_mtime > index_stat_buf.st_mtime)) {
                                                first_search = 1;
                                                if ((ret = fileagrep_search(AM,
APattern, 1, outname, 0, stdout)) > 0) {
                                                        gnum_of_matched += ret;
                                                        gfiles_matched ++;
                                                }
                                        }
                                        else {
                                                if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
```

```
                                                        /* fprintf(stderr, "Warning:
file modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                                        first_search = 1;
                                                        if ((ret =
fileagrep_search(AM, APattern, 1, outname, 0, stdout)) > 0) {
                                                                gnum_of_matched +=
ret;
                                                                gfiles_matched ++;
                                                        }
                                                }
                                                else if ((ret = glimpse_search(AM,
APattern, GD_length, GD_pattern, GTextfiles[i], outname[0], GFileIndex[i],
src_offset_table, stdout)) > 0) {
                                                        gfiles_matched ++;
                                                        gnum_of_matched += ret;
                                                }
                                        }
                                        SetCurrentFileName = 0;
                                        if (GPRINTFILETIME) SetCurrentFileTime
= 0;
                                }
                                else {
                                        if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL) {
                                                first_search = 1;
                                                if ((ret = fileagrep_search(AM,
APattern, 1, &GTextfiles[i], 0, stdout)) > 0) {
                                                        gnum_of_matched += ret;
                                                        gfiles_matched ++;
                                                }
                                        }
                                        else {
                                                SetCurrentFileName = 1;
                                                if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                                else strcpy(CurrentFileName,
GTextfiles[i]);
                                                if (my_stat(GTextfiles[i],
&file_stat_buf) == -1) {
                                                        if
(GPRINTNONEXISTENTFILE) printf("%s\n", CurrentFileName);
                                                        continue;
                                                }
                                                if (GPRINTFILETIME) {
```

```
                                                SetCurrentFileTime = 1;
                                                CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                        }
                                        if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
                                                /* fprintf(stderr, "Warning:
file modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                                first_search = 1;
                                                if ((ret =
fileagrep_search(AM, APattern, 1, &GTextfiles[i], 0, stdout)) > 0) {
                                                        gnum_of_matched +=
ret;
                                                        gfiles_matched ++;
                                                }
                                        }
                                        else if ((ret = glimpse_search(AM,
APattern, GD_length, GD_pattern, GTextfiles[i], GTextfiles[i], GFileIndex[i],
src_offset_table, stdout)) > 0) {
                                                gnum_of_matched += ret;
                                                gfiles_matched ++;
                                        }
                                        SetCurrentFileName = 0;
                                        if (GPRINTFILETIME)
SetCurrentFileTime = 0;
                                }
                        }
                        if (GLIMITOUTPUT > 0) {
                                if (GLIMITOUTPUT <=
gnum_of_matched) break;
                                LIMITOUTPUT = GLIMITOUTPUT -
gnum_of_matched;
                        }
                        if (GLIMITTOTALFILE > 0) {
                                if (GLIMITTOTALFILE <=
gfiles_matched) break;
                                LIMITTOTALFILE =
GLIMITTOTALFILE - gfiles_matched;
                        }
                        if ((ret < 0) && (errno == AGREP_ERROR))
break;
                        if (glimpse_clientdied) break;
                        fflush(stdout);
                }
```

```
                }
        } /* end of WHOLEFILESCOPE <= 0 */
        else {
                FILE    *tmpfp = NULL;        /* to store structured query-search
output */
                int     OLDFILENAMEONLY;/* don't use FILENAMEONLY
for agrepping the stuff: handle it in filtering */
                int     OLDLIMITOUTPUT; /* don't use LIMITs for search: only
for filtering=identify_region(): agrep NEVER changes these 3 */
                int     OLDLIMITPERFILE;
                int     OLDLIMITTOTALFILE;
                int     OLDPRINTRECORD;         /* don't use PRINTRECORD
for search: only after filter_output() recognizes boolean in wholefilescope */
                int     OLDCOUNT; /* don't use OLDCOUNT for search: only
after filter_output() recognizes boolean in wholefilescope */

                if (!UseFilters) {
                        for (i=0; i<GNumfiles; i++) {
                                /* if (RecordLevelIndex &&
(src_offset_table[GFileIndex[i]] == NULL)) continue; */
                                OLDFILENAMEONLY = FILENAMEONLY;
                                FILENAMEONLY = 0;
                                OLDLIMITOUTPUT = LIMITOUTPUT;
                                LIMITOUTPUT = 0;
                                OLDLIMITPERFILE = LIMITPERFILE;
                                LIMITPERFILE = 0;
                                OLDLIMITTOTALFILE = LIMITTOTALFILE;
                                LIMITTOTALFILE = 0;
                                OLDPRINTRECORD = PRINTRECORD;
                                PRINTRECORD = 1;
                                OLDCOUNT = COUNT;
                                COUNT = 0;
                                gprev_num_of_matched = gnum_of_matched;
                                if ((tmpfp = fopen(tempfile, "w")) == NULL) {
                                        fprintf(stderr, "%s: cannot open for writing:
%s, errno=%d\n", GProgname, tempfile, errno);
                                        RETURN(usage());
                                }
                                SetCurrentFileName = 1;
                                if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                else strcpy(CurrentFileName, GTextfiles[i]);
                                if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL) {
                                        if (GPRINTFILETIME) {
                                                SetCurrentFileTime = 1;
```

```
                                        CurrentFileTime =
get_file_time(timesfp, NULL, GTextfiles[i], GFileIndex[i]);
                                }
                                first_search = 1;
                                ret = fileagrep_search(AM, APattern, 1,
&GTextfiles[i], 0, tmpfp);
                        }
                        else {
                                if (my_stat(GTextfiles[i], &file_stat_buf) ==
-1) {
                                        if (GPRINTNONEXISTENTFILE)
printf("%s\n", CurrentFileName);
                                        fclose(tmpfp);
                                        continue;
                                }
                                if (GPRINTFILETIME) {
                                        SetCurrentFileTime = 1;
                                        CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                }
                                if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
                                        /* fprintf(stderr, "Warning: file
modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                        first_search = 1;
                                        ret = fileagrep_search(AM, APattern,
1, &GTextfiles[i], 0, tmpfp);
                                }
                                else ret = glimpse_search(AM, APattern,
GD_length, GD_pattern, GTextfiles[i], GTextfiles[i], GFileIndex[i], src_offset_table,
tmpfp);
                        }
                        SetCurrentFileName = 0;
                        if (GPRINTFILETIME) SetCurrentFileTime = 0;
                        fflush(tmpfp);
                        fclose(tmpfp);
                        tmpfp = NULL;
                        if ((ret < 0) && (errno == AGREP_ERROR))
break;
#if      DEBUG
                        printf("done search\n");
                        fflush(stdout);
#endif  /*DEBUG*/
                        FILENAMEONLY = OLDFILENAMEONLY;
```

```
                                        LIMITOUTPUT = OLDLIMITOUTPUT;
                                        LIMITPERFILE = OLDLIMITPERFILE;
                                        LIMITTOTALFILE = OLDLIMITTOTALFILE;
                                        PRINTRECORD = OLDPRINTRECORD;
                                        COUNT = OLDCOUNT;
                                if (!UseFilters) {
                                    ret = filter_output(GTextfiles[i], tempfile, GParse,
GD_pattern, GD_length, GOUTTAIL, nullfp, StructuredIndex);
                                } else {
                                    ret = filter_output(outname[0], tempfile, GParse,
GD_pattern, GD_length, GOUTTAIL, nullfp, StructuredIndex);
                                    unlink(outname[0]);
                                }
                                        gnum_of_matched += (ret > 0) ? ret : 0;
                                        gfiles_matched += (ret > 0) ? 1 : 0;
                                        if (GLIMITOUTPUT > 0) {
                                                if (GLIMITOUTPUT <=
gnum_of_matched) break;
                                                LIMITOUTPUT = GLIMITOUTPUT -
gnum_of_matched;
                                        }
                                        if (GLIMITTOTALFILE > 0) {
                                                if (GLIMITTOTALFILE <=
gfiles_matched) break;
                                                LIMITTOTALFILE =
GLIMITTOTALFILE - gfiles_matched;
                                        }
                                        if (glimpse_clientdied) break;
                                        fflush(stdout);
                                }
                        }
                        else {  /* we should try to apply the filter (we come here with -W -
z, say) */
                                sprintf(outname[0], "%s/.glimpse_apply.%d", TEMP_DIR,
getpid());
                                for (i=0; i<GNumfiles; i++) {
                                        /* if (RecordLevelIndex &&
(src_offset_table[GFileIndex[i]] == NULL)) continue; */
                                        OLDFILENAMEONLY = FILENAMEONLY;
                                        FILENAMEONLY = 0;
                                        OLDLIMITOUTPUT = LIMITOUTPUT;
                                        LIMITOUTPUT = 0;
                                        OLDLIMITPERFILE = LIMITPERFILE;
                                        LIMITPERFILE = 0;
                                        OLDLIMITTOTALFILE = LIMITTOTALFILE;
                                        LIMITTOTALFILE = 0;
```

```
                                        OLDPRINTRECORD = PRINTRECORD;
                                        PRINTRECORD = 1;
                                        OLDCOUNT = COUNT;
                                        COUNT = 0;
                                        gprev_num_of_matched = gnum_of_matched;
                                        if ((tmpfp = fopen(tempfile, "w")) == NULL) {
                                                fprintf(stderr, "%s: cannot open for writing:
%s, errno=%d\n", GProgname, tempfile, errno);
                                                RETURN(usage());
                                        }

                                        SetCurrentFileName = 1;
                                        if (GPRINTFILENUMBER)
sprintf(CurrentFileName, "%d", GFileIndex[i]);
                                        else strcpy(CurrentFileName, GTextfiles[i]);
                                        if (apply_filter(GTextfiles[i], outname[0]) == 1) {
                                                if (my_stat(GTextfiles[i], &file_stat_buf) ==
-1) {
                                                        if (GPRINTNONEXISTENTFILE)
printf("%s\n", CurrentFileName);
                                                        fclose(tmpfp);
                                                        continue;
                                                }
                                                if (GPRINTFILETIME) {
                                                        SetCurrentFileTime = 1;
                                                        CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                                }
                                                if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL || (file_stat_buf.st_mtime > index_stat_buf.st_mtime)) {
                                                        first_search = 1;
                                                        ret = fileagrep_search(AM, APattern,
1, outname, 0, tmpfp);
                                                }
                                                else {
                                                        if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
                                                                /* fprintf(stderr, "Warning:
file modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                                                first_search = 1;
                                                                ret = fileagrep_search(AM,
APattern, 1, outname, 0, tmpfp);
                                                        }
```

```
                                                else ret = glimpse_search(AM,
APattern, GD_length, GD_pattern, GTextfiles[i], outname[0], GFileIndex[i],
src_offset_table, tmpfp);
                                        }
                                }
                                else {
                                        if (!ByteLevelIndex || RecordLevelIndex ||
NOBYTELEVEL) {
                                                if (GPRINTFILETIME) {
                                                        SetCurrentFileTime = 1;
                                                        CurrentFileTime =
get_file_time(timesfp, NULL, GTextfiles[i], GFileIndex[i]);
                                                }
                                                first_search = 1;
                                                ret = fileagrep_search(AM, APattern,
1, &GTextfiles[i], 0, tmpfp);
                                        }
                                        else {
                                                if (my_stat(GTextfiles[i],
&file_stat_buf) == -1) {
                                                        if
(GPRINTNONEXISTENTFILE) printf("%s\n", CurrentFileName);
                                                        fclose(tmpfp);
                                                        continue;
                                                }
                                                if (GPRINTFILETIME) {
                                                        SetCurrentFileTime = 1;
                                                        CurrentFileTime =
get_file_time(timesfp, &file_stat_buf, GTextfiles[i], GFileIndex[i]);
                                                }
                                                if (file_stat_buf.st_mtime >
index_stat_buf.st_mtime) {
                                                        /* fprintf(stderr, "Warning:
file modified after indexing: must SEARCH %s\n", CurrentFileName); */

        free_list(&src_offset_table[GFileIndex[i]]);
                                                        first_search = 1;
                                                        ret = fileagrep_search(AM,
APattern, 1, &GTextfiles[i], 0, tmpfp);
                                                }
                                                else ret = glimpse_search(AM,
APattern, GD_length, GD_pattern, GTextfiles[i], GTextfiles[i], GFileIndex[i],
src_offset_table, tmpfp);
                                        }
                                }
                                SetCurrentFileName = 0;
```

```
                                        if (GPRINTFILETIME) SetCurrentFileTime = 0;
                                        fflush(tmpfp);
                                        fclose(tmpfp);
                                        tmpfp = NULL;
                                        if ((ret < 0) && (errno == AGREP_ERROR))
break;
#if     DEBUG
                                        printf("done search\n");
                                        fflush(stdout);
#endif  /*DEBUG*/
                                        FILENAMEONLY = OLDFILENAMEONLY;
                                        LIMITOUTPUT = OLDLIMITOUTPUT;
                                        LIMITPERFILE = OLDLIMITPERFILE;
                                        LIMITTOTALFILE = OLDLIMITTOTALFILE;
                                        PRINTRECORD = OLDPRINTRECORD;
                                        COUNT = OLDCOUNT;
                                        if (!UseFilters) {   /* Added to do structured queries
from Webglimpse */
                                                ret = filter_output(GTextfiles[i], tempfile,
GParse, GD_pattern, GD_length, GOUTTAIL, nullfp, StructuredIndex);
                                        } else {
                                                ret = filter_output(outname[0], tempfile,
GParse, GD_pattern, GD_length, GOUTTAIL, nullfp, StructuredIndex);
                                        }
                                        gnum_of_matched += (ret > 0) ? ret : 0;
                                        gfiles_matched += (ret > 0) ? 1 : 0;
                                        if (GLIMITOUTPUT > 0) {
                                                if (GLIMITOUTPUT <=
gnum_of_matched) break;
                                                LIMITOUTPUT = GLIMITOUTPUT -
gnum_of_matched;
                                        }
                                        if (GLIMITTOTALFILE > 0) {
                                                if (GLIMITTOTALFILE <=
gfiles_matched) break;
                                                LIMITTOTALFILE =
GLIMITTOTALFILE - gfiles_matched;
                                        }
                                        if (glimpse_clientdied) break;
                                        fflush(stdout);
                                }
                        }
                }
                if (errno == AGREP_ERROR) {
                        fprintf(stderr, "%s: error in options or arguments to `agrep'\n",
HARVEST_PREFIX);
```

```
                }

                RETURN(gnum_of_matched);
        }
        else { /* argc > 0: simply call agrep */
#if     DEBUG
                for (i=0; i<agrep_argc; i++)
                        printf("agrep_argv[%d] = %s\n", i, agrep_argv[i]);
#endif  /*DEBUG*/
                i = fileagrep(oldargc, oldargv, 0, stdout);
                RETURN(i);
        }
}
/* end of process_query() */

/*
 * Simple function to remove the non-existent files from the set of
 * files passed onto agrep for search. These are the files which got
 * DELETED after the index was built (but a fresh index was NOT built).
 * Redundant since agrep opens them anyway and stat is as bad as open.
 */
int
purge_filenames(filenames, num)
        CHAR **filenames;
        int     num;
{
        struct stat     buf;
        int             i, j;
        int             newnum = num;
        int             ret;

        for (i=0; i<newnum; i++) {
                if (-1 == (ret = my_stat(filenames[i], &buf))) {
#if     BG_DEBUG
                        fprintf(debug, "stat on %s = %d\n", filenames[i], ret);
#endif  /*BG_DEBUG*/
                        my_free(filenames[i], 0);
                        for (j=i; j<newnum-1; j++)
                                filenames[j] = filenames[j+1];
                        filenames[j] = NULL;
                        newnum --;
                        i--;    /* to counter the ++ on top */
                }
        }

#if     BG_DEBUG
```

```
        fprintf(debug, "Old numfiles=%d\tNew numfiles=%d\n", num, newnum);
        for (i=0; i<newnum; i++)
                fprintf(debug, "file %d = %s\n", i, filenames[i]);
#endif  /*BG_DEBUG*/
        return newnum;
}

CHAR filter_buf[BLOCKSIZE + MAXPAT*2];

/* returns #of bytes stripped off */
int getbyteoff(buf, pbyteoff)
        CHAR *buf;
        int     *pbyteoff;
{
        CHAR temp[32];
        int     i = 0;

        while (isdigit(*buf) && (i<32)) temp[i++] = *buf++;
        if ((*buf != '=') || (*(buf + 1) != ' ')) return -1;
        temp[i] = '\0';
        *pbyteoff = atoi(temp);
        return i+2;
}

/*
 * Filter the output in infile:
 *
 * -- get the matched line/record-s using GD_pattern, GD_length and GOUTAIL
 * -- call identify regions using matched line/record's byte offset
 * -- collect patterns corr. to that attribute into a new pattern (in split_pat itself)
 * -- see if one of them matches that line/record using memagrep
 * -- if so, output that line/record onto stdout
 */
 /* May have to change name to reflect the fact that it does booleans too */
int
filter_output(infile, outfile, GParse, GD_pattern, GD_length, GOUTTAIL, nullfp,
num_attr)
        char    *infile;
        char    *outfile;
        ParseTree *GParse;
        CHAR GD_pattern[];
        int     GD_length;
        int     GOUTTAIL;
        FILE    *nullfp;
        int     num_attr;
{
```

```
        FILE    *outfp;
        FILE    *displayfp = NULL;
        FILE    *storefp = NULL;
        int     num_read, total_read = 0;
        int     residue = 0;
        int     byteoff;
        int     attribute;
        int     i, ii;     /* i is forloop index, ii is booleaneval index */
        CHAR *final_end;
        CHAR *current_end;
        CHAR *current_begin;
        CHAR *previous_begin;
        int     skiplen;
        char    s[MAX_LINE_LEN];
        CHAR c1, c2;
        int     printed, numprinted = 0;        /* returns number of printed records if
successful in matching the pattern in the object infile */
        char    *attrname;
        int     success = 0;    /* do we print the stored output or not */
        int     count = 0;
        int     OLDLIMITOUTPUT = 0, OLDLIMITPERFILE = 0,
OLDLIMITTOTALFILE = 0;

#if     BG_DEBUG
        printf("INFILE=%s\n", infile);
        printf("OUTFILE\n");
        sprintf(s, "exec cat %s\n", outfile);
        system(s);
        printf("OUTFILEDONE\n");
#endif  /*BG_DEBUG*/
        if ((outfp = fopen(outfile, "r")) == NULL) return 0;
        if (StructuredIndex && (-1 == region_create(infile))) {
                fclose(outfp);
                return 0;
        }
        if (ComplexBoolean || ((long)GParse & AND_EXP)) {
                sprintf(s, "%s/.glimpse_storeoutput.%d", TEMP_DIR, getpid());
                if ((displayfp = storefp = fopen(s, "w")) == NULL) {
                        if (StructuredIndex) region_destroy();
                        fclose(outfp);
                        return 0;
                }
        }
        else {
                displayfp = stdout;
                /* cannot come to filter_output in this case unless -a! */
```

```
	}
	memset(matched_terminals, '\0', num_terminals);

	while ( ( (num_read = fread(filter_buf + residue, 1, BLOCKSIZE - residue,
outfp)) > 0) || (residue > 0)) {
		total_read += num_read;
		if (num_read <= 0) {
			final_end = filter_buf + residue;
			num_read = residue;
			residue = 0;
		}
		else {
			num_read += residue;
			final_end = (CHAR *)backward_delimiter(filter_buf + num_read,
filter_buf, GD_pattern, GD_length, GOUTTAIL);
			residue = filter_buf + num_read - final_end;
		}
#if	DEBUG
		fprintf(stderr, "filter_buf=%x final_end=%x residue=%x
last_chars=%c%c%c num_read=%x\n",
			filter_buf, final_end, residue, *(final_end-2), *(final_end-1),
*(final_end), num_read);
#endif  /*DEBUG*/

		current_begin = previous_begin = filter_buf;
#if	1
		current_end = (CHAR *)forward_delimiter(filter_buf, filter_buf +
num_read, GD_pattern, GD_length, GOUTTAIL);   /* skip over prefixes like filename */
		if (!GOUTTAIL) current_end = (CHAR
*)forward_delimiter((long)current_end + GD_length, final_end, GD_pattern, GD_length,
GOUTTAIL);
#else	/*1*/
		current_end = (CHAR *)forward_delimiter(filter_buf+1, final_end,
GD_pattern, GD_length, GOUTTAIL);

#endif  /*1*/

#if	DEBUG
		fprintf(stderr, "current_begin=%x current_end=%x\n", current_begin,
current_end);
#endif  /*DEBUG*/

		while (current_end <= final_end) {
			previous_begin = current_begin;
			/* look for %d= */
			byteoff = -1;
```

```
                        while (current_begin < current_end) {
                                if (isdigit(*current_begin)) {
                                        skiplen = getbyteoff(current_begin, &byteoff);
#if     BG_DEBUG
                                        fprintf(debug, "byteoff=%d skiplen=%d\n", byteoff,
skiplen);
#endif  /*BG_DEBUG*/
                                        if ((skiplen < 0) || (byteoff < 0)) {
                                                current_begin ++;
                                                continue;
                                        }
                                        else break;
                                }
                                else current_begin ++;
                        }
#if     DEBUG
                        printf("current_begin=%x current_end=%x final_end=%x
residue=%x num_read=%x\n", current_begin, current_end, final_end, residue,
num_read);
#endif  /*DEBUG*/

#if     DEBUG
                        printf("byteoff=%d skiplen=%d\n", byteoff, skiplen);
#endif  /*DEBUG*/
                        if ((skiplen < 0) || (byteoff < 0)) {       /* output the whole line as it
is: there is nothing to strip (e.g., -l) */
#if     0
                                /* This is an error: -l is now handled completely inside
filter_output --> agrep won't processes it when -W */
                                if (!SILENT) fwrite(previous_begin, 1, current_end-
previous_begin, displayfp);
                                numprinted ++;
#endif
                        }
                        else if ( (num_attr <= 0) || (((attribute = region_identify(byteoff, 0))
< num_attr) && (attribute >= 0)) ) {
                                /* prefix is from previous_begin to current_begin. Skip
skiplen from current_begin. Rest until current_end is valid output */
                                if (num_attr <= 0) attribute = 0;
#if     BG_DEBUG
                                fprintf(debug, "region@%d=%d\n", byteoff, attribute);
#endif  /*BG_DEBUG*/
                                c1 = *(current_begin + skiplen - 1);
                                c2 = *(current_end + 1);
                                printed = 0;
```

```
                                for (i=0; i<num_terminals; i++) {
#if       0
                                        printf("--> already_matched[%d] = %d, going to
look at '%s'\n", i, matched_terminals[i], terminals[i].data.leaf.value);
#endif
                                        if (matched_terminals[i] && (GFILENAMEONLY
|| FILEOUT || printed || ((LIMITOUTPUT > 0) && (numprinted >= LIMITOUTPUT)) ||
((LIMITPERFILE > 0) && (numprinted >= LIMITPERFILE)))) continue;
                                        if ((terminals[i].data.leaf.attribute == 0)  ||
((int)(terminals[i].data.leaf.attribute) == attribute)) {
                                                *(current_begin + skiplen - 1) = '\n';
                                                *(current_end + 1) = '\n';
                                                OLDLIMITOUTPUT = LIMITOUTPUT;
                                                LIMITOUTPUT = 0;
                                                OLDLIMITPERFILE = LIMITPERFILE;
                                                LIMITPERFILE = 0;
                                                OLDLIMITTOTALFILE =
LIMITTOTALFILE;
                                                LIMITTOTALFILE = 0;
                                                if (memagrep_search(
        strlen(terminals[i].data.leaf.value), terminals[i].data.leaf.value,
                                                                current_end -
current_begin - skiplen + 1, current_begin + skiplen - 1,
                                                                0, nullfp) > 0) {
                                                        LIMITOUTPUT =
OLDLIMITOUTPUT;
                                                        LIMITPERFILE =
OLDLIMITPERFILE;
                                                        LIMITTOTALFILE =
OLDLIMITTOTALFILE;

#if       0
                                                        *(current_end + 1) = '\0';
                                                        printf("--> search succeeded for %s
in %s\n", terminals[i].data.leaf.value, previous_begin);
#endif  /*0*/
                                                        *(current_begin + skiplen - 1) = c1;
                                                        *(current_end + 1) = c2;
                                                        matched_terminals[i] = 1;       /* must
reevaluate/set since don't know if it should be printed */

                                                        if (!(((LIMITOUTPUT > 0) &&
(numprinted >= LIMITOUTPUT)) ||
                                                             ((LIMITPERFILE > 0) &&
(numprinted >= LIMITPERFILE))) && !printed) {  /* see if it was useful later */
```

```
                                                        if (!COUNT && !FILEOUT
&& !SILENT) {
                                                        fwrite(previous_begin, 1,
current_begin - previous_begin, displayfp);
                                                        if (PRINTATTR)
fprintf(displayfp, "%s# ",
                                                                (attrname =
attr_id_to_name(attribute)) == NULL ? "(null)" : attrname);
                                                        if (GBYTECOUNT)
fprintf(displayfp, "%d= ", byteoff);
                                                        if (PRINTRECORD) {
                                                        fwrite(current_begin +
skiplen, 1, current_end - current_begin - skiplen, displayfp);
                                                        }
                                                        else {
                                                                if (*(current_begin +
skiplen) == '@') {
                                                                        int       iii = 0;
                                                                        while
(current_begin[skiplen + iii] != '}')

        fputc(current_begin[skiplen + iii++], displayfp);
                                                                        fputc('}',
displayfp);
                                                                }
                                                                fputc('\n', displayfp);
                                                        }
                                                        }
                                                        printed = 1;
                                                        numprinted ++;
                                                }
                                        }
                                        else {
                                                LIMITOUTPUT =
OLDLIMITOUTPUT;
                                                LIMITPERFILE =
OLDLIMITPERFILE;
                                                LIMITTOTALFILE =
OLDLIMITTOTALFILE;
#if       0
                                                *(current_end + 1) = '\0';
                                                printf("--> search failed for %s in
%s\n", terminals[i].data.leaf.value, previous_begin);
#endif  /*0*/
                                                *(current_begin + skiplen - 1) = c1;
                                                *(current_end + 1) = c2;
```

```
                                        }
                                }
                        }

                        if (!success) {
                                if (ComplexBoolean) {
                                        success = eval_tree(GParse,
matched_terminals);
                                }
                                else {
                                        if ((long)GParse & AND_EXP) {
                                                success = 0;
                                                for (ii=0; ii<num_terminals; ii++) {
                                                        if (!matched_terminals[ii])
break;
                                                }
                                                if (ii >= num_terminals) success = 1;
                                        }
                                        else {
                                                success = 0;
                                                /* cannot come to filter_output in
this case unless -a! */
                                        }
                                }
                        }

                        /* optimize options that do not need all the matched lines */
                        if (success) {
                                if (GFILENAMEONLY) {
                                        if (GPRINTFILETIME) {      /* from bug
fix message by  Dr Jaime Prilusky lsprilus@weizmann.weizmann.ac.il
jaimep@terminator.pdb.bnl.gov */
                                                if (!SILENT) fprintf(stdout,
"%s%s\n", CurrentFileName, aprint_file_time(CurrentFileTime));
                                        }
                                        else {
                                                if (!SILENT) fprintf(stdout, "%s\n",
CurrentFileName);
                                        }
                                        if (storefp != NULL) fclose(storefp); /* don't
bother to flush! */
                                        storefp = NULL;
                                        goto unlink_and_quit;
                                }
                                else if (FILEOUT) {
                                        /* file_out(infile); */
```

```
                                        if (storefp != NULL) fclose(storefp); /* don't
bother to flush! */
                                        storefp = NULL;
                                        goto unlink_and_quit;
                                }
                        }
                }
                if (success && (((LIMITOUTPUT > 0) && (numprinted >=
LIMITOUTPUT)) || ((LIMITPERFILE > 0) && (numprinted >= LIMITPERFILE))))
goto double_break;
                if (glimpse_clientdied) goto double_break;
                if (current_end >= final_end) break;
                current_begin = current_end;
                if (!GOUTTAIL) current_end = (CHAR
*)forward_delimiter((long)current_end + GD_length, final_end, GD_pattern, GD_length,
GOUTTAIL);
                else current_end = (CHAR *)forward_delimiter(current_end,
final_end, GD_pattern, GD_length, GOUTTAIL);
#if     DEBUG
        fprintf(stderr, "current_begin=%x current_end=%x\n", current_begin,
current_end);
#endif  /*DEBUG*/
        }
        if (residue > 0) {
                memcpy(filter_buf, final_end, residue);
                memcpy(filter_buf+residue, GD_pattern, GD_length);
        }
}

double_break:
        /* Come here on normal exit or when the current agrep-output is no longer of any
use */
        if (!success && (total_read > 0)) {
                if (ComplexBoolean) {
                        success = eval_tree(GParse, matched_terminals);
                }
                else {
                        if ((long)GParse & AND_EXP) {
                                success = 0;
                                for (ii=0; ii<num_terminals; ii++) {
                                        if (!matched_terminals[ii]) break;
                                }
                                if (ii >= num_terminals) success = 1;
                        }
                        else {
                                success = 0;
```

```
                        /* cannot come to filter_output in this case unless -a! */
                }
        }
}

        /* Print the temporary output onto stdout if search was successful; unlink the
temprorary file */
        if (success) {
                if (GFILENAMEONLY) {    /* all other output options are useless since
they all deal with the MATCHED line */
                        if (GPRINTFILETIME) {     /* from bug fix message by  Dr
Jaime Prilusky lsprilus@weizmann.weizmann.ac.il jaimep@terminator.pdb.bnl.gov */
                                if (!SILENT) fprintf(stdout, "%s%s\n", CurrentFileName,
aprint_file_time(CurrentFileTime));
                        }
                        else {
                                if (!SILENT) fprintf(stdout, "%s\n", CurrentFileName);
                        }
                        if (!SILENT) fprintf(stdout, "%s\n", CurrentFileName);
                        if (storefp != NULL) fclose(storefp); /* don't bother to flush! */
                        storefp = NULL;
                }
                else if (COUNT && !FILEOUT) {
                        if (!SILENT) {
                                if(!NOFILENAME) fprintf(stdout, "%s: %d\n",
CurrentFileName, numprinted);
                                else fprintf(stdout, "%d\n", numprinted);
                        }
                        if (storefp != NULL) fclose(storefp); /* don't bother to flush! */
                        storefp = NULL;
                }
                else if (FILEOUT) {
                        /* file_out(infile); */
                        if (storefp != NULL) fclose(storefp); /* don't bother to flush! */
                        storefp = NULL;
                }
                else if (storefp != NULL) {
                        fflush(storefp);
                        fclose(storefp);
#if     DEBUG
                        printf("STOREOUTPUT\n");
                        sprintf(s, "exec cat %s/.glimpse_storeoutput.%d\n", TEMP_DIR,
getpid());
                        system(s);
#endif  /*DEBUG*/
                        sprintf(s, "%s/.glimpse_storeoutput.%d", TEMP_DIR, getpid());
```

```
                        if ((storefp = fopen(s, "r")) != NULL) {
                                if (!SILENT) while (fgets(s, MAX_LINE_LEN, storefp) !=
NULL) fputs(s, stdout);
                                fclose(storefp);
                        }
                        storefp = NULL;
                }
        }
        else {
                if (storefp != NULL) fclose(storefp); /* else don't bother to flush */
        }

unlink_and_quit:
        sprintf(s, "%s/.glimpse_storeoutput.%d", TEMP_DIR, getpid());
        unlink(s);

        if (StructuredIndex) region_destroy();
        fclose(outfp);

        if (GFILENAMEONLY) {
                if (numprinted > 0) return 1;
                else return 0;
        }
        else if (ComplexBoolean || ((long)GParse & AND_EXP)) {
                if (success) return numprinted;
                else return 0;
        } else {        /* must be -a */
                return numprinted;
        }
}

usage()
{
        fprintf(stderr, "\nThis is glimpse version %s, %s.\n\n", GLIMPSE_VERSION,
GLIMPSE_DATE);
        fprintf(stderr, "usage:  %s [-#abceghijklnprstwxyBCDEGIMNPQSVWZ] [-d
DEL] [-f FILE] [-F PAT] [-H DIR] [-J HOST] [-K PORT] [-L X[:Y:Z]] [-R X] [-T DIR]
[-Y D] pattern [files]", GProgname);
        fprintf(stderr, "\n");
        fprintf(stderr, "List of options (see %s for more details):\n", GLIMPSE_URL);
        fprintf(stderr, "\n");

        fprintf(stderr, "-#: find matches with at most # errors\n");
        fprintf(stderr, "-a: print attribute names (useful only for Harvest SOIF format)\n");
        fprintf(stderr, "-b: print the byte offset of the record from the beginning of the
file\n");
```

```
        fprintf(stderr, "-B: best match mode: find the closest matches to the pattern\n");
        fprintf(stderr, "-c: output the number of matched records\n");
        fprintf(stderr, "-C: send queries to glimpseserver\n");
        fprintf(stderr, "-d DEL: define record delimiter DEL\n");
        fprintf(stderr, "-D x: adjust the cost of deletions to x\n");
        fprintf(stderr, "-e: for patterns starting with -\n");
        fprintf(stderr, "-E: print matching lines as they appear in the index (useful in -
EQNg)\n");
        fprintf(stderr, "-f FILE: restrict the search to files whose names appear in
FILE\n");
        fprintf(stderr, "-F PAT: restrict the search to files matching PAT\n");
        fprintf(stderr, "-g: print the file number (in the index)\n");
        fprintf(stderr, "-G: output the (whole) files that contain a match\n");
        fprintf(stderr, "-h: do not output file names before matched record\n");
        fprintf(stderr, "-H DIR: the glimpse index is located in directory DIR\n");
        fprintf(stderr, "-i: case-insensitive search, e.g., 'a' = 'A'\n");
        fprintf(stderr, "-I x: adjust the cost of insertions to x\n");
        fprintf(stderr, "-j: output file modification dates (if -t was used for the
indexing)\n");
        fprintf(stderr, "-J HOST: send queries to glimpseserver at HOST\n");
        fprintf(stderr, "-k: use the pattern as is (no meta characters)\n");
        fprintf(stderr, "-K PORT: send queries to glimpseserver at TCP port number
PORT\n");
        fprintf(stderr, "-l: output only the names of files that contain a match\n");
        fprintf(stderr, "-L X[:Y:Z]: limit the output to X records [Y files, Z matches per
file]\n");
        fprintf(stderr, "-n: output record prefixed by record number\n");
        fprintf(stderr, "-N: search only the index (may not be precise for some queries)
\n");
        fprintf(stderr, "-o: delimiter is output at the beginning of the matched record\n");
        fprintf(stderr, "-O: file names are printed only once per file\n");
        fprintf(stderr, "-p FILE:off:len:endian: restrict the search to the files whose
names\n\tare specified as a bit-field OR sparse-set in FILE\n");
        fprintf(stderr, "-P: print the pattern that matched before the matched record\n");
        fprintf(stderr, "-q: print the offsets of the beginning and end of each matched
record\n");
        fprintf(stderr, "-Q: (with -N) print offsets of matches from (only the large)
index\n");
        fprintf(stderr, "-r: (used only for agrep) - recursive search\n");
        fprintf(stderr, "-R X: set the maximum size of a record to X\n");
        fprintf(stderr, "-s: display nothing except error messages\n");
        fprintf(stderr, "-S x: adjust the cost of substitutions to x\n");
        fprintf(stderr, "-t: use in combination with -d DEL so that the delimiter DEL
appears\n\tat the end of each output record instead of the beginning\n");
        fprintf(stderr, "-T DIR: temporary files are put in directory DIR (instead of
/tmp)\n");
```

```
        fprintf(stderr, "-u: do not output matched records (useful in -qbug)\n");
        fprintf(stderr, "-U: interpret .glimpse_filenames when -U / -X option is used in
glimpseindex\n");
        fprintf(stderr, "-v: (works ONLY for agrep) - output all records that do not
contain a match\n");
        fprintf(stderr, "-V,--version: print the current version of glimpse\n");
        fprintf(stderr, "-w: pattern has to match as a word, e.g., 'win' will not match
'wind'\n");
        fprintf(stderr, "-W: the scope of Booleans is the whole file (except for structured
queries)\n");
        fprintf(stderr, "-x: the pattern must match the whole line\n");
        fprintf(stderr, "-X: if an indexed file that matches 'pattern' doesn't exist, print its
name\n");
        fprintf(stderr, "-y: no prompt\n");
        fprintf(stderr, "-Y D: output only matches in files that were updated in the last D
days\n");
        fprintf(stderr, "-z: customizable filtering using the .glimpse_filters file\n");
        fprintf(stderr, "-Z: no op\n");
        fprintf(stderr, "--help: this message\n");
        fprintf(stderr, "\n");

        fprintf(stderr, "For questions about glimpse, please contact: `%s'\n",
GLIMPSE_EMAIL);

        return -1;      /* useful if we make glimpse into a library */

/*
 * Undocumented Option Combinations for SFS (like RPC calls)
 *      print file number of match instead of file name: -g
 *      print enclosing offsets of matched record: -q
 *      NOT print matched record: -u
 * E.G. USAGE: -qbug (b prints offset of pattern: can also use -lg or -Ng)
 *      look only at index: -E
 *      look at matched offsets in files as seen in index (w/o searching): -QN
 * E.G. USAGE: -EQNgy
 *      read the -EQNg or just -QNg output from stdin and perform actual search w/o
 *      searching the index (take hints from user): -U
 * NOTE: can't use U unless QNg are all used together (e.g., BEGIN/END won't be
printed)
 */
}

usageS()
{
        fprintf(stderr, "\nThis is glimpse server version %s, %s.\n\n",
GLIMPSE_VERSION, GLIMPSE_DATE);
```

```
        fprintf(stderr, "usage:  %s [-H DIR] [-J HOST] [-K PORT]", GProgname);
        fprintf(stderr, "\n");
        fprintf(stderr, "-H DIR: the glimpse index is located in directory DIR\n");
        fprintf(stderr, "-J HOST: the host name (string) clients must use / server runs
on\n");
        fprintf(stderr, "-K PORT: the port (short integer) clients must use / server runs
on\n");
        fprintf(stderr, "\n");
        fprintf(stderr, "For questions about glimpse, please contact `%s'\n",
GLIMPSE_EMAIL);

        return -1;      /* useful if we make glimpse into a library */
}

#if     CLIENTSERVER
/*
 *  do_select() - based on select_loop() from the Harvest Broker.
 *  -- Courtesy: Darren Hardy, hardy@cs.colorado.edu
 */
int do_select(sock, sec)
int sock;               /* the socket to wait for */
int sec;        /* the number of seconds to wait */
{
        struct timeval to;
        fd_set qready;
        int err;

        if (sock < 0 || sec < 0)
                return 0;

        FD_ZERO(&qready);
        FD_SET(sock, &qready);
        to.tv_sec = sec;
        to.tv_usec = 0;
        if ((err = select(sock + 1, &qready, NULL, NULL, &to)) < 0) {
                if (errno == EINTR)
                        return 0;
                perror("select");
                return -1;
        }
        if (err == 0)
                return 0;
        /* If there's someone waiting to get it, let them through */
        return (FD_ISSET(sock, &qready) ? 1 : 0);
}
#endif  /* CLIENTSERVER */
```

APPENDIX E: GLIMPSE – GLIMPSE.C

Throughout the indexing and searching processes, there were a number of prompts that were eliminated. Glimpse also performed a number of calculation before indexing a collection, including counting the number of files in the collection and calculating the size of the collection. These operations were bypassed to create an operational environment similar to the other retrieval solution mentioned above.

```
/* Copyright (c) 1994 Sun Wu, Udi Manber, Burra Gopal.  All Rights Reserved. */
/* ./glimpse/index/glimpse.c */
#include "glimpse.h"
#include <stdlib.h>
#include <sys/time.h>
#if       ISO_CHAR_SET
#include <locale.h>    /* support for 8bit character set:ew@senate.be */
#endif
#include <errno.h>

extern char **environ;
extern int errno;
extern FILE *TIMEFILE;      /* file descriptor for sorting .glimpse_filenames by time */
#if       BG_DEBUG
extern FILE  *LOGFILE;      /* file descriptor for LOG output */
#endif  /*BG_DEBUG*/
extern FILE  *STATFILE;     /* file descriptor for statistical data about indexed files */
extern FILE  *MESSAGEFILE;        /* file descriptor for important messages meant for
the user */
extern char  INDEX_DIR[MAX_LINE_LEN];
extern struct stat istbuf;

#ifdef BUILDCAST
/* TEMP_DIR is normally defined in ../main.c; if we're building
 * buildcast, that's not linked in, so we need to define one here. */
/* char * TEMP_DIR = NULL; */
static char * TEMP_DIR = "/tmp";
#else
extern char      *TEMP_DIR; /* directory to store glimpse temporary files, usually /tmp
unless -T is specified */
#endif /* BUILDCAST */
extern int indexable_char[256];
extern int GenerateHash;
extern int KeepFilenames;
```

```
extern int OneFilePerBlock;
extern int IndexNumber;
extern int CountWords;
extern int StructuredIndex;
extern int attr_num;
extern int MAXWORDSPERFILE;
extern int NUMERICWORDPERCENT;
extern int AddToIndex;
extern int DeleteFromIndex;
extern int PurgeIndex;
extern int FastIndex;
extern int BuildDictionary;
extern int BuildDictionaryExisting;
extern int CompressAfterBuild;
extern int IncludeHigherPriority;
extern int FilenamesOnStdin;
extern int ExtractInfo;
extern int InfoAfterFilename;
extern int FirstWordOfInfoIsKey;
extern int UseFilters;
extern int ByteLevelIndex;
extern int RecordLevelIndex;
extern int StoreByteOffset;
extern char rdelim[MAX_LINE_LEN];
extern char old_rdelim[MAX_LINE_LEN];
extern int rdelim_len;
/* extern int IndexUnderscore; */
extern int IndexableFile;
extern int MAX_PER_MB, MAX_INDEX_PERCENT;
extern int I_THRESHOLD;
extern int BigHashTable;
extern int BigFilenameHashTable;
extern int IndexEverything;
extern int BuildTurbo;
extern int SortByTime;

extern int AddedMaxWordsMessage;
extern int AddedMixedWordsMessage;

extern int file_num;
extern int old_file_num;
extern int new_file_num;
extern int file_id;
extern int part_num;
extern char **name_list[MAXNUM_INDIRECT];
extern int p_table[MAX_PARTITION];
```

```
extern int  *size_list[MAXNUM_INDIRECT];
extern int p_size_list[];
extern unsigned int *disable_list;
extern int memory_usage;
extern int mask_int[];
extern int REAL_PARTITION, REAL_INDEX_BUF, MAX_ALL_INDEX,
FILEMASK_SIZE;
extern struct indices    *deletedlist;
extern char sync_path[MAX_LINE_LEN];
extern int ATLEASTONEFILE;

extern set_usemalloc();          /* compress/misc.c */

char IProgname[MAX_LINE_LEN];
int ModifyFilenamesIndex = 0;

/*
 * Has newnum crossed the boundary of an encoding? This is so rare that we
 * needn't optimize it by changing the format of the old index and reusing it.
 */
cross_boundary(oldnum, newnum)
        int       oldnum, newnum;
{
        int       ret;

        if (oldnum <= 0) return 0;
        ret =  ( ((oldnum <= MaxNum8bPartition) && (newnum > MaxNum8bPartition))
||
                  ((oldnum <= MaxNum12bPartition) && (newnum >
MaxNum12bPartition)) ||
                  ((oldnum <= MaxNum16bPartition) && (newnum >
MaxNum16bPartition)) );
        if (ret) fprintf(MESSAGEFILE, "Must change index format. Commencing fresh
indexing...\n");
        return ret;
}

determine_sync()
{
        char     S[1024], s1[256], s2[256];
        FILE    *fp;
        int       i, ret;

        strcpy(sync_path, "sync");
        sprintf(S, "exec whereis sync > %s/zz.%d", TEMP_DIR,getpid());
        /* Change it to use which: not urgent. */
```

```
        system(S);
        sprintf(S, "%s/zz.%d", TEMP_DIR,getpid());
        if ((fp = fopen(S, "r")) == NULL) {
                /* printf("11111\n"); */
                return 0;
        }
        if ((ret = fread(S, 1, sizeof(S)-1, fp)) <= 0) {
                sprintf(S, "%s/zz.%d", TEMP_DIR,getpid());
                unlink(S);
                fclose(fp);
                /* printf("22222\n"); */
                return 0;
        }
      S [ret] = 0; /* terminate string */
        sprintf(s1, "%s/zz.%d", TEMP_DIR,getpid());
        unlink(s1);
        fclose(fp);
        /* printf("read: %s\n", S); */

        sscanf(S, "%s%s", s1, s2);
        /* printf("s1=%s s2=%s\n", s1, s2); */
        if (strncmp(s1, "sync", 4)) {
                /* printf("33333\n"); */
                return 0;
        }
        if (!strcmp(s2, "") || !strcmp(s2, " ")) {
                /* printf("44444\n"); */
                return 0;
        }
        if (strstr(s2, "sync") == NULL) {
                /* printf("55555\n"); */
                return 0;
        }
        strcpy(sync_path, s2);
        /* printf("Using sync in: %s\n", sync_path); */
        return 1;
}

main(argc, argv)
int argc;
char **argv;
{
   int pid = getpid();
   int   i, m = 0;
   char *indexdir, es1[MAX_LINE_LEN], es2[MAX_LINE_LEN];
   char s[MAX_LINE_LEN], s1[MAX_LINE_LEN];
```

```
    char working_dir[MAX_LINE_LEN];
    FILE *tmpfp;
    char hash_file[MAX_LINE_LEN], string_file[MAX_LINE_LEN],
freq_file[MAX_LINE_LEN];
    char tmpbuf[1024];
    struct stat stbuf;
    char name[MAX_LINE_LEN];
    char outname[MAX_LINE_LEN];
    int specialwords, threshold;
    int backup;
    struct indices *get_removed_indices();
    struct timeval tv;

#if       ISO_CHAR_SET
    setlocale(LC_ALL,""); /* support for 8bit character set: ew@senate.be,
Henrik.Martin@eua.ericsson.se */
#endif
    BuildDictionary = ON;
    set_usemalloc();
    srand(pid);
    umask(077);
    determine_sync();

    INDEX_DIR[0] = '\0';
    specialwords = threshold = -1;       /* so that compute_dictionary can use defaults not
visible here */
    strncpy(IProgname, argv[0], MAX_LINE_LEN);
    memset(size_list, '\0', sizeof(int *) * MAXNUM_INDIRECT);    /* free it once
partition successfully calculates p_size_list */
    memset(name_list, '\0', sizeof(char **) * MAXNUM_INDIRECT);
    memset(p_size_list, '\0', sizeof(int) * MAX_PARTITION);
    build_filename_hashtable((char *)NULL, 0);

    /*
     * Process options.
     */

    BuildTurbo = ON;  /* always ON: user can remove .glimpse_turbo if not needed */

    /*
     * Find the index directory since it is used in all options.
     */

    if (INDEX_DIR[0] == '\0') {
          if ((indexdir = getenv("HOME")) == NULL) {
             getcwd(INDEX_DIR, MAX_LINE_LEN-1);
```

```
            fprintf(stderr, "Using working-directory '%s' to store index\n\n", INDEX_DIR);
        }
        else strncpy(INDEX_DIR, indexdir, MAX_LINE_LEN);
    }
    getcwd(working_dir, MAX_LINE_LEN - 1);
    if (-1 == chdir(INDEX_DIR)) {
        fprintf(stderr, "Cannot change directory to %s\n", INDEX_DIR);
        return usage(0);
    }
    getcwd(INDEX_DIR, MAX_LINE_LEN - 1);      /* must be absolute path name */
    chdir(working_dir);/* get back to where you were */

    if (ByteLevelIndex) {
        CountWords = OFF;
        OneFilePerBlock = ON;
    }

    read_filters(INDEX_DIR, UseFilters);

    freq_file[0] = hash_file[0] = string_file[0] = '\0';
    strcpy(freq_file, INDEX_DIR);
    strcat(freq_file, "/");
    strcat(freq_file, DEF_FREQ_FILE);
    strcpy(hash_file, INDEX_DIR);
    strcat(hash_file, "/");
    strcat(hash_file, DEF_HASH_FILE);
    strcpy(string_file, INDEX_DIR);
    strcat(string_file, "/");
    strcat(string_file, DEF_STRING_FILE);
    initialize_tuncompress(string_file, freq_file, 0);

    sprintf(s, "%s/%s", INDEX_DIR, DEF_TIME_FILE);
    if((TIMEFILE = fopen(s, "w")) == 0) {
        fprintf(stderr, "can't open %s for writing\n", s);
        exit(2);
    }

#if     BG_DEBUG
    sprintf(s, "%s/%s", INDEX_DIR, DEF_LOG_FILE);
    if((LOGFILE = fopen(s, "w")) == 0) {
        fprintf(stderr, "can't open %s for writing\n", s);
        LOGFILE = stderr;
    }
#endif  /*BG_DEBUG*/
    sprintf(s, "%s/%s", INDEX_DIR, DEF_MESSAGE_FILE);
    if((MESSAGEFILE = fopen(s, "w")) == 0) {
```

```
        fprintf(stderr, "can't open %s for writing\n", s);
        MESSAGEFILE = stderr;
    }

    sprintf(s, "%s/%s", INDEX_DIR, DEF_STAT_FILE);
    if((STATFILE = fopen(s, "a")) == 0) {
        fprintf(stderr, "can't open %s for appending\n", s);
        STATFILE = stderr;
    }
    gettimeofday(&tv, NULL);
#if     BUILDCAST
    fprintf(STATFILE, "\nThis is buildcast version %s, %s. %s", GLIMPSE_VERSION,
GLIMPSE_DATE, ctime(&tv.tv_sec));
#endif

#if     BG_DEBUG
    fprintf(LOGFILE, "Index Directory = %s\n\n", INDEX_DIR);
#endif  /*BG_DEBUG*/
    if (MAXWORDSPERFILE != 0) fprintf(MESSAGEFILE, "Index: maximum number
of indexed words per file = %d\n", MAXWORDSPERFILE);
    else fprintf(MESSAGEFILE, "Index: maximum number of indexed words per file =
infinity\n");
    fprintf(MESSAGEFILE, "Index: maximum percentage of numeric words per file =
%d\n", NUMERICWORDPERCENT);
    set_indexable_char(indexable_char);
        /*printf("Before IF...");*/
#if     BUILDCAST
    CountWords = ON;
    AddToIndex = OFF;
    FastIndex = OFF;

    /* Save old search-dictionaries */

    sprintf(s, "%s/.glimpse_index", INDEX_DIR);
    if (!access(s, R_OK)) {
        sprintf(s, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        if (-1 == mkdir(s, 0700)) {
            fprintf(stderr, "cannot create temporary directory %s\n", s);
            return -1;
        }
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, INDEX_FILE);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
```

```
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), INDEX_FILE, escapesinglequote(INDEX_DIR,
es2), pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, P_TABLE);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), P_TABLE, escapesinglequote(INDEX_DIR, es2),
pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, NAME_LIST);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), NAME_LIST, escapesinglequote(INDEX_DIR,
es2), pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, NAME_LIST_INDEX);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), NAME_LIST_INDEX,
escapesinglequote(INDEX_DIR, es1), pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, NAME_HASH);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), NAME_HASH, escapesinglequote(INDEX_DIR,
es2), pid);
        system(s);
#endif
#if     SFS_COMPAT
```

```
        sprintf(s, "%s/%s", INDEX_DIR, NAME_HASH_INDEX);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), NAME_HASH_INDEX,
escapesinglequote(INDEX_DIR, es2), pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, MINI_FILE);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), MINI_FILE, escapesinglequote(INDEX_DIR,
es2), pid);
        system(s);
#endif
#if     SFS_COMPAT
        sprintf(s, "%s/%s", INDEX_DIR, DEF_STAT_FILE);
        sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rename(s, s1);
#else
        sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), DEF_STAT_FILE,
escapesinglequote(INDEX_DIR, es2), pid);
        system(s);
#endif
        /* Don't save messages, log, debug, etc. */
        sprintf(s, "%s/.glimpse_attributes", INDEX_DIR);
        if (!access(s, R_OK)) {
#if     SFS_COMPAT
           sprintf(s, "%s/%s", INDEX_DIR, ATTRIBUTE_FILE);
           sprintf(s1, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
           rename(s, s1);
#else
           sprintf(s, "exec %s -f '%s/%s' '%s/.glimpse_tempdir.%d'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), ATTRIBUTE_FILE,
escapesinglequote(INDEX_DIR, es2), pid);
           system(s);
#endif
        }
   }
```

```
   /* Backup old cast-dictionaries: don't use move since indexing might want to use them
*/
   sprintf(s, "%s/.glimpse_quick", INDEX_DIR);
   if (!access(s, R_OK)) {       /* there are previous cast dictionaries */
         backup = rand();
         sprintf(s, "%s/.glimpse_backup.%x", INDEX_DIR, backup);
         if (-1 == mkdir(s, 0700)) {
            fprintf(stderr, "cannot create backup directory %s\n", s);
            return -1;
         }
         sprintf(s, "exec %s -f '%s/.glimpse_quick' '%s/.glimpse_backup.%x'\n",
SYSTEM_CP, escapesinglequote(INDEX_DIR, es1), escapesinglequote(INDEX_DIR,
es2), backup);
         system(s);
         sprintf(s, "exec %s -f '%s/.glimpse_compress' '%s/.glimpse_backup.%x'\n",
SYSTEM_CP, escapesinglequote(INDEX_DIR, es1), escapesinglequote(INDEX_DIR,
es2), backup);
         system(s);
         sprintf(s, "exec %s -f '%s/.glimpse_compress.index' '%s/.glimpse_backup.%x'\n",
SYSTEM_CP, escapesinglequote(INDEX_DIR, es1), escapesinglequote(INDEX_DIR,
es2), backup);
         system(s);
         sprintf(s, "exec %s -f '%s/.glimpse_uncompress' '%s/.glimpse_backup.%x'\n",
SYSTEM_CP, escapesinglequote(INDEX_DIR, es1), escapesinglequote(INDEX_DIR,
es2), backup);
         system(s);
         sprintf(s, "exec %s -f '%s/.glimpse_uncompress.index'
'%s/.glimpse_backup.%x'\n", SYSTEM_CP, escapesinglequote(INDEX_DIR, es1),
escapesinglequote(INDEX_DIR, es2), backup);
         system(s);
         printf("Saved previous cast-dictionary in %s/.glimpse_backup.%x\n",
INDEX_DIR, backup);
   }

   /* Now index these files, and build new dictionaries */
   partition(argc, argv);
   initialize_data_structures(file_num);
   old_file_num = file_num;
   build_index();

   cleanup();
   save_data_structures();
   destroy_filename_hashtable();
   uninitialize_common();
   uninitialize_tcompress();
   uninitialize_tuncompress();
```

```
   compute_dictionary(threshold, DISKBLOCKSIZE, specialwords, INDEX_DIR);

   if (CompressAfterBuild) {
        /* For the new compression */
        if (!initialize_tcompress(hash_file, freq_file, TC_ERRORMSGS)) goto
docleanup;
        printf("Compressing files with new dictionary...\n");
        /* Use the set of file-names collected during partition() / modified during
build_hash */
        for(i=0; i<file_num; i++) {
           if ((disable_list != NULL) && (disable_list[block2index(i)] &
mask_int[i%(8*sizeof(int))])) continue;       /* nop since disable_list IS NULL */
           strcpy(name, LIST_GET(name_list, i));
           tcompress_file(name, outname, TC_REMOVE | TC_EASYSEARCH |
TC_OVERWRITE | TC_NOPROMPT);
        }
   }

docleanup:
   /* Restore old search-dictionaries */
   sprintf(s, "%s/.glimpse_tempdir.%d/.glimpse_index", INDEX_DIR, pid);
   if (!access(s, R_OK)) {
#if      SFS_COMPAT
        sprintf(s1, "%s/%s", INDEX_DIR, INDEX_FILE);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, INDEX_FILE);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, P_TABLE);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, P_TABLE);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, NAME_LIST);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, NAME_LIST);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, NAME_LIST_INDEX);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid,
NAME_LIST_INDEX);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, NAME_HASH);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, NAME_HASH);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, NAME_HASH_INDEX);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid,
NAME_HASH_INDEX);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, MINI_FILE);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, MINI_FILE);
        rename(s, s1);
```

```
        sprintf(s1, "%s/%s", INDEX_DIR, DEF_STAT_FILE);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid, DEF_STAT_FILE);
        rename(s, s1);
        sprintf(s1, "%s/%s", INDEX_DIR, ATTRIBUTE_FILE);
        sprintf(s, "%s/.glimpse_tempdir.%d/%s", INDEX_DIR, pid,
ATTRIBUTE_FILE);
        rename(s, s1);
#else
        /* sprintf(s, "exec %s -f %s/.glimpse_tempdir.%d/.glimpse_* %s\n",
SYSTEM_MV, INDEX_DIR, pid, INDEX_DIR); */
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, escapesinglequote(INDEX_FILE, es2),
INDEX_DIR);
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, P_TABLE, escapesinglequote(INDEX_DIR,
es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, NAME_LIST,
escapesinglequote(INDEX_DIR, es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, NAME_LIST_INDEX,
escapesinglequote(INDEX_DIR, es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, NAME_HASH,
escapesinglequote(INDEX_DIR, es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, NAME_HASH_INDEX,
escapesinglequote(INDEX_DIR, es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, MINI_FILE, escapesinglequote(INDEX_DIR,
es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, DEF_STAT_FILE,
escapesinglequote(INDEX_DIR, es2));
        system(s);
        sprintf(s, "exec %s -f '%s/.glimpse_tempdir.%d/%s' '%s'\n", SYSTEM_MV,
escapesinglequote(INDEX_DIR, es1), pid, ATTRIBUTE_FILE,
escapesinglequote(INDEX_DIR, es2));
        system(s);
```

```
#endif
        sprintf(s, "%s/.glimpse_tempdir.%d", INDEX_DIR, pid);
        rmdir(s);
   }
   printf("\nBuilt new cast-dictionary in %s\n", INDEX_DIR);

#else   /*BUILDCAST*/
        /*printf("\nWhere am I?");*/
   if (AddToIndex || DeleteFromIndex || FastIndex) {
        /* Not handling byte level indices here for now */
        int     indextype = 0, indexnumber = OFF, structuredindex = OFF,
recordlevelindex = OFF, temp_attr_num = 0, bytelevelindex = OFF;
        char    temp_rdelim[MAX_LINE_LEN];

        sprintf(s, "%s/%s", INDEX_DIR, INDEX_FILE);
        if (-1 == stat(s, &istbuf)) {
           if (AddToIndex || DeleteFromIndex) {
                fprintf(stderr, "Cannot find previous index %s! Fresh indexing
recommended\n", s);
                return usage(0);
           }
           file_num = 0;
           file_id = 0;
           part_num = 1;
           goto fresh_indexing;
        }

        /* Find out existing index of words and partitions/filenumbers */
        if ((indextype = get_index_type(s, &indexnumber, &indextype, &structuredindex,
temp_rdelim)) < 0) {
#if     0
           fprintf(stderr, "Fresh indexing recommended: -a and -f are not supported with -
b as yet\n");
           exit(1);
           /* we support it now */
#endif
        }
        if (structuredindex == -2) {
           recordlevelindex = 1;
           bytelevelindex = 1;
        }
        if (structuredindex <= 0) structuredindex = 0;
        else {
           temp_attr_num = structuredindex;
           structuredindex = 1;
        }
```

```
        file_num = part_num = 0;
        sprintf(s, "%s/%s", INDEX_DIR, NAME_LIST);
        file_num = get_array_of_lines(s, name_list, MaxNum24bPartition, 1);
        initialize_disable_list(file_num);
        initialize_data_structures(file_num);

        if (!indextype) {
                sprintf(s, "%s/%s", INDEX_DIR, P_TABLE);
                part_num = get_table(s, p_table, MAX_PARTITION, 1) - 1;          /*
part_num INCLUDES last partition */
        }
        else merge_splits();

        /* Check for errors, Set OneFilePerBlock */
        if ( (file_num <= 0) || (!indextype && (part_num <= 0)) ) {
           if (AddToIndex || DeleteFromIndex) {
                fprintf(stderr, "Cannot find previous glimpseindex files! Fresh indexing
recommended\n");
                return usage(0);
           }
           file_num = 0;
           file_id = 0;
           part_num = 1;
           my_free(disable_list);
           disable_list = NULL;
           goto fresh_indexing;
        }

        if (OneFilePerBlock && !indextype) {
           fprintf(stderr, "Warning: ignoring option -o: using format of existing index\n");
           OneFilePerBlock = 0;
           ByteLevelIndex = 0;
        }
        else {
           OneFilePerBlock = abs(indextype);
           if (indextype < 0) ByteLevelIndex = ON;
        }
        if (StructuredIndex && !structuredindex) {
           fprintf(stderr, "Warning: ignoring option -s: using format of existing index\n");
           StructuredIndex = 0;
           attr_num = 0;
        }
        else {
           StructuredIndex = structuredindex;
           attr_num = temp_attr_num;
        }
```

```
        if (RecordLevelIndex && !recordlevelindex) {
           fprintf(stderr, "Warning: ignoring option -r: using format of existing index\n");
           RecordLevelIndex = 0;
           ByteLevelIndex = 0;
           rdelim[0] = '\0';
           old_rdelim[0] = '\0';
           rdelim_len = 0;
        }
        else {
           RecordLevelIndex = recordlevelindex;
           strcpy(old_rdelim, temp_rdelim);
           strcpy(rdelim, old_rdelim);
           rdelim_len = strlen(rdelim);
           preprocess_delimiter(rdelim, rdelim_len, rdelim, &rdelim_len);
        }

        /* Used in FastIndex for all existing files, used in AddToIndex/DeleteFromIndex
if we are trying to add/remove an existing file */
        build_filename_hashtable(name_list, file_num);

#if     0
        /* Test if these are inverses of each other */
        save_data_structures();
        merge_splits();
#endif  /*0*/

        /*
         * FastIndex: set disable-flag for unchanged files: remove AND
         * disable non-existent files. Let hole remain in file-names/partitions.
         */
        if (FastIndex) {
           for (i=0; i<file_num; i++)
                if (-1 == my_stat(LIST_GET(name_list, i), &stbuf)) {
                        remove_filename(i, -1);
                }
                else if (((stbuf.st_mode & S_IFMT) == S_IFREG) && (stbuf.st_ctime <=
istbuf.st_ctime)) {
                   /* This is just used as a cache since exclude/include processing is not
done here: see dir.c */
                   disable_list[block2index(i)] |= mask_int[i % (8*sizeof(int))];
                }
                else {
                   /* Can't do it for directories since files in it can be modified w/o date
reflected in the directory. Same for symlinks. */
                   LIST_ADD(size_list, i, stbuf.st_size, int);
                   disable_list[block2index(i)] &= ~(mask_int[i % (8*sizeof(int))]);
```

```
                }
        }
        /*
         * AddToIndex without FastIndex: disable all existing files, remove those that
don't exist now.
         * Out of old ones, only ADDED FILES are re-enabled: dir.c
         */
        else if (AddToIndex) {
           for (i=0; i<file_num; i++) {
                if (-1 == my_stat(LIST_GET(name_list, i), &stbuf)) {
                   remove_filename(i, -1);
                }
                else {
                   LIST_ADD(size_list, i, stbuf.st_size, int); /* ONLY for proper statistics
in save_data_structures() */
                   disable_list[block2index(i)] |= mask_int[i % (8*sizeof(int))];
                }
           }
        }
        /* else: DeleteFromIndex without FastIndex: don't touch other files */

        old_file_num = file_num;
        destroy_data_structures();

        /* Put old/new files into partitions/filenumbers */
        if (-1 == oldpartition(argc, argv)) {
           for(i=0;i<file_num;i++) {
#if      BG_DEBUG
                memory_usage -= (strlen(LIST_GET(name_list, i)) + 2);
#endif  /*BG_DEBUG*/
                if (LIST_GET(name_list, i) != NULL) {
                        my_free(LIST_GET(name_list, i), 0);
                        LIST_SUREGET(name_list, i) = NULL;
                }
           }
           file_num = 0;
           file_id = 0;
           for (i=0;i<part_num; i++) {
                p_table[i] = 0;
           }
           part_num = 1;
           my_free(disable_list);
           disable_list = NULL;
           goto fresh_indexing;
        }
```

```
        /* Reindex all the files but use the file-names obtained with oldpartition() */
        if (cross_boundary(OneFilePerBlock, file_num)) {
           my_free(disable_list);
           disable_list = NULL;
        }

        initialize_data_structures(file_num);
        if (!DeleteFromIndex || FastIndex) build_index();
        if ((deletedlist = get_removed_indices()) == NULL) new_file_num = file_num;
        else if (PurgeIndex) new_file_num = purge_index();

#if     BG_DEBUG
        fprintf(LOGFILE, "Built indices in %s/%s\n", INDEX_DIR, INDEX_FILE);
#endif  /*BG_DEBUG*/
        goto docleanup;
   }

fresh_indexing:
   /* remove it to create space since it can be large: don't need for fresh indexing */
   sprintf(s, "%s/%s", INDEX_DIR, P_TABLE);
   unlink(s);
   /* These should be zeroed since they can confuse fsize and fsize_directory() */
   AddToIndex = 0;
   FastIndex = 0;
#if     BG_DEBUG
   fprintf(LOGFILE, "Commencing fresh indexing\n");
#endif  /*BG_DEBUG*/
   partition(argc, argv);
   destroy_filename_hashtable();
   initialize_data_structures(file_num);
   old_file_num = file_num;
   build_index();
#if     BG_DEBUG
   fprintf(LOGFILE, "\nBuilt indices in %s/%s\n", INDEX_DIR, INDEX_FILE);
#endif  /*BG_DEBUG*/

docleanup:
   cleanup();
   save_data_structures();
   destroy_filename_hashtable();
#if     BG_DEBUG
   fflush(LOGFILE);
   fclose(LOGFILE);
#endif  /*BG_DEBUG*/
   fflush(MESSAGEFILE);
   fclose(MESSAGEFILE);
```

```
    fflush(STATFILE);
    fclose(STATFILE);
    if (AddedMaxWordsMessage) printf("\nSome files contributed > %d words to the
index: check %s\n", MAXWORDSPERFILE, DEF_MESSAGE_FILE);
    if (AddedMixedWordsMessage) printf("Some files had numerals in > %d%% of the
indexed words: check %s\n", NUMERICWORDPERCENT, DEF_MESSAGE_FILE);
/*
    printf("\nIndex-directory: \"%s\"\nGlimpse-files created here:\n", INDEX_DIR);
    chdir(INDEX_DIR);
    sprintf(s, "exec %s -l .glimpse_* > %s/%d\n", SYSTEM_LS, TEMP_DIR,pid);
    system(s);
    sprintf(s, "%s/%d", TEMP_DIR,pid);
    if ((tmpfp = fopen(s, "r")) != NULL) {
        memset(tmpbuf, '\0', 1024);
        while(fgets(tmpbuf, 1024, tmpfp) != NULL) fputs(tmpbuf, stdout);
        fflush(tmpfp);
        fclose(tmpfp);
        unlink(s);
    }
    else fprintf(stderr, "cannot open %s to `cat': check %s for .glimpse - files\n", s,
INDEX_DIR);
*/
#endif  /*BUILDCAST*/

    if (!ATLEASTONEFILE) exit(1);
    return 0;
}

cleanup()
{
    char s[MAX_LINE_LEN];

    sprintf(s, "%s/%s", INDEX_DIR, I1);
    unlink(s);
    sprintf(s, "%s/%s", INDEX_DIR, I2);
    unlink(s);
    sprintf(s, "%s/%s", INDEX_DIR, I3);
    unlink(s);
    sprintf(s, "%s/%s", INDEX_DIR, O1);
    unlink(s);
    sprintf(s, "%s/%s", INDEX_DIR, O2);
    unlink(s);
    sprintf(s, "%s/%s", INDEX_DIR, O3);
    unlink(s);
    sprintf(s, "%s/.glimpse_apply.%d", INDEX_DIR, getpid());
    unlink(s);
```

```
   return(1); /* We return 1, because function expect int as return type */
}

#if       !BUILDCAST
usage(flag)
int       flag;
{
          fprintf(stderr, "usage: %s [-help] [-a] [-d] [-f] [-i] [-n [X]] [-o] [-r delim] [-s] [-t] [-
w X] [-B] [-F] [-H DIR] [-I] [-M X] [-R] [-S X] [-T] [-V] NAMES\n", IProgname);
          fprintf(stderr, "List of options (see %s for more details):\n", GLIMPSE_URL);

          fprintf(stderr, "-help: outputs this menu\n");
          fprintf(stderr, "-a: add given files/directories to an existing index\n");
          fprintf(stderr, "-b: build a (large) byte-level index \n");
          fprintf(stderr, "-B: use a hash table that is 4 times bigger (256k entries instead of
64K) \n");
          fprintf(stderr, "-d NAMES: delete (file or directory) NAMES from an existing
index\n");
          fprintf(stderr, "-D NAMES: delete NAMES from the list of files (but not from the
index!)\n");
          fprintf(stderr, "-E: do not run a check on file types\n");
          fprintf(stderr, "-f: incremental indexing (add all newly modified files)\n");
          fprintf(stderr, "-F: the list of files to index is obtained from standard input\n");
          fprintf(stderr, "-h: generates some hash-tables for WebGlimpse\n");
          fprintf(stderr, "-H DIR: the index is put in directory DIR\n");
          fprintf(stderr, "-i: make .glimpse_include take precedence over
.glimpse_exclude\n");
          fprintf(stderr, "-I: output the list of files that would be indexed (but don't
index)\n");
          fprintf(stderr, "-M X: use X MBytes of memory for temporary tables\n");
        fprintf(stderr, "-n [X]: index numbers as well as words; warn (into
.glimpse_messages)\n\tif file adds > X%% numeric words: default is %d\n",
DEF_NUMERIC_WORD_PERCENT);
          fprintf(stderr, "-o: build a small (rather than tiny) size index (the recommended
option!)\n");
          /*fprintf(stderr, "-O: when using -r option, store byte offset of each
record,\n\tinstead of the record number, for faster access\n");*/
          fprintf(stderr, "-r delim: build an index at the granularity of delimiter `delim'\n\tto
do booleans by reading ONLY the index\n");
          fprintf(stderr, "-R: recompute .glimpse_filenames_index from .glimpse_filenames
if it changes\n");
          fprintf(stderr, "-s: build index to support structured (Harvest SOIF type)
queries\n");
          fprintf(stderr, "-S X: adjust the size of the stop list\n");
          fprintf(stderr, "-t: sort the indexed files by date and time (most recent first)\n");
```

```
        fprintf(stderr, "-T: build .glimpse_turbo for very fast search with -i -w in
glimpse\n");
        fprintf(stderr, "-U: there is extra information after filenames: works only with -
F\n");
        fprintf(stderr, "-w X: warn (into .glimpse_messages) if a file adds >= X words to
the index\n");
        fprintf(stderr, "-X: extract titles of all documents with .html, .htm, .shtm, .shtml
suffix\n");
        fprintf(stderr, "-z: customizable filtering using .glimpse_filters \n");

        fprintf(stderr, "\n");
        fprintf(stderr, "For questions about glimpse, please contact: `%s'\n",
GLIMPSE_EMAIL);
        exit(1);
}
#else   /*!BUILDCAST*/
usage(flag)
int     flag;
{
        if (flag) fprintf(stderr, "\nThis is buildcast version %s, %s.\n\n",
GLIMPSE_VERSION, GLIMPSE_DATE);
        fprintf(stderr, "usage: %s [-help] [-t] [-i] [-l] [-n [X]] [-w X] [-C] [-E] [-F] [-H
DIR] [-V] NAMES\n", IProgname);
      fprintf(stderr, "summary of frequently used options\n(for a more detailed listing see
'man cast'):\n");
        fprintf(stderr, "-help: output this menu\n");
      fprintf(stderr, "-n [X]: index numbers as well as words; warn (into
.glimpse_messages)\n\tif file adds > X%% numeric words: default is %d\n",
DEF_NUMERIC_WORD_PERCENT);
      fprintf(stderr, "-w X: warn if a file adds > X words to the index\n");
        fprintf(stderr, "-C: compress files with the new dictionary after building it\n");
        fprintf(stderr, "-E: build cast dictionary using existing compressed files only\n");
        fprintf(stderr, "-F: expect filenames on stdin (useful for pipelining)\n");
      fprintf(stderr, "-H DIR: .glimpse-files should be in directory DIR: default is '~'\n");
        fprintf(stderr, "\n");
        fprintf(stderr, "For questions about glimpse, please contact: `%s'\n",
GLIMPSE_EMAIL);
        exit(1);
        return(1); /* We type return 1, because function expect int as return type */
}
#endif  /*!BUILDCAST*/
```

APPENDIX F: LEMUR – INDRIBUILDINDEX.CPP

IndriBuildIndex.cpp was modified to normalize input and output behavior. Prompts were eliminated and a single command was enabled.

```
/*==========================================================
* Copyright (c) 2004 University of Massachusetts.  All Rights Reserved.
 *
 * Use of the Lemur Toolkit for Language Modeling and Information Retrieval
 * is subject to the terms of the software license set forth in the LICENSE
 * file included with this software, and also available at
 * http://www.lemurproject.org/license.html
 *
 *==========================================================*/

//
// buildindex
//
// 10 February 2004 -- tds
//
// Indri index build driver application
//

/*! \page IndriParameters Indri Parameter Files

<P>The indri applications, IndriBuildIndex, IndriDaemon, and
IndriRunQuery accept parameters from either the command line or from a
file. The parameter file uses an XML format. The command line uses
dotted path notation. The top level element in the parameters file is
named <em>parameters</em>.

<H3> Repository construction parameters</h3>
<dl>
<dt>memory</dt>
<dd> an integer value specifying the number of bytes to use for the
indexing process. The value can include a scaling factor by adding a
suffix. Valid values are (case insensitive) K = 1000, M = 1000000, G =
1000000000. So 100M would be equivalent to 100000000. The value should
contain only decimal digits and the optional suffix. Specified as
&lt;memory&gt;100M&lt;/memory&gt; in the parameter file and as
<tt>-memory=100M</tt> on the command line. </dd>
<dt>corpus</dt>
```

<dd>a complex element containing parameters related to a corpus. This
element can be specified multiple times. The parameters are
<dl>
<dt>path</dt>
<dd>The pathname of the file or directory containing documents to
index. Specified as
<corpus><path>/path/to/file_or_directory</path></corpus>
in the parameter file and as
<tt>-corpus.path=/path/to/file_or_directory</tt> on the command
line.</dd>
<dt>class</dt>
<dd>The FileClassEnviroment of the file or directory containing
documents to index. Specified as
<corpus><class>trecweb</class></corpus> in the
parameter file and as <tt>-corpus.class=trecweb</tt> on the command
line. The known classes are:

html -- web page data.
trecweb -- TREC web format, eg terabyte track.
trectext -- TREC format, eg TREC-3 onward.
trecalt -- TREC format, eg TREC-3 onward, with only the TEXT field included.
doc -- Microsoft Word format (windows platform only).
ppt -- Microsoft Powerpoint format (windows platform only).
pdf -- Adobe PDF format.
txt -- Plain text format.

</dd>
<dt>annotations</dt>
<dd>The pathname of the file containing offset annotations for the documents
specified in <tt>path</tt>. Specified as
<corpus><annotations>/path/to/file</annotations></corpus>
in the parameter file and as
<tt>-corpus.annotations=/path/to/file</tt> on the command
line.</dd>
<dt>metadata</dt>
<dd>The pathname of the file or directory containing offset metadata for the
documents specified in <tt>path</tt>. Specified as
<corpus><metadata>/path/to/file</metadata></corpus>
in the parameter file and as
<tt>-corpus.metadata=/path/to/file</tt> on the command
line.</dd>

Combining the first two of these elements, the parameter file would contain:

<corpus>
  <path>/path/to/file_or_directory</path>

  <class>trecweb</class>
</corpus>
</dd>
</dl>

<dt>metadata</dt>
<dd>a complex element containing one or more entries
specifying the metadata fields to index, eg title, headline.
There are three options

 <tt>field</tt> -- Make the named field available for retrieval as
metadata. Specified as
<metadata><field>fieldname</field></metadata>
in the parameter file and as <tt>metadata.field=fieldname</tt> on the
command line.

 <tt>forward</tt> -- Make the named field available for retrieval as
metadata and build a lookup table to make retrieving the value more
efficient. Specified as
<metadata><forward>fieldname</forward></metadata>
in the parameter file and as <tt>metadata.forward=fieldname</tt> on the
command line. The external document id field "docno" is automatically
added as a forward metadata field.

 <tt>backward</tt> -- Make the named field available for retrieval
as metadata and build a lookup table for inverse lookup of documents
based on the value of the field. Specified as
<metadata><backward>fieldname</backward></metadata>
in the parameter file and as <tt>metadata.backward=fieldname</tt> on
the command line. The external document id field "docno" is automatically
added as a backward metadata field.

</dd>

<dt>field</dt>

<dd>a complex element specifying the fields to index as data, eg
TITLE. This parameter can appear multiple times in a parameter file.
If provided on the command line, only the first field specified will
be indexed. The subelements are:

<dl>
<dt>name</dt><dd>the field name, specified as
<field><name>fieldname</name></field> in the

parameter file and as <tt>-field.name=fieldname</tt> on the command
line.</dd>
<dt>numeric</dt><dd>the symbol <tt>true</tt> if the field contains
numeric data, otherwise the symbol <tt>false</tt>, specified as
<field><numeric>true</numeric></field> in the
parameter file and as <tt>-field.numeric=true</tt> on the command
line. This is an optional parameter, defaulting to false. Note that <tt>0</tt>
can be used for false and <tt>1</tt> can be used for true. </dd>
<dt>parserName</dt><dd>the name of the parser to use to convert a numeric
field to an unsigned integer value. The default is NumericFieldAnnotator. If numeric field
data is provided via offset annotations, you should use the value
OffsetAnnotationAnnotator. If the field contains a formatted date (see <a
href="DateFields.html">Date Fields) you should use the value DateFieldAnnotator.
</dd>
</dl>
</dd>
<dt>stemmer</dt>
<dd>a complex element specifying the stemming algorithm to use in the
subelement name. Valid options are Porter or Krovetz (case
insensitive). Specified as
<stemmer><name>stemmername</name></stemmer> and
as <tt>-stemmer.name=stemmername</tt> on the command line. This is an
optional parameter with the default of no stemming.
</dd>
<dt>stopper</dt>
<dd>a complex element containing one or more subelements named word,
specifying the stopword list to use. Specified as
<stopper><word>stopword</word></stopper> and
as <tt>-stopper.word=stopword</tt> on the command line. This is an
optional parameter with the default of no stopping.</dd>
<dt>offsetannotationhint</dt>
<dd>An optional parameter to provide a hint to the indexer to speed
up indexing of offset annotations when using offset annotation files
as specified in the <corpus> parameter. Valid values here
are "unordered" and "ordered". An "unordered"
hint (the default) will inform the indexer that the document IDs of the
annotations are not necessarily in the same order as the documents
in the corpus. The indexer will adjust its internal memory allocations
appropriately to pre-allocate enough memory before reading in the annotations
file. If you are absolutely certain that the annotations in the offset annotation
file are in the exact same order as the documents, then you can use the
"ordered"
hint. This will tell the indexer to not read in the entire file at once, but rather
read in the offset annotations file as needed for only the annotations that
are specified for the currently indexing document ID.</dd>
</dl>

<H3>QueryEnvironment Parameters</H3>
<H4>Retrieval Parameters</H4>
<dl>
<dt>memory</dt>
<dd> an integer value specifying the number of bytes to use for the
query retrieval process. The value can include a scaling factor by
adding a suffix. Valid values are (case insensitive) K = 1000, M =
1000000, G = 1000000000. So 100M would be equivalent to 100000000. The
value should contain only decimal digits and the optional
suffix. Specified as <memory>100M</memory> in the parameter
file and as <tt>-memory=100M</tt> on the command line. </dd>
<dt>index</dt>
<dd> path to an Indri Repository. Specified as
<index>/path/to/repository</index> in the parameter file and
as <tt>-index=/path/to/repository</tt> on the command line. This element
can be specified multiple times to combine Repositories.
</dd>
<dt>server</dt>
<dd> hostname of a host running an Indri server (IndriDaemon). Specified as
<server>hostname</server> in the parameter file and
as <tt>-server=hostname</tt> on the command line. The hostname can
include an optional port number to connect to, using the form
<tt>hostname:portnum</tt>. This element
can be specified multiple times to combine servers.
</dd>
<dt>count</dt>
<dd>an integer value specifying the maximum number of results to
return for a given query. Specified as
<count>number</count> in the parameter file and
as <tt>-count=number</tt> on the command line. </dd>
<dt>query</dt>
<dd>An indri query language query to run. This element can be specified
multiple times.
</dd>
<dt>rule</dt>
<dd>specifies the smoothing rule (TermScoreFunction) to apply. Format of
the rule is:

<tt> (key ":" value) ["," key ":" value]* </tt>
<p>
Here's an example rule in command line format:

  <tt>-rule=method:linear,collectionLambda:0.2,field:title</tt>
<p> and in parameter file format:
<tt>

<rule>method:linear,collectionLambda:0.2,field:title</rule>
</tt>

<p>This corresponds to Jelinek-Mercer smoothing with background lambda
equal to 0.2, only for items in a title field.

<p>If nothing is listed for a key, all values are assumed.
So, a rule that does not specify a field matches all fields. This makes
<tt>-rule=method:linear,collectionLambda:0.2</tt> a valid rule.

<p>Valid keys:
<dl>
<dt> method</dt><dd> smoothing method (text)</dd>
<dt> field</dt><dd> field to apply this rule to</dd>
<dt> operator
<dd> type of item in query to apply to { term, window }</dd>
</dl>

<p>Valid methods:
<dl>
<dt> dirichlet</dt><dd> (also 'd', 'dir') (default mu=2500)</dd>
<dt> jelinek-mercer</dt><dd> (also 'jm', 'linear') (default
collectionLambda=0.4, documentLambda=0.0), collectionLambda is also
known as just "lambda", either will work </dt>
<dt> twostage</dt><dd> (also 'two-stage', 'two') (default mu=2500,
lambda=0.4)</dd>
</dl>
If the rule doesn't parse correctly, the default is Dirichlet, mu=2500.
</dd>
<dt>stopper</dt>
<dd>a complex element containing one or more subelements named word,
specifying the stopword list to use. Specified as
<stopper><word>stopword</word></stopper> and
as <tt>-stopper.word=stopword</tt> on the command line. This is an
optional parameter with the default of no stopping.</dd>
<dt>maxWildcardTerms</dt>
<dd>
(optional) An integer specifying the maximum number of wildcard terms that can
be generated for a synonym list for this query or set of queries. If this limit
is reached for a wildcard term, an exception will be thrown. If this parameter
is not specified, a default of 100 will be used.
</dd>
</dl>
<H4>Formatting Parameters</H4>
<dl>
<dt>queryOffset</dt>

<dd>an integer value specifying one less than the starting query number,
eg 150 for TREC formatted output. Specified as
<queryOffset>number</queryOffset> in the parameter file and
as <tt>-queryOffset=number</tt> on the command line.</dd>
<dt>runID</dt>
<dd>a string specifying the id for a query run, used in TREC scorable
output. Specified as
<runID>someID</runID> in the parameter file and
as <tt>-runID=someID</tt> on the command line.</dd>
<dt>trecFormat</dt>
<dd>the symbol <tt>true</tt> to produce TREC scorable output, otherwise
the symbol <tt>false</tt>. Specified as
<trecFormat>true</trecFormat> in the parameter file and
as <tt>-trecFormat=true</tt> on the command line. Note that <tt>0</tt>
can be used for false, and <tt>1</tt> can be used for true.</dd>
</dl>
<H4>Pseudo-Relevance Feedback Parameters</H4>
<dl>
<dt>fbDocs</dt>
<dd>an integer specifying the number of documents to use for
feedback. Specified as
<fbDocs>number</fbDocs> in the parameter file and
as <tt>-fbDocs=number</tt> on the command line.</dd>
<dt>fbTerms</dt>
<dd>an integer specifying the number of terms to use for
feedback. Specified as
<fbTerms>number</fbTerms> in the parameter file and
as <tt>-fbTerms=number</tt> on the command line.</dd>
<dt>fbMu</dt>
<dd>a floating point value specifying the value of mu to use for
feedback. Specified as
<fbMu>number</fbMu> in the parameter file and
as <tt>-fbMu=number</tt> on the command line.</dd>
<dt>fbOrigWeight</dt>
<dd>a floating point value in the range [0.0..1.0] specifying the weight
for the original query in the expanded query. Specified as
<fbOrigWeight>number</fbOrigWeight> in the parameter file and
as <tt>-fbOrigWeight=number</tt> on the command line.</dd>
</dl>

<H3>IndriDaemon Parameters</H3>
<dl>
<dt>memory</dt>
<dd> an integer value specifying the number of bytes to use for the
query retrieval process. The value can include a scaling factor by
adding a suffix. Valid values are (case insensitive) K = 1000, M =

1000000, G = 1000000000. So 100M would be equivalent to 100000000. The value should contain only decimal digits and the optional suffix. Specified as <memory>100M</memory> in the parameter file and as <tt>-memory=100M</tt> on the command line. </dd>
<dt>index</dt>
<dd> path to the Indri Repository to act as server for. Specified as <index>/path/to/repository</index> in the parameter file and as <tt>-index=/path/to/repository</tt> on the command line.
</dd>
<dt>port</dt>
<dd> an integer value specifying the port number to use.Specified as <port>number</port> in the parameter file and as <tt>-port=number</tt> on the command line. </dd>
</dl>

*/
/*! \page IndriIndexer Indri Repository Builder
<P>
This application builds an Indri Repository for a collection of documents.
Parameter formats for all Indri applications are also described in
IndriParameters.html
<H3> Repository construction parameters</h3>
<dl>
<dt>memory</dt>
<dd> an integer value specifying the number of bytes to use for the indexing process. The value can include a scaling factor by adding a suffix. Valid values are (case insensitive) K = 1000, M = 1000000, G = 1000000000. So 100M would be equivalent to 100000000. The value should contain only decimal digits and the optional suffix. Specified as <memory>100M</memory> in the parameter file and as <tt>-memory=100M</tt> on the command line. </dd>
<dt>corpus</dt>
<dd>a complex element containing parameters related to a corpus. This element can be specified multiple times. The parameters are
<dl>
<dt>path</dt>
<dd>The pathname of the file or directory containing documents to index. Specified as <corpus><path>/path/to/file_or_directory</path></corpus> in the parameter file and as <tt>-corpus.path=/path/to/file_or_directory</tt> on the command line.</dd>
<dt>class</dt>
<dd>The FileClassEnviroment of the file or directory containing documents to index. Specified as <corpus><class>trecweb</class></corpus> in the

parameter file and as <tt>-corpus.class=trecweb</tt> on the command
line. The known classes are:

html -- web page data.
trecweb -- TREC web format, eg terabyte track.
trectext -- TREC format, eg TREC-3 onward.
trecalt -- TREC format, eg TREC-3 onward, with only the TEXT field included.
doc -- Microsoft Word format (windows platform only).
ppt -- Microsoft Powerpoint format (windows platform only).
pdf -- Adobe PDF format.
txt -- Plain text format.

</dd>
<dt>annotations</dt>
<dd>The pathname of the file containing offset annotations for the documents
specified in <tt>path</tt>. Specified as
<corpus><annotations>/path/to/file</annotations></corpus>
in the parameter file and as
<tt>-corpus.annotations=/path/to/file</tt> on the command
line.</dd>
<dt>metadata</dt>
<dd>The pathname of the file or directory containing offset metadata for the
documents specified in <tt>path</tt>. Specified as
<corpus><metadata>/path/to/file</metadata></corpus>
in the parameter file and as
<tt>-corpus.metadata=/path/to/file</tt> on the command
line.</dd>

Combining the first two of these elements, the parameter file would contain:

<corpus>
  <path>/path/to/file_or_directory</path>
  <class>trecweb</class>
</corpus>
</dd>
</dl>

<dt>metadata</dt>
<dd>a complex element containing one or more entries
specifying the metadata fields to index, eg title, headline.
There are three options

 <tt>field</tt> -- Make the named field available for retrieval as
 metadata. Specified as
 <metadata><field>fieldname</field></metadata>

in the parameter file and as <tt>metadata.field=fieldname</tt> on the command line.

 <tt>forward</tt> -- Make the named field available for retrieval as metadata and build a lookup table to make retrieving the value more efficient. Specified as <metadata><forward>fieldname</forward></metadata> in the parameter file and as <tt>metadata.forward=fieldname</tt> on the command line.

 <tt>backward</tt> -- Make the named field available for retrieval as metadata and build a lookup table for inverse lookup of documents based on the value of the field. Specified as <metadata><backward>fieldname</backward></metadata> in the parameter file and as <tt>metadata.backward=fieldname</tt> on the command line.

</dd>

<dt>field</dt>

<dd>a complex element specifying the fields to index as data, eg TITLE. This parameter can appear multiple times in a parameter file. If provided on the command line, only the first field specified will be indexed. The subelements are:

<dl>
<dt>name</dt><dd>the field name, specified as <field><name>fieldname</name></field> in the parameter file and as <tt>-field.name=fieldname</tt> on the command line.</dd>
<dt>numeric</dt><dd>the symbol <tt>true</tt> if the field contains numeric data, otherwise the symbol <tt>false</tt>, specified as <field><numeric>true</numeric></field> in the parameter file and as <tt>-field.numeric=true</tt> on the command line. This is an optional parameter, defaulting to false. Note that <tt>0</tt> can be used for false and <tt>1</tt> can be used for true. </dd>
<dt>parserName</dt><dd>the name of the parser to use to convert a numeric field to an unsigned integer value. The default is NumericFieldAnnotator. If numeric field data is provided via offset annotations, you should use the value OffsetAnnotationAnnotator. If the field contains a formatted date (see Date Fields) you should use the value DateFieldAnnotator.
</dd>
</dl>
</dd>

<dt>stemmer</dt>
<dd>a complex element specifying the stemming algorithm to use in the
subelement name. Valid options are Porter or Krovetz (case
insensitive). Specified as
<stemmer><name>stemmername</name></stemmer> and
as <tt>-stemmer.name=stemmername</tt> on the command line. This is an
optional parameter with the default of no stemming.
</dd>
<dt>stopper</dt>
<dd>a complex element containing one or more subelements named word,
specifying the stopword list to use. Specified as
<stopper><word>stopword</word></stopper> and
as <tt>-stopper.word=stopword</tt> on the command line. This is an
optional parameter with the default of no stopping.</dd>
<dt>offsetannotationhint</dt>
<dd>An optional parameter to provide a hint to the indexer to speed
up indexing of offset annotations when using offset annotation files
as specified in the <corpus> parameter. Valid values here
are "unordered" and "ordered". An "unordered"
hint (the default) will inform the indexer that the document IDs of the
annotations are not necessarily in the same order as the documents
in the corpus. The indexer will adjust its internal memory allocations
appropriately to pre-allocate enough memory before reading in the annotations
file. If you are absolutely certain that the annotations in the offset annotation
file are in the exact same order as the documents, then you can use the
"ordered"
hint. This will tell the indexer to not read in the entire file at once, but rather
read in the offset annotations file as needed for only the annotations that
are specified for the currently indexing document ID.</dd>
</dl>

*/

```
#include <algorithm>
#include <cstring>
#include <string>
#include <cctype>

#include "indri/Parameters.hpp"
#include "indri/IndexEnvironment.hpp"
#include <time.h>
#include "indri/Path.hpp"
#include "indri/ConflationPattern.hpp"
#include "Exception.hpp"
#include "indri/FileTreeIterator.hpp"
#include <vector>
```

```
#include <map>
#include "indri/IndriTimer.hpp"

#include "indri/QueryEnvironment.hpp"
#include "indri/Thread.hpp"
#include "indri/SequentialWriteBuffer.hpp"

#include <math.h>
static indri::utility::IndriTimer g_timer;

static void buildindex_start_time() {
//  g_timer.start();
}

static void buildindex_print_status( const char* status, int count ) {
//  g_timer.printElapsedSeconds(std::cout);
//  std::cout << ": " << status << count << "\r";
}

static void buildindex_print_status( const char* status, int count, const char* status2,
INT64 count2 ) {
//  g_timer.printElapsedSeconds(std::cout);
//  std::cout << ": " << status << count << status2 << count2 << "\r";
}

static void buildindex_flush_status() {
  std::cout.flush();
}

static void buildindex_print_event( const char* event ) {
//  g_timer.printElapsedSeconds(std::cout);
//  std::cout << ": " << event << std::endl;
}

static void buildindex_print_event( std::string event ) {
//  buildindex_print_event( event.c_str() );
}

class StatusMonitor : public indri::api::IndexStatus {
  void operator() ( int code, const std::string& documentFile, const std::string& error, int
documentsParsed, int documentsSeen ) {
    std::stringstream event;

/*    switch(code) {
      case indri::api::IndexStatus::FileOpen:
        event << "Hi, I Opened " << documentFile;
```

```
      buildindex_print_event( event.str() );
      break;

    case indri::api::IndexStatus::FileClose:
      buildindex_print_status( "Documents: ", documentsParsed );
      buildindex_print_event( "" );
      event << "Closed " << documentFile;
      buildindex_print_event( event.str() );
      break;

    case indri::api::IndexStatus::FileSkip:
      event << "Skipped " << documentFile;
      buildindex_print_event( event.str() );
      break;

    case indri::api::IndexStatus::FileError:
      event << "Error in " << documentFile << " : " << error;
      buildindex_print_event( event.str() );
      break;

    default:
    case indri::api::IndexStatus::DocumentCount:
      if( !(documentsParsed % 500) )
        buildindex_print_status( "Documents: ", documentsParsed );
        buildindex_flush_status();
      break;
   }*/
  }
};

static std::string downcase_string( const std::string& str ) {
  std::string result;
  result.resize( str.size() );

  for( size_t i=0; i<str.size(); i++ ) {
    result[i] = tolower(str[i]);
  }

  return result;
}

static void downcase_string_vector (std::vector<std::string>& vec) {
  for( size_t i=0; i<vec.size(); i++ ) {
    vec[i] = downcase_string( vec[i] );
  }
}
```

```
static bool copy_parameters_to_string_vector( std::vector<std::string>& vec,
indri::api::Parameters p, const std::string& parameterName, const std::string* subName =
0 ) {
  if( !p.exists(parameterName) )
    return false;

  indri::api::Parameters slice = p[parameterName];

  for( size_t i=0; i<slice.size(); i++ ) {
    if( subName ) {
      if( slice[i].exists(*subName) ) {
        vec.push_back( slice[i][*subName] );
      }
    } else {
      vec.push_back( slice[i] );
    }
  }

  return true;
}

/*! Given a Specification and a field name, return a vector containing all
 * of the field names that conflate to that name as well as the original
 * name.
 * @param spec The indri::parse::FileClassEnvironmentFactory::Specification to inspect.
 * @param name The field name to look for.
 * @return a vector containing all of the field names that conflate to that name as well as
the original name.
 */
static std::vector<std::string>
findConflations(indri::parse::FileClassEnvironmentFactory::Specification *spec,
std::string & name) {
  std::vector<std::string> retval;
  // have to walk the map and add an entry for each
  // conflation to a given name
  std::map<indri::parse::ConflationPattern*, std::string>::const_iterator iter;
  for (iter = spec->conflations.begin();
       iter != spec->conflations.end(); iter++) {
    if( iter->second == name )
      retval.push_back(iter->first->tag_name);
  }
  // put the original into the list
  retval.push_back(name);
  return retval;
}
```

```
/*! Add a string to a vector if not already present.
 * @return true if the string was added.
 */
static bool addNew(std::vector<std::string>& vec, string &name,
                   std::string &specName, const char *msg) {
  bool retval = false;
  if( std::find( vec.begin(), vec.end(), name ) == vec.end() ) {
    std::cerr << "Adding " << name << " to " << specName << msg << std::endl;
    vec.push_back(name);
    retval = true;
  }
  return retval;
}
/*! Add field names to index or metadata for an existing file class
 * specification.
 */
static bool augmentSpec(indri::parse::FileClassEnvironmentFactory::Specification *spec,
                        std::vector<std::string>& fields,
                        std::vector<std::string>& metadata,
                        std::vector<std::string>& metadataForward,
                        std::vector<std::string>& metadataBackward ) {
  // add to index and metadata fields in spec if necessary.
  // return true if a field is changed.
  bool retval = false;
  // input field names are potentially conflated names:
  // eg headline for head, hl, or headline tags.
  std::vector<std::string> conflations;
  std::vector<std::string>::iterator i1;

  for (i1 = fields.begin(); i1 != fields.end(); i1++) {
    // find any conflated names
    conflations = findConflations(spec, *i1);
    for (size_t j = 0; j < conflations.size(); j++) {
      // only add the field for indexing if it doesn't already exist
      if (addNew(spec->index, conflations[j],
                 spec->name, " as an indexed field")) {
        // added a field, make sure it is indexable
        // only add include tags if there are some already.
        // if it is empty, *all* tags are included.
        if( !spec->include.empty() ) {
          addNew(spec->include, conflations[j], spec->name,
                 " as an included tag");
        }
        retval = true;
      }
```

```
    }
  }

  // add fields that should be marked metadata for retrieval
  for (i1 = metadata.begin(); i1 != metadata.end(); i1++) {
    // find any conflated names
    conflations = findConflations(spec, *i1);
    for (size_t j = 0; j < conflations.size(); j++)
      retval |= addNew(spec->metadata, conflations[j], spec->name,
                       " as a metadata field");
  }
  // add fields that should have a metadata forward lookup table.
  for (i1 = metadataForward.begin(); i1 != metadataForward.end(); i1++) {
    // find any conflated names
    conflations = findConflations(spec, *i1);
    for (size_t j = 0; j < conflations.size(); j++)
      retval |= addNew(spec->metadata, conflations[j], spec->name,
                       " as a forward indexed metadata field");
  }
  // add fields that should have a metadata reverse lookup table.
  for (i1 = metadataBackward.begin(); i1 != metadataBackward.end(); i1++) {
    // find any conflated names
    conflations = findConflations(spec, *i1);
    for (size_t j = 0; j < conflations.size(); j++)
      retval |= addNew(spec->metadata, conflations[j], spec->name,
                       " as a forward indexed metadata field");
  }

  return retval;
}

//
// process_numeric_fields
//

static void process_numeric_fields( indri::api::Parameters parameters,
indri::api::IndexEnvironment& env ) {
  std::string numName = "numeric";
  std::string subName = "name";
  indri::api::Parameters slice = parameters["field"];
  for( size_t i=0; i<slice.size(); i++ ) {
    bool isNumeric = slice[i].get(numName, false);
    if( isNumeric ) {
      // let user override default NumericFieldAnnotator for parser
      // enabling numeric fields in offset annotations.
      std::string parserName = slice[i].get("parserName", "NumericFieldAnnotator");
```

```
      std::string fieldName = slice[i][subName];
      fieldName = downcase_string( fieldName );
      env.setNumericField(fieldName, isNumeric, parserName);
    }
  }
}

//
// process_ordinal_fields
//

static void process_ordinal_fields( indri::api::Parameters parameters,
indri::api::IndexEnvironment& env ) {
  std::string ordName = "ordinal";
  std::string subName = "name";
  indri::api::Parameters slice = parameters["field"];

  for( size_t i=0; i<slice.size(); i++ ) {
    bool isOrdinal = slice[i].get(ordName, false);

    if( isOrdinal ) {
      std::string fieldName = slice[i][subName];
      fieldName = downcase_string( fieldName );
      env.setOrdinalField(fieldName, isOrdinal);
    }
  }
}

void require_parameter( const char* name, indri::api::Parameters& p ) {
  if( !p.exists( name ) ) {
    LEMUR_THROW( LEMUR_MISSING_PARAMETER_ERROR, "Must specify a "
+ name + " parameter." );
  }
}

int main(int argc, char * argv[]) {
  try {
    indri::api::Parameters& parameters = indri::api::Parameters::instance();
    parameters.loadCommandLine( argc, argv );

//    require_parameter( "corpus", parameters );
//    require_parameter( "index", parameters );

    StatusMonitor monitor;
    indri::api::IndexEnvironment env;
    std::string repositoryPath = parameters["index"];
```

```
//    buildindex_start_time();

//    if( parameters.get( "version", 0 ) ) {
//      std::cout << INDRI_DISTRIBUTION << std::endl;
//    }

    env.setMemory( parameters.get("memory", INT64(100*1024*1024)) );

    std::string offsetAnnotationHint=parameters.get("offsetannotationhint", "default");
    if (offsetAnnotationHint=="ordered") {
      env.setOffsetAnnotationIndexHint(indri::parse::OAHintOrderedAnnotations);
    } if (offsetAnnotationHint=="unordered") {
      env.setOffsetAnnotationIndexHint(indri::parse::OAHintSizeBuffers);
    } else {
      env.setOffsetAnnotationIndexHint(indri::parse::OAHintDefault);
    }

    std::string stemmerName = parameters.get("stemmer.name", "");
    if( stemmerName.length() )
      env.setStemmer(stemmerName);

    std::vector<std::string> stopwords;
    if( copy_parameters_to_string_vector( stopwords, parameters, "stopper.word" ) )
      env.setStopwords(stopwords);
    // fields to include as metadata (unindexed)
    std::vector<std::string> metadata;
    // metadata fields that should have a forward lookup table.
    std::vector<std::string> metadataForward;
    // metadata fields that should have a backward lookup table.
    std::vector<std::string> metadataBackward;
    copy_parameters_to_string_vector( metadata, parameters, "metadata.field" );
    downcase_string_vector(metadata);

    copy_parameters_to_string_vector( metadataForward, parameters, "metadata.forward"
);
    downcase_string_vector(metadataForward);
    copy_parameters_to_string_vector( metadataBackward, parameters,
"metadata.backward" );
    downcase_string_vector(metadataBackward);
    // docno is a special field, automagically add it as forward and backward.
    std::string docno = "docno";
    if( std::find( metadataForward.begin(),
                   metadataForward.end(),
                   docno ) == metadataForward.end() )
      metadataForward.push_back(docno);
```

```
    if( std::find( metadataBackward.begin(),
                   metadataBackward.end(),
                   docno ) == metadataBackward.end() )
      metadataBackward.push_back(docno);

    env.setMetadataIndexedFields( metadataForward, metadataBackward );

    std::vector<std::string> fields;
    std::string subName = "name";
    if( copy_parameters_to_string_vector( fields, parameters, "field", &subName ) ) {
      downcase_string_vector(fields);
      env.setIndexedFields(fields);
      process_numeric_fields( parameters, env );
      process_ordinal_fields( parameters, env );
    }

    if( indri::collection::Repository::exists( repositoryPath ) ) {
      env.open( repositoryPath, &monitor );
//      buildindex_print_event( std::string() + "Opened repository " + repositoryPath );
    } else {
      env.create( repositoryPath, &monitor );
//      buildindex_print_event( std::string() + "Created repository " + repositoryPath );
    }

    indri::api::Parameters corpus = parameters["corpus"];

    for( unsigned int i=0; i<corpus.size(); i++ ) {
      indri::api::Parameters thisCorpus = corpus[i];
      require_parameter( "path", thisCorpus );
      std::string corpusPath = thisCorpus["path"];
      std::string fileClass = thisCorpus.get("class", "");

      // augment field/metadata tags in the environment if needed.
      if( fileClass.length() ) {
        indri::parse::FileClassEnvironmentFactory::Specification *spec =
env.getFileClassSpec(fileClass);
        if( spec ) {
          // add fields if necessary, only update if changed.
          if( augmentSpec( spec, fields, metadata, metadataForward, metadataBackward ) )
            env.addFileClass(*spec);
          delete(spec);
        }
      }

      bool isDirectory = indri::file::Path::isDirectory( corpusPath );
```

```
    // First record the document root, and then the paths to any annotator inputs
    env.setDocumentRoot( corpusPath );

    // Support for anchor text
    std::string anchorText = thisCorpus.get("inlink", "");
    env.setAnchorTextPath( anchorText );

    // Support for offset annotations
    std::string offsetAnnotationsPath = thisCorpus.get( "annotations", "" );
    env.setOffsetAnnotationsPath( offsetAnnotationsPath );

    // Support for offset metadata file
    std::string offsetMetadataPath = thisCorpus.get( "metadata", "" );
    env.setOffsetMetadataPath( offsetMetadataPath );

    if( isDirectory ) {
      indri::file::FileTreeIterator files( corpusPath );

      for( ; files != indri::file::FileTreeIterator::end(); files++ ) {
        if( fileClass.length() )
          env.addFile( *files, fileClass );
        else
          env.addFile( *files );
      }
    } else {
      if( fileClass.length() )
        env.addFile( corpusPath, fileClass );
      else
        env.addFile( corpusPath );
    }
  }

//   buildindex_print_event( "Closing index" );
  env.close();
//   buildindex_print_event( "Finished" );
 } catch( lemur::api::Exception& e ) {
  LEMUR_ABORT(e);
 }

 return 0;
}
```

APPENDIX G: LEMUR – INDRIRUNQUERY.CPP

IndriRunQuery.cpp was modified to normalize input and output behavior. Prompts were eliminated and a single command was enabled.

```
/*==========================================================
* Copyright (c) 2004 University of Massachusetts.  All Rights Reserved.
 *
 * Use of the Lemur Toolkit for Language Modeling and Information Retrieval
 * is subject to the terms of the software license set forth in the LICENSE
 * file included with this software, and also available at
 * http://www.lemurproject.org/license.html
 *
 *==========================================================*/

//
// runquery
//
// 24 February 2004 -- tds
//
// 18 August 2004 -- dam
// incorporated multiple query, query expansion, and TREC output support
//
//
// Indri local machine query application
//
/*! \page IndriRunQuery Indri Query Retrieval
<H3>QueryEnvironment Parameters</H3>
<H4>Retrieval Parameters</H4>
<dl>
<dt>memory</dt>
<dd> an integer value specifying the number of bytes to use for the
query retrieval process. The value can include a scaling factor by
adding a suffix. Valid values are (case insensitive) K = 1000, M =
1000000, G = 1000000000. So 100M would be equivalent to 100000000. The
value should contain only decimal digits and the optional
suffix. Specified as &lt;memory&gt;100M&lt;/memory&gt; in the parameter
file and as <tt>-memory=100M</tt> on the command line. </dd>
<dt>index</dt>
<dd> path to an Indri Repository. Specified as
&lt;index&gt;/path/to/repository&lt;/index&gt; in the parameter file and
as <tt>-index=/path/to/repository</tt> on the command line. This element
```

can be specified multiple times to combine Repositories.
</dd>
<dt>server</dt>
<dd> hostname of a host running an Indri server (IndriDaemon). Specified as
<server>hostname</server> in the parameter file and
as <tt>-server=hostname</tt> on the command line. The hostname can
include an optional port number to connect to, using the form
<tt>hostname:portnum</tt>. This element
can be specified multiple times to combine servers.
</dd>
<dt>count</dt>
<dd>an integer value specifying the maximum number of results to
return for a given query. Specified as
<count>number</count> in the parameter file and
as <tt>-count=number</tt> on the command line. </dd>
<dt>query</dt>
<dd>An indri query language query to run. This element can be specified
multiple times.
</dd>
<dt>rule</dt>
<dd>specifies the smoothing rule (TermScoreFunction) to apply. Format of
the rule is:

<tt> (key ":" value) ["," key ":" value]* </tt>
<p>
Here's an example rule in command line format:

  <tt>-rule=method:linear,collectionLambda:0.2,field:title</tt>
<p> and in parameter file format:
<tt>
<rule>method:linear,collectionLambda:0.2,field:title</rule>
</tt>

<p>This corresponds to Jelinek-Mercer smoothing with background lambda
equal to 0.2, only for items in a title field.

<p>If nothing is listed for a key, all values are assumed.
So, a rule that does not specify a field matches all fields. This makes
<tt>-rule=method:linear,collectionLambda:0.2</tt> a valid rule.

<p>Valid keys:
<dl>
<dt> method</dt><dd> smoothing method (text)</dd>
<dt> field</dt><dd> field to apply this rule to</dd>
<dt> operator
<dd> type of item in query to apply to { term, window }</dd>

</dl>

<p>Valid methods:
<dl>
<dt> dirichlet</dt><dd> (also 'd', 'dir') (default mu=2500)</dd>
<dt> jelinek-mercer</dt><dd> (also 'jm', 'linear') (default
collectionLambda=0.4, documentLambda=0.0), collectionLambda is also
known as just "lambda", either will work </dt>
<dt> twostage</dt><dd> (also 'two-stage', 'two') (default mu=2500,
lambda=0.4)</dd>
</dl>
If the rule doesn't parse correctly, the default is Dirichlet, mu=2500.
</dd>
<dt>stopper</dt>
<dd>a complex element containing one or more subelements named word,
specifying the stopword list to use. Specified as
<stopper><word>stopword</word></stopper> and
as <tt>-stopper.word=stopword</tt> on the command line. This is an
optional parameter with the default of no stopping.</dd>
<dt>maxWildcardTerms</dt>
<dd>
(optional) An integer specifying the maximum number of wildcard terms that can
be generated for a synonym list for this query or set of queries. If this limit
is reached for a wildcard term, an exception will be thrown. If this parameter
is not specified, a default of 100 will be used.
</dd>
</dl>
<H4>Formatting Parameters</H4>
<dl>
<dt>queryOffset</dt>
<dd>an integer value specifying one less than the starting query number,
eg 150 for TREC formatted output. Specified as
<queryOffset>number</queryOffset> in the parameter file and
as <tt>-queryOffset=number</tt> on the command line.</dd>
<dt>runID</dt>
<dd>a string specifying the id for a query run, used in TREC scorable
output. Specified as
<runID>someID</runID> in the parameter file and
as <tt>-runID=someID</tt> on the command line.</dd>
<dt>trecFormat</dt>
<dd>the symbol <tt>true</tt> to produce TREC scorable output, otherwise
the symbol <tt>false</tt>. Specified as
<trecFormat>true</trecFormat> in the parameter file and
as <tt>-trecFormat=true</tt> on the command line. Note that <tt>0</tt>
can be used for false, and <tt>1</tt> can be used for true.</dd>
<dt>inex participant-id</dt>

<dd>triggers output of results in INEX format and specifies the participant-id attribute used in submissions.
Specified as
<inex><particpantID>someID</participantID><inex> in the parameter file and as
<tt>-inex.participantID=someID</tt> on the command line.
</dd>
<dt>inex task</dt>
<dd>triggers output of results in INEX format and specifies the task attribute (default CO.Thorough).
Specified as <inex><task>someTask</task><inex> in the parameter file and as
<tt>-inex.task=someTask</tt> on the command line.
</dd>
<dt>inex query</dt>
<dd>triggers output of results in INEX format and specifies the query attribute (default automatic).
Specified as <inex><query>someQueryType</query><inex> in the parameter file and as
<tt>-inex.query=someQueryType</tt> on the command line.
</dd>
<dt>inex topic-part</dt>
<dd>triggers output of results in INEX format and specifies the topic-part attribute (default T).
Specified as <inex><topicPart>someTopicPart</topicPart><inex> in the parameter file and as
<tt>-inex.topicPart=someTopicPart</tt> on the command line.
</dd>
<dt>inex description</dt>
<dd>triggers output of results in INEX format and specifies the contents of the description tag.
Specified as <inex><description>some description</description><inex> in the parameter file and as
<tt>-inex.description="some description"</tt> on the command line.
</dd>
</dl>
<H4>Pseudo-Relevance Feedback Parameters</H4>
<dl>
<dt>fbDocs</dt>
<dd>an integer specifying the number of documents to use for feedback. Specified as
<fbDocs>number</fbDocs> in the parameter file and as <tt>-fbDocs=number</tt> on the command line.</dd>
<dt>fbTerms</dt>
<dd>an integer specifying the number of terms to use for feedback. Specified as

<fbTerms>number</fbTerms> in the parameter file and
as <tt>-fbTerms=number</tt> on the command line.</dd>
<dt>fbMu</dt>
<dd>a floating point value specifying the value of mu to use for
feedback. Specified as
<fbMu>number</fbMu> in the parameter file and
as <tt>-fbMu=number</tt> on the command line.</dd>
<dt>fbOrigWeight</dt>
<dd>a floating point value in the range [0.0..1.0] specifying the weight
for the original query in the expanded query. Specified as
<fbOrigWeight>number</fbOrigWeight> in the parameter file and
as <tt>-fbOrigWeight=number</tt> on the command line.</dd>
</dl>
*/

```
#include <time.h>
#include "indri/QueryEnvironment.hpp"
#include "indri/LocalQueryServer.hpp"
#include "indri/delete_range.hpp"
#include "indri/NetworkStream.hpp"
#include "indri/NetworkMessageStream.hpp"
#include "indri/NetworkServerProxy.hpp"

#include "indri/ListIteratorNode.hpp"
#include "indri/ExtentInsideNode.hpp"
#include "indri/DocListIteratorNode.hpp"
#include "indri/FieldIteratorNode.hpp"

#include "indri/Parameters.hpp"

#include "indri/ParsedDocument.hpp"
#include "indri/Collection.hpp"
#include "indri/CompressedCollection.hpp"
#include "indri/TaggedDocumentIterator.hpp"
#include "indri/XMLNode.hpp"

#include "indri/QueryExpander.hpp"
#include "indri/RMExpander.hpp"
#include "indri/PonteExpander.hpp"

#include "indri/IndriTimer.hpp"
#include "indri/UtilityThread.hpp"
#include "indri/ScopedLock.hpp"
#include "indri/delete_range.hpp"
#include "indri/SnippetBuilder.hpp"
```

```
#include <queue>

static bool copy_parameters_to_string_vector( std::vector<std::string>& vec,
indri::api::Parameters p, const std::string& parameterName ) {
  if( !p.exists(parameterName) )
    return false;

  indri::api::Parameters slice = p[parameterName];

  for( size_t i=0; i<slice.size(); i++ ) {
    vec.push_back( slice[i] );
  }

  return true;
}

struct query_t {
  struct greater {
    bool operator() ( query_t* one, query_t* two ) {
      return one->index > two->index;
    }
  };

  query_t( int _index, int _number, const std::string& _text, const std::string &queryType
) :
    index( _index ),
    number( _number ),
    text( _text ), qType(queryType)
  {
  }

  query_t( int _index, int _number, const std::string& _text ) :
    index( _index ),
    number( _number ),
    text( _text )
  {
  }

  int number;
  int index;
  std::string text;
  std::string qType;
};

class QueryThread : public indri::thread::UtilityThread {
private:
```

```
indri::thread::Lockable& _queueLock;
indri::thread::ConditionVariable& _queueEvent;
std::queue< query_t* >& _queries;
std::priority_queue< query_t*, std::vector< query_t* >, query_t::greater >& _output;

indri::api::QueryEnvironment _environment;
indri::api::Parameters& _parameters;
int _requested;
int _initialRequested;

bool _printDocuments;
bool _printPassages;
bool _printSnippets;
bool _printQuery;

std::string _runID;
bool _trecFormat;
bool _inexFormat;

indri::query::QueryExpander* _expander;
std::vector<indri::api::ScoredExtentResult> _results;
indri::api::QueryAnnotation* _annotation;

// Runs the query, expanding it if necessary.  Will print output as well if verbose is on.
void _runQuery( std::stringstream& output, const std::string& query,
                const std::string &queryType ) {
  try {
    if( _printQuery ) output << "# query: " << query << std::endl;

    if( _printSnippets ) {
      _annotation = _environment.runAnnotatedQuery( query, _initialRequested );
      _results = _annotation->getResults();
    } else {
      _results = _environment.runQuery( query, _initialRequested, queryType );
    }

    if( _expander ) {
      std::string expandedQuery = _expander->expand( query, _results );
      if( _printQuery ) output << "# expanded: " << expandedQuery << std::endl;
      _results = _environment.runQuery( expandedQuery, _requested, queryType );
    }
  }
  catch( lemur::api::Exception& e )
  {
    _results.clear();
    LEMUR_RETHROW(e, "QueryThread::_runQuery Exception");
```

```
  }
 }

 void _printResultRegion( std::stringstream& output, int queryIndex, int start, int end  ) {
  std::vector<std::string> documentNames;
  std::vector<indri::api::ParsedDocument*> documents;

  std::vector<indri::api::ScoredExtentResult> resultSubset;

  resultSubset.assign( _results.begin() + start, _results.begin() + end );


  // Fetch document data for printing
  if( _printDocuments || _printPassages || _printSnippets ) {
   // Need document text, so we'll fetch the whole document
   documents = _environment.documents( resultSubset );
   documentNames.clear();

   for( size_t i=0; i<resultSubset.size(); i++ ) {
    indri::api::ParsedDocument* doc = documents[i];
    std::string documentName;

    indri::utility::greedy_vector<indri::parse::MetadataPair>::iterator iter = std::find_if(
documents[i]->metadata.begin(),
      documents[i]->metadata.end(),
      indri::parse::MetadataPair::key_equal( "docno" ) );

    if( iter != documents[i]->metadata.end() )
     documentName = (char*) iter->value;

    // store the document name in a separate vector so later code can find it
    documentNames.push_back( documentName );
   }
  } else {
   // We only want document names, so the documentMetadata call may be faster
   documentNames = _environment.documentMetadata( resultSubset, "docno" );
  }

  std::vector<std::string> pathNames;
  if ( _inexFormat ) {
   // output topic header
   output << "  <topic topic-id=\"" << queryIndex << "\">" << std::endl
          << "    <collections>" << std::endl
          << "      <collection>ieee</collection>" << std::endl
          << "    </collections>" << std::endl;
```

```
    // retrieve path names
    pathNames = _environment.pathNames( _results );
  }

  // Print results
  output << resultSubset.size() << "\n";

/*  for( size_t i=0; i < resultSubset.size(); i++ ) {
    int rank = start+i+1;
    int queryNumber = queryIndex;

    if( _trecFormat ) {
      // TREC formatted output: queryNumber, Q0, documentName, rank, score, runID
      output << queryNumber << " "
             << "Q0 "
             << documentNames[i] << " "
             << rank << " "
             << resultSubset[ i ].score << " "
             << _runID << std::endl;
    } else if( _inexFormat ) {

        output << "    <result>" << std::endl
               << "      <file>" << documentNames[i] << "</file>" << std::endl
               << "      <path>" << pathNames[i] << "</path>" << std::endl
               << "      <rsv>" << resultSubset[i].score << "</rsv>"  << std::endl
               << "    </result>" << std::endl;
    }
    else {
      // score, documentName, firstWord, lastWord
      output << resultSubset[i].score << "hELLO wORLD\t"
             << documentNames[i] << "\t"
             << resultSubset[i].begin << "\t"
             << resultSubset[i].end << std::endl;
    }

    if( _printDocuments ) {
      output << documents[i]->text << std::endl;
    }

    if( _printPassages ) {
      int byteBegin = documents[i]->positions[ resultSubset[i].begin ].begin;
      int byteEnd = documents[i]->positions[ resultSubset[i].end-1 ].end;
      output.write( documents[i]->text + byteBegin, byteEnd - byteBegin );
      output << std::endl;
    }
```

```
      if( _printSnippets ) {
        indri::api::SnippetBuilder builder(false);
        output << builder.build( resultSubset[i].document, documents[i], _annotation ) <<
std::endl;
      }

      if( documents.size() )
        delete documents[i];
    }*/
    if( _inexFormat ) {
      output << "  </topic>" << std::endl;
    }
  }

  void _printResults( std::stringstream& output, int queryIndex ) {
    for( size_t start = 0; start < _results.size(); start += 50 ) {
      size_t end = std::min<size_t>( start + 50, _results.size() );
      _printResultRegion( output, queryIndex, start, end );
    }
    delete _annotation;
    _annotation = 0;
  }


public:
  QueryThread( std::queue< query_t* >& queries,
               std::priority_queue< query_t*, std::vector< query_t* >, query_t::greater >&
output,
               indri::thread::Lockable& queueLock,
               indri::thread::ConditionVariable& queueEvent,
               indri::api::Parameters& params ) :
    _queries(queries),
    _output(output),
    _queueLock(queueLock),
    _queueEvent(queueEvent),
    _parameters(params),
    _expander(0),
    _annotation(0)
  {
  }

  ~QueryThread() {
  }

  UINT64 initialize() {
    _environment.setMemory( _parameters.get("memory", 100*1024*1024) );
```

```
    _environment.setSingleBackgroundModel(
_parameters.get("singleBackgroundModel", false) );

    std::vector<std::string> stopwords;
    if( copy_parameters_to_string_vector( stopwords, _parameters, "stopper.word" ) )
      _environment.setStopwords(stopwords);

    std::vector<std::string> smoothingRules;
    if( copy_parameters_to_string_vector( smoothingRules, _parameters, "rule" ) )
      _environment.setScoringRules( smoothingRules );

   if( _parameters.exists( "index" ) ) {
      indri::api::Parameters indexes = _parameters["index"];

      for( size_t i=0; i < indexes.size(); i++ ) {
        _environment.addIndex( std::string(indexes[i]) );
      }
    }

    if( _parameters.exists( "server" ) ) {
      indri::api::Parameters servers = _parameters["server"];

      for( size_t i=0; i < servers.size(); i++ ) {
        _environment.addServer( std::string(servers[i]) );
      }
    }

    if( _parameters.exists("maxWildcardTerms") )
       _environment.setMaxWildcardTerms(_parameters.get("maxWildcardTerms", 100));

    _requested = _parameters.get( "count", 1000 );
    _initialRequested = _parameters.get( "fbDocs", _requested );
    _runID = _parameters.get( "runID", "indri" );
    _trecFormat = _parameters.get( "trecFormat" , false );
    _inexFormat = _parameters.exists( "inex" );

    _printQuery = _parameters.get( "printQuery", false );
    _printDocuments = _parameters.get( "printDocuments", false );
    _printPassages = _parameters.get( "printPassages", false );
    _printSnippets = _parameters.get( "printSnippets", false );

    if( _parameters.get( "fbDocs", 0 ) != 0 ) {
      _expander = new indri::query::RMExpander( &_environment, _parameters );
    }

                    if (_parameters.exists("maxWildcardTerms")) {
```

```
        _environment.setMaxWildcardTerms((int)_parameters.get("maxWildcardTerms")
);
                }

    return 0;
  }

  void deinitialize() {
    delete _expander;
    _environment.close();
  }

  bool hasWork() {
    indri::thread::ScopedLock sl( &_queueLock );
    return _queries.size() > 0;
  }

  UINT64 work() {
    query_t* query;
    std::stringstream output;

    // pop a query off the queue
    {
      indri::thread::ScopedLock sl( &_queueLock );
      if( _queries.size() ) {
        query = _queries.front();
        _queries.pop();
      } else {
        return 0;
      }
    }

    // run the query
    try {
      _runQuery( output, query->text, query->qType );
    } catch( lemur::api::Exception& e ) {
      output << "# EXCEPTION in query " << query->number << ": " << e.what() <<
std::endl;
    }

    // print the results to the output stream
    _printResults( output, query->number );

    // push that data into an output queue...?
    {
```

```
      indri::thread::ScopedLock sl( &_queueLock );
      _output.push( new query_t( query->index, query->number, output.str() ) );
      _queueEvent.notifyAll();
    }

    delete query;
    return 0;
  }
};

void push_queue( std::queue< query_t* >& q, indri::api::Parameters& queries,
                 int queryOffset ) {

  for( size_t i=0; i<queries.size(); i++ ) {
    int queryNumber;
    std::string queryText;
    std::string queryType = "indri";
    if( queries[i].exists( "type" ) )
      queryType = (std::string) queries[i]["type"];

    if( queries[i].exists( "number" ) ) {
      queryText = (std::string) queries[i]["text"];
      queryNumber = (int) queries[i]["number"];
    } else {
      queryText = (std::string) queries[i];
      queryNumber = queryOffset + int(i);
    }
    q.push( new query_t( i, queryNumber, queryText, queryType ) );
  }
}

int main(int argc, char * argv[]) {
  try {
    indri::api::Parameters& param = indri::api::Parameters::instance();
    param.loadCommandLine( argc, argv );

    if( param.get( "version", 0 ) ) {
      std::cout << INDRI_DISTRIBUTION << std::endl;
    }

    if( !param.exists( "query" ) )
      LEMUR_THROW( LEMUR_MISSING_PARAMETER_ERROR, "Must specify at
least one query." );

    if( !param.exists("index") && !param.exists("server") )
```

```
    LEMUR_THROW( LEMUR_MISSING_PARAMETER_ERROR, "Must specify a
server or index to query against." );

   int threadCount = param.get( "threads", 1 );
   std::queue< query_t* > queries;
   std::priority_queue< query_t*, std::vector< query_t* >, query_t::greater > output;
   std::vector< QueryThread* > threads;
   indri::thread::Mutex queueLock;
   indri::thread::ConditionVariable queueEvent;

   // push all queries onto a queue
   indri::api::Parameters parameterQueries = param[ "query" ];
   int queryOffset = param.get( "queryOffset", 0 );
   push_queue( queries, parameterQueries, queryOffset );
   int queryCount = (int)queries.size();

   // launch threads
   for( int i=0; i<threadCount; i++ ) {
     threads.push_back( new QueryThread( queries, output, queueLock, queueEvent,
param ) );
     threads.back()->start();
   }

   int query = 0;

   bool inexFormat = param.exists( "inex" );
   if( inexFormat ) {
     std::string participantID = param.get( "inex.participantID", "1");
     std::string runID = param.get( "runID", "indri" );
     std::string inexTask = param.get( "inex.task", "CO.Thorough" );
     std::string inexTopicPart = param.get( "inex.topicPart", "T" );
     std::string description = param.get( "inex.description", "" );
     std::string queryType = param.get("inex.query", "automatic");
     std::cout << "<inex-submission participant-id=\"" << participantID
                  << "\" run-id=\"" << runID
                  << "\" task=\"" << inexTask
                  << "\" query=\"" << queryType
                  << "\" topic-part=\"" << inexTopicPart
                  << "\">" << std::endl
                  << "  <description>" << std::endl << description
                  << std::endl << "  </description>" << std::endl;
   }

   // acquire the lock.
   queueLock.lock();
```

```
  // process output as it appears on the queue
  while( query < queryCount ) {
    query_t* result = NULL;

    // wait for something to happen
    queueEvent.wait( queueLock );

    while( output.size() && output.top()->index == query ) {
      result = output.top();
      output.pop();

      queueLock.unlock();

      std::cout << result->text;
      delete result;
      query++;

      queueLock.lock();
    }
  }
  queueLock.unlock();

  if( inexFormat ) {
    std::cout << "</inex-submission>" << std::endl;
  }

  // join all the threads
  for( size_t i=0; i<threads.size(); i++ )
    threads[i]->join();

  // we've seen all the query output now, so we can quit
  indri::utility::delete_vector_contents( threads );
 } catch( lemur::api::Exception& e ) {
  LEMUR_ABORT(e);
 } catch( ... ) {
  std::cout << "Caught unhandled exception" << std::endl;
  return -1;
 }

 return 0;
}
```

APPENDIX H: LUCENE – INDEX.JAVA

Index.java was copied from the delivered IndexFiles.java file. It was modified to accept two input parameters: the location of the object collection to be indexed and the location of the directory wherein to build the index. All commands to print to the screen were commented out. The resulting program created a new instance of an IndexWriter, created an index for an object collection, and terminated. By design, Lucene only indexes the first five kB of data in each file. This limit was removed so that the entire file would be indexed. Index.java was compiled and added to the delivered lucene-demos-2.1.0.jar file.

```
package org.apache.lucene.demo;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.index.IndexWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Date;

/** Index all text files under a directory. */
public class index {

  private index() {}

  /** Index all text files under a directory. */
  public static void main(String[] args) {
    final File docDir = new File(args[0]);
    final File INDEX_DIR = new File(args[1]);

    try {
      IndexWriter writer = new IndexWriter(INDEX_DIR, new StandardAnalyzer(), true);
      indexDocs(writer, docDir);
//      System.out.println("Optimizing...");
//      writer.optimize();
      writer.close();
    } catch (IOException e) {
      System.out.println(" caught a " + e.getClass() +
       "\n with message: " + e.getMessage());
    }
  }

  static void indexDocs(IndexWriter writer, File file)
    throws IOException {
```

```
    if (file.canRead()) {
      if (file.isDirectory()) {
        String[] files = file.list();
        if (files != null) {
          for (int i = 0; i < files.length; i++) {
            indexDocs(writer, new File(file, files[i]));
          }
        }
      } else {
//        System.out.println("adding " + file);
        try {
          writer.addDocument(FileDocument.Document(file));
        }
        catch (FileNotFoundException fnfe) {
          ;
        }
      }
    }
  }
}
```

APPENDIX I: LUCENE – SEARCH.JAVA

Search.java was copied from the delivered SearchFiles.java file. It was modified to accept two input parameters: the location of the index and the query parameters. With the exception of two, all commands to print to the screen were commented out. The program still prints the query parameters and the number of matching documents found. Code designed to format the result set was also commented out. The resulting program created a new instance of an IndexReader, created a query, printed the query parameters, executed the query, printed the number of matching documents, and terminated. Search.java was compiled and added to the delivered lucene-demos-2.1.0.jar file.

```
package org.apache.lucene.demo;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.FilterIndexReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.Searcher;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;

/** Simple command-line based search demo. */
public class search {

  /** Use the norms from one field for all fields.  Norms are read into memory,
   * using a byte of memory per document per searched field.  This can cause
   * search of large collections with a large number of fields to run out of
   * memory.  If all of the fields contain only a single token, then the norms
   * are all identical, then single norm vector may be shared. */
  private static class OneNormsReader extends FilterIndexReader {
    private String field;

    public OneNormsReader(IndexReader in, String field) {
      super(in);
      this.field = field;
```

```
    }

    public byte[] norms(String field) throws IOException {
      return in.norms(this.field);
    }
  }

  private search() {}

  /** Simple command-line based search demo. */
  public static void main(String[] args) throws Exception {
    String INDEX_DIR = args[0];
    String SEARCH_STR = args[1];
    String field = "contents";
//    String queries = null;
//    boolean raw = false;
    String normsField = null;

    IndexReader reader = IndexReader.open(INDEX_DIR);

    if (normsField != null)
      reader = new OneNormsReader(reader, normsField);

    Searcher searcher = new IndexSearcher(reader);
    Analyzer analyzer = new StandardAnalyzer();

//    BufferedReader in = null;
//   if (queries != null) {
//      in = new BufferedReader(new FileReader(queries));
//   } else {
//      in = new BufferedReader(new InputStreamReader(System.in, "UTF-8"));
//    }
      QueryParser parser = new QueryParser(field, analyzer);

      Query query = parser.parse(SEARCH_STR);
      System.out.println("Searching for: " + query.toString(field));

      Hits hits = searcher.search(query);

//      hits = searcher.search(query);

      System.out.println(hits.length() + " total matching documents");

/*      final int HITS_PER_PAGE = 10;
      for (int start = 0; start < hits.length(); start += HITS_PER_PAGE) {
        int end = Math.min(hits.length(), start + HITS_PER_PAGE);
```

```
      for (int i = start; i < end; i++) {

        if (raw) {                            // output raw format
          System.out.println("doc="+hits.id(i)+" score="+hits.score(i));
          continue;
        }

        Document doc = hits.doc(i);
        String path = doc.get("path");
        if (path != null) {
          System.out.println((i+1) + ". " + path);
          String title = doc.get("title");
          if (title != null) {
            System.out.println("   Title: " + doc.get("title"));
          }
        } else {
          System.out.println((i+1) + ". " + "No path for this document");
        }
      }

      if (queries != null)                    // non-interactive
        break;

      if (hits.length() > end) {
        System.out.print("more (y/n) ? ");
        line = in.readLine();
        if (line.length() == 0 || line.charAt(0) == 'n')
          break;
      }
    }
*/
    reader.close();
  }
}
```

APPENDIX J: RUN CONTROL

The Run Control is a PERL script designed to uniformly and accurately administer all of the aspects of the tests. It is written in PERL to minimize resource consumption during the execution of tests. The Run Control is responsible for system initialization, recording all of performance characteristics, and deleting temporary files created during execution. For more detail, see section 3.1.5.

```
#!/usr/bin/perl
use Time::HiRes qw(gettimeofday tv_interval);

my $path_to_docs = $ARGV[0];
my $path_to_indx = $ARGV[1];
my $search_str   = $ARGV[2];
my $results      = $ARGV[3];
my $collection   = "/home/data/temporary_collection/";

#LUCENE
#print "*******************************Warming up.\n";
for ($int=0; $int<0; $int++) {
  print `rm /home/index/*`;
  print `rmdir /home/index/`;
  $start = [gettimeofday];
  print `/home/engines/jdk1.6.0_01/bin/java -classpath /home/engines/lucene-
2.1.0/lucene-demos-2.1.0.jar:/home/engines/lucene-2.1.0/lucene-core-2.1.0.jar
org.apache.lucene.demo.index /home/scripts/trash /home/index/`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Lucene\twarmup\t$int\t$total" >> $results`;
}

for ($int=0; $int<0; $int++) {
  print "*******************************Lucene $int\n";
  print `rm $path_to_indx*`;
  print `rm $collection*`;
  print `rmdir $path_to_indx`;
  print `cp $path_to_docs* $collection`;

  $start = [gettimeofday];
  print `/home/engines/jdk1.6.0_01/bin/java -classpath /home/engines/lucene-
2.1.0/lucene-demos-2.1.0.jar:/home/engines/lucene-2.1.0/lucene-core-2.1.0.jar
org.apache.lucene.demo.index $collection $path_to_indx`;
```

```
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Lucene\tindex1\t$int\t$total" >> $results`;

  $start = [gettimeofday];
  print `/home/engines/jdk1.6.0_01/bin/java -classpath /home/engines/lucene-
2.1.0/build/classes/:/home/engines/lucene-2.1.0/lucene-core-
2.1.0.jar:/home/engines/lucene-2.1.0/lucene-demos-2.1.0.jar
org.apache.lucene.demo.search $path_to_indx $search_str >> $results`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Lucene\tsearch1\t$int\t$total" >> $results`;

#  print `cp /home/data/100Collection/* $collection`;

#  $start = [gettimeofday];
#  print `/home/engines/jdk1.6.0_01/bin/java -classpath /home/engines/lucene-
2.1.0/lucene-demos-2.1.0.jar:/home/engines/lucene-2.1.0/lucene-core-2.1.0.jar
org.apache.lucene.demo.index $collection $path_to_indx`;
#  $end = [gettimeofday];
#  $total = tv_interval $start, $end;
#  print `echo "Lucene\tupdate1\t$int\t$total" >> $results`;

#  $start = [gettimeofday];
#  print `/home/engines/jdk1.6.0_01/bin/java -classpath /home/engines/lucene-
2.1.0/build/classes/:/home/engines/lucene-2.1.0/lucene-core-
2.1.0.jar:/home/engines/lucene-2.1.0/lucene-demos-2.1.0.jar
org.apache.lucene.demo.search $path_to_indx $search_str >> $results`;
#  $end = [gettimeofday];
#  $total = tv_interval $start, $end;
#  print `echo "Lucene\tsearch2\t$int\t$total" >> $results`;
}

#CLUCENE
for ($int=0; $int<0; $int++) {
  print "*****************************CLucene $int\n";
  print `rm $path_to_indx*`;
  print `rmdir $path_to_indx`;
  print `rm $collection*`;
  print `cp $path_to_docs* $collection`;

#  $start = [gettimeofday];
#  print `/home/engines/clucene-core-0.9.16a/src/demo/index -i $path_to_docs
$path_to_indx`;
#  $end = [gettimeofday];
#  $total = tv_interval $start, $end;
```

```
#  print `echo "CLucene\tindex1\t$int\t$total" >> $results`;

#  $start = [gettimeofday];
#        print    `/home/engines/clucene-core-0.9.16a/src/demo/index    -s    $path_to_docs
$path_to_indx $search_str`;
#  $end = [gettimeofday];
#  $total = tv_interval $start, $end;
#  print `echo "CLucene\tsearch1\t$int\t$total" >> $results`;

  print `cp /home/data/100Collection/* $collection`;

  $start = [gettimeofday];
  print          `/home/engines/clucene-core-0.9.16a/src/demo/index          -i          $collection
$path_to_indx`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "CLucene\tupdate1\t$int\t$total" >> $results`;

  $start = [gettimeofday];
  print `/home/engines/clucene-core-0.9.16a/src/demo/index -s $collection $path_to_indx
$search_str`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "CLucene\tsearch2\t$int\t$total" >> $results`;
}

#Glimpse
for ($int=0; $int<0; $int++) {
  print "*****************************Glimpse $int\n";
  print `rm /root/.glimp*`;
  print `rm $collection*`;
  print `cp $path_to_docs* $collection`;

  $start = [gettimeofday];
  print `glimpseindex $path_to_docs`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Glimpse\tindex1\t$int\t$total" >> $results`;

  $start = [gettimeofday];
  print `glimpse $search_str`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "GLimpse\tsearch1\t$int\t$total" >> $results`;

  print `rm /root/.glimps*`;
```

```
  print `cp /home/data/100Collection/* $collection`;

  $start = [gettimeofday];
  print `glimpseindex $collection`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Glimpse\tupdate1\t$int\t$total" >> $results`;

  $start = [gettimeofday];
  print `glimpse $search_str`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "GLimpse\tsearch2\t$int\t$total" >> $results`;
 }

#Zettair
for ($int=0; $int<10; $int++) {
#  print "*******************************Zettair $path_to_docs $int\n";
  print `rm index.*`;
  print `rm $collection*`;
  print `cp $path_to_docs* $collection`;

  $start = [gettimeofday];
  print `find $collection* | ../engines/zettair-0.9.3/zet -i -t HTML >> $results`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Zettair\tindex1\t$int\t$total" >> $results`;

  $start = [gettimeofday];
  print `/home/engines/zettair-0.9.3/zet hat >> $results`;
  $end = [gettimeofday];
  $total = tv_interval $start, $end;
  print `echo "Zettair\tsearch1\t$int\t$total" >> $results`;

#  if ($path_to_docs != "/home/data/100Collection/"){
    print `rm index.*`;
    print `cp /home/data/100Collection/* $collection`;

    $start = [gettimeofday];
    print `find $collection* | ../engines/zettair-0.9.3/zet -i -t HTML >> $results`;
    $end = [gettimeofday];
    $total = tv_interval $start, $end;
    print `echo "Zettair\tupdate1\t$int\t$total" >> $results`;

    $start = [gettimeofday];
    print `/home/engines/zettair-0.9.3/zet hat >> $results`;
```

```
    $end = [gettimeofday];
    $total = tv_interval $start, $end;
    print `echo "Zettair\tsearch2\t$int\t$total" >> $results`;
#   }
}
```

APPENDIX K: TEST DOCUMENT

This document is a test document intentionally designed with problematic language patterns. It was used during the execution of tests designed to measure retrieval accuracy.

The amount of information available on both public and private networks continues to grow at a phenomenal rate. An equation that estimates this growth can be expressed as follows:

*I = Y * 100,000,000*
I = amount of information tomorrow
Y = amount of information yesterday

This information is contained within a wide variety of objects, including documents, e-mail archives, medical records, manuals, pictures and music. To be of any value, it must be easily searchable and accessible. Information Retrieval (IR) is concerned with the ability to find and gain access to relevant information.

As electronic data repositories continue to proliferate, so too, grows the variety of methods used to locate and access the information contained therein. The growing variety of retrieval strategies introduces a need for an infrastructure capable of measuring and comparing the performance of competing solutions, but such an environment does not yet exist. The purpose of this research is to develop an infrastructure wherein IR solutions may be evaluated and compared. In 1979, Mr. van Rijsbergen, an expert in the IR field believed the need for a system-independent benchmarking utility was long overdue—twenty-five years later, progress in this area has been minimal. Contrastingly, new theories have emerged; new techniques have been introduced; all with the goal of improving retrieval performance. The need for a system-independent analysis of retrieval performance is more critical now as a tool to aid IT professionals in the development and implementation of IR solutions.

Appendix L: Test Results

These tables contain results for each IR solution evaluated in this research.

CLUCENE:

100Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	23.633358	1.099732	52	29	N/A	N/A	N/A	N/A
2	27.741108	1.320965	52	29	N/A	N/A	N/A	N/A
3	18.767055	0.919287	52	29	N/A	N/A	N/A	N/A
4	21.892728	0.964395	52	29	N/A	N/A	N/A	N/A
5	19.191439	1.916113	52	29	N/A	N/A	N/A	N/A
6	20.114832	0.793689	52	29	N/A	N/A	N/A	N/A
7	26.980446	0.78488	52	29	N/A	N/A	N/A	N/A
8	19.069258	0.787515	52	29	N/A	N/A	N/A	N/A
9	23.086808	0.83344	52	29	N/A	N/A	N/A	N/A
10	20.817973	1.993866	52	29	N/A	N/A	N/A	N/A

200Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	126.604055	0.818932	90	99.8	136.043325	0.614748	142	128.6
2	145.829117	0.779741	90	99.8	130.091886	0.765646	142	128.6
3	116.133245	0.74707	90	99.8	123.35552	0.892206	142	128.6
4	126.022225	0.894713	90	99.8	125.694087	0.760009	142	128.6
5	129.65238	0.668788	90	99.8	126.416492	0.711304	142	128.6
6	133.786238	1.394394	90	99.8	126.379734	0.743848	142	128.6
7	136.164018	0.763189	90	99.8	123.581032	0.721114	142	128.6
8	124.8958	0.71508	90	99.8	128.152461	0.782538	142	128.6
9	132.687575	0.884992	90	99.8	131.221277	0.720767	142	128.6
10	144.851026	0.950921	90	99.8	121.976471	0.721387	142	128.6

300Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	145.42087	1.22877	108	134.4	167.211123	0.861586	160	163.1
2	145.214122	1.253272	108	134.4	154.045141	0.64913	160	163.1
3	146.093328	0.832299	108	134.4	151.55091	1.445197	160	163.1
4	147.865529	0.946505	108	134.4	153.283582	1.14727	160	163.1
5	147.963123	0.799978	108	134.4	155.894698	0.734135	160	163.1
6	137.593166	0.854266	108	134.4	152.270658	0.738807	160	163.1
7	139.775393	0.987774	108	134.4	152.140367	0.567109	160	163.1
8	148.08283	0.73198	108	134.4	156.974101	1.20641	160	163.1
9	138.346682	0.998504	108	134.4	166.866102	0.68441	160	163.1
10	141.55173	0.842905	108	134.4	149.757155	0.775376	160	163.1

CLucene continued…

400Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	182.004001	0.93608	164	168.8	191.480935	1.372344	216	197.5
2	185.093666	1.427547	164	168.8	187.679842	0.837593	216	197.5
3	190.737096	2.09908	164	168.8	187.481988	0.828528	216	197.5
4	187.654926	1.795683	164	168.8	187.225324	0.769837	216	197.5
5	186.351325	0.974098	164	168.8	188.365835	0.791934	216	197.5
6	183.527986	0.992368	164	168.8	185.545546	1.030287	216	197.5
7	182.467406	0.769708	164	168.8	186.673091	0.960635	216	197.5
8	186.263847	1.34698	164	168.8	187.629379	0.824452	216	197.5
9	184.100384	0.930957	164	168.8	188.74734	0.821243	216	197.5
10	183.315555	0.874328	164	168.8	190.196475	1.021422	216	197.5

500Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	19.164769	1.61537		**16.9**			**30**	20
2	24.288331	3.235432		**16.9**			**30**	
3	27.124503	1.114046		**16.9**			**30**	
4	25.83521	1.43778		**16.9**			**30**	
5	27.720011	2.438962		**16.9**			**30**	
6	23.419593	1.132876		**16.9**			**30**	
7	25.972134	1.401461		**16.9**			**30**	
8	23.8836	2.673917		**16.9**			**30**	
9	29.492446	2.030015		**16.9**			**30**	
10	25.904464	2.026272		**16.9**			**30**	

600Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	309.079646	0.799605	250	254	275.419322	0.886991	302	282.7
2	316.264963	0.829948	250	254	266.803877	0.720691	302	282.7
3	310.471277	0.804233	250	254	274.753051	1.258793	302	282.7
4	297.292194	0.841164	250	254	267.160627	0.78191	302	282.7
5	307.260456	0.919044	250	254	268.446945	0.937231	302	282.7
6	308.647189	0.937716	250	254	273.367939	0.657302	302	282.7
7	300.201423	0.81919	250	254	267.907495	0.822157	302	282.7
8	308.622642	1.069225	250	254	273.628302	0.757929	302	282.7
9	301.457967	0.758035	250	254	267.88095	0.825181	302	282.7
10	301.961197	1.08898	250	254	273.512746	0.623585	302	282.7

CLucene continued…

700Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	290.198001	0.796288	280	291.4	314.021285	1.2502	332	320.1
2	286.37343	0.89539	280	291.4	315.488513	0.786077	332	320.1
3	279.826598	0.841911	280	291.4	316.487015	0.83864	332	320.1
4	285.583572	0.734852	280	291.4	312.016088	0.768465	332	320.1
5	281.59406	0.739561	280	291.4	311.05039	0.96966	332	320.1
6	282.520592	0.760555	280	291.4	315.538451	0.975115	332	320.1
7	280.748539	0.728912	280	291.4	316.667408	0.840351	332	320.1
8	287.895209	0.71816	280	291.4	312.925425	0.794346	332	320.1
9	286.335653	0.917732	280	291.4	311.493561	0.816357	332	320.1
10	279.142712	0.78444	280	291.4	315.280058	0.787629	332	320.1

800Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	319.667353	0.696807	294	321.2	342.417022	0.790321	346	349.9
2	320.389424	0.660611	294	321.2	363.237458	0.694205	346	349.9
3	318.014557	0.73821	294	321.2	357.270759	0.815079	346	349.9
4	316.580293	0.860451	294	321.2	359.491067	0.938104	346	349.9
5	321.656091	0.773281	294	321.2	356.814764	0.790939	346	349.9
6	319.176303	0.727608	294	321.2	351.49075	0.85774	346	349.9
7	315.97343	0.77046	294	321.2	351.309804	1.003135	346	349.9
8	318.815447	0.637956	294	321.2	349.952795	0.823136	346	349.9
9	319.506064	0.605078	294	321.2	353.511096	0.810026	346	349.9
10	317.080211	1.061308	294	321.2	350.65888	0.936345	346	349.9

900Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	341.757646	0.970128	339	344.9	374.35822	0.730253	391	373.6
2	341.758428	0.945987	339	344.9	373.549508	1.684924	391	373.6
3	337.163913	0.749727	339	344.9	371.326689	0.830801	391	373.6
4	341.582598	0.714044	339	344.9	368.376412	0.71902	391	373.6
5	339.776731	0.740526	339	344.9	371.081601	0.756901	391	373.6
6	348.087992	0.761475	339	344.9	373.318848	0.855878	391	373.6
7	339.067603	0.795807	339	344.9	364.583401	1.460194	391	373.6
8	344.368914	0.93441	339	344.9	371.335774	1.183453	391	373.6
9	343.420392	0.852642	339	344.9	375.578259	0.723439	391	373.6
10	343.206653	0.963987	339	344.9	368.51535	1.175224	391	373.6

CLucene continued…

1000Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	387.93814		381	393.9	522.473272	0.675199	433	422.6
2	381.526639		381	393.9	497.607195	0.702261	433	422.6
3	382.337046		381	393.9	491.215578	0.744973	433	422.6
4	385.022199		381	393.9	493.339273	0.798069	433	422.6
5	381.22793		381	393.9	493.924447	0.748537	433	422.6
6	380.171047		381	393.9	496.857305	0.673057	433	422.6
7	602.269435		381	393.9	493.741822	0.70163	433	422.6
8	662.43196		381	393.9	502.123944	0.770584	433	422.6
9	660.942194		381	393.9	495.306273	0.760464	433	422.6
10	675.017111		381	393.9	507.824396	0.803553	433	422.6

GLIMPSE:

100Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	15.974806	0.186693	100	1.7	N/A	N/A	N/A	N/A
1	12.600891	0.151974	100	1.7	N/A	N/A	N/A	N/A
2	14.077058	0.325787	100	1.7	N/A	N/A	N/A	N/A
3	12.155966	0.15033	100	1.7	N/A	N/A	N/A	N/A
4	15.765864	0.094892	100	1.7	N/A	N/A	N/A	N/A
5	12.329419	0.130192	100	1.7	N/A	N/A	N/A	N/A
6	15.472493	0.370968	100	1.7	N/A	N/A	N/A	N/A
7	11.941812	0.137862	100	1.7	N/A	N/A	N/A	N/A
8	14.873494	0.121489	100	1.7	N/A	N/A	N/A	N/A
9	12.572138	0.130213	100	1.7	N/A	N/A	N/A	N/A

200Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	57.559552	0.515038	200	4.7	72.19973	0.610983	300	5.5MB
1	50.702541	0.169193	200	4.7	73.391058	0.519811	300	5.5MB
2	52.491229	0.469394	200	4.7	86.51535	0.470785	300	5.5MB
3	57.377323	0.497547	200	4.7	76.216408	0.508846	300	5.5MB
4	54.850193	0.487223	200	4.7	76.007137	0.175897	300	5.5MB
5	54.178863	0.75391	200	4.7	85.48264	0.500418	300	5.5MB
6	53.819794	0.205276	200	4.7	75.606714	0.471528	300	5.5MB
7	52.138736	0.434275	200	4.7	74.864428	0.525101	300	5.5MB
8	51.798442	0.431224	200	4.7	76.864336	0.500019	300	5.5MB
9	56.280125	0.414279	200	4.7	75.509963	0.503699	300	5.5MB

Glimpse continued…

300Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	70.453121	0.21433	300	5.4	100.55778	0.467562	400	6.3MB
1	70.008965	0.363884	300	5.4	93.472526	0.360621	400	6.3MB
2	67.361294	0.486559	300	5.4	88.76157	0.501578	400	6.3MB
3	70.392406	0.471921	300	5.4	96.759192	0.470675	400	6.3MB
4	71.931141	0.158618	300	5.4	91.707926	0.228348	400	6.3MB
5	66.206932	0.174955	300	5.4	90.431807	0.184225	400	6.3MB
6	72.930785	0.401682	300	5.4	88.973091	0.190343	400	6.3MB
7	66.286121	0.159716	300	5.4	90.01982	0.1568	400	6.3MB
8	67.261265	0.496209	300	5.4	91.885177	0.486701	400	6.3MB
9	68.815681	0.436248	300	5.4	87.69811	0.541253	400	6.3MB

400Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	84.679678	0.192223	400	5.9	118.130369	0.557909	500	6.6MB
1	83.900118	0.408139	400	5.9	104.411065	0.504904	500	6.6MB
2	86.045686	0.449312	400	5.9	119.162785	0.529198	500	6.6MB
3	83.429633	0.533469	400	5.9	102.59408	0.151734	500	6.6MB
4	80.342448	0.23908	400	5.9	102.208812	0.51795	500	6.6MB
5	80.449732	0.389306	400	5.9	105.707985	0.159115	500	6.6MB
6	83.133304	0.409746	400	5.9	102.33572	0.160473	500	6.6MB
7	82.000812	0.211398	400	5.9	103.852103	0.174309	500	6.6MB
8	87.698474	0.159911	400	5.9	115.915563	0.172454	500	6.6MB
9	82.888915	0.17825	400	5.9	101.924496	0.221989	500	6.6MB

500Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	113.146969	0.169554	498	5.6	133.497689	0.181042	598	7.5MB
1	108.793678	0.183074	498	5.6	133.774207	0.175718	598	7.5MB
2	111.6073	2.101069	498	5.6	138.317149	0.220069	598	7.5MB
3	110.195172	0.159749	498	5.6	138.077041	0.18358	598	7.5MB
4	108.257376	0.185024	498	5.6	140.437973	0.196584	598	7.5MB
5	109.337886	0.172373	498	5.6	139.706761	0.189225	598	7.5MB
6	109.880036	0.167618	498	5.6	138.557217	0.229805	598	7.5MB
7	107.671103	0.173682	498	5.6	136.694785	0.248185	598	7.5MB
8	120.69001	0.20042	498	5.6	151.637131	0.185325	598	7.5MB
9	113.166508	0.15208	498	5.6	140.440953	0.168249	598	7.5MB

Glimpse continued…

600Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	156.033069	0.232074	599	8.4	179.890648	0.164016	699	9.0MB
1	148.951166	0.296158	599	8.4	176.630499	0.185338	699	9.0MB
2	152.637348	0.181099	599	8.4	177.763055	0.195047	699	9.0MB
3	158.780309	0.654734	599	8.4	180.655491	0.175936	699	9.0MB
4	150.903235	0.185122	599	8.4	179.611506	0.239216	699	9.0MB
5	152.691656	0.196048	599	8.4	181.620244	0.197004	699	9.0MB
6	169.281702	0.190195	599	8.4	203.389315	0.674063	699	9.0MB
7	151.239505	0.228946	599	8.4	179.571276	0.191796	699	9.0MB
8	152.809591	0.195094	599	8.4	179.877911	0.192882	699	9.0MB
9	156.957661	0.179902	599	8.4	181.542081	0.191082	699	9.0MB

700Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	215.188592	0.205429	698	9.6	233.609922	0.185618	798	10.2MB
1	197.770132	0.178408	698	9.6	219.385328	0.183827	798	10.2MB
2	191.263727	0.191306	698	9.6	224.242313	0.182231	798	10.2MB
3	198.268563	0.184231	698	9.6	219.376079	0.25459	798	10.2MB
4	189.788673	0.200119	698	9.6	231.036683	0.182835	798	10.2MB
5	270.580664	0.188855	698	9.6	228.252638	0.18151	798	10.2MB
6	192.708248	0.16859	698	9.6	219.116496	0.255715	798	10.2MB
7	193.002104	0.169791	698	9.6	222.65211	0.224227	798	10.2MB
8	194.28487	0.171469	698	9.6	223.120235	0.421946	798	10.2MB
9	190.574578	0.194108	698	9.6	217.860946	0.172001	798	10.2MB

800Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	249.997322	0.178164	800	10.8	263.214393	0.196218	900	11.5MB
1	246.171174	0.193339	800	10.8	287.929097	0.523729	900	11.5MB
2	257.814649	0.596365	800	10.8	310.420613	1.131442	900	11.5MB
3	248.797835	0.19308	800	10.8	264.432705	0.200026	900	11.5MB
4	249.220032	0.205456	800	10.8	282.243071	0.206058	900	11.5MB
5	258.12381	0.228886	800	10.8	286.465605	0.239568	900	11.5MB
6	263.928352	0.339509	800	10.8	319.50456	0.51172	900	11.5MB
7	242.214378	0.44574	800	10.8	263.817975	0.193553	900	11.5MB
8	250.380458	0.233104	800	10.8	281.760213	0.209945	900	11.5MB
9	244.203876	0.23328	800	10.8	255.98932	0.258323	900	11.5MB

Glimpse continued…

900Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	245.928868	0.418711	898	10.3	263.783078	0.392114	998	10.9MB
1	244.378623	0.230048	898	10.3	276.134143	0.267376	998	10.9MB
2	238.85025	0.547882	898	10.3	270.670667	0.89464	998	10.9MB
3	235.426733	0.250389	898	10.3	250.200981	0.239449	998	10.9MB
4	234.97772	0.191278	898	10.3	278.222001	0.206458	998	10.9MB
5	234.921415	0.553908	898	10.3	269.803153	0.223733	998	10.9MB
6	234.162032	0.82963	898	10.3	272.436461	0.273245	998	10.9MB
7	234.703454	0.2163	898	10.3	271.028591	0.218774	998	10.9MB
8	237.400898	0.694744	898	10.3	280.825187	0.194648	998	10.9MB
9	238.560686	0.249908	898	10.3	275.223118	0.280136	998	10.9MB

1000Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
0	274.078501	0.435951	988	11.2	295.432186	0.273328	1088	11.8MB
1	259.976803	1.046302	988	11.2	291.314458	0.253225	1088	11.8MB
2	259.817979	0.439211	988	11.2	292.650973	0.239525	1088	11.8MB
3	257.604902	0.182339	988	11.2	295.699844	0.210908	1088	11.8MB
4	259.73548	0.234968	988	11.2	294.798268	0.201336	1088	11.8MB
5	253.621844	0.205099	988	11.2	288.03901	0.265836	1088	11.8MB
6	253.816419	0.17115	988	11.2	287.167894	0.196163	1088	11.8MB
7	260.405952	0.202849	988	11.2	289.953673	0.208639	1088	11.8MB
8	256.784544	0.201035	988	11.2	285.544854	0.255549	1088	11.8MB
9	272.046573	0.40431	988	11.2	319.032114	0.223226	1088	11.8MB

LEMUR:

100Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	60.233002	2.406265	50		N/A	N/A	N/A	N/A
2	45.203034	1.38839	50		N/A	N/A	N/A	N/A
3	47.246937	1.127022	50		N/A	N/A	N/A	N/A
4	48.996961	3.482673	50		N/A	N/A	N/A	N/A
5	57.517361	1.104716	50		N/A	N/A	N/A	N/A
6	59.437543	1.285693	50		N/A	N/A	N/A	N/A
7	61.282913	1.143174	50		N/A	N/A	N/A	N/A
8	69.41472	1.071994	50		N/A	N/A	N/A	N/A
9	80.018652	1.025834	50		N/A	N/A	N/A	N/A
10	90.991357	1.000549	50		N/A	N/A	N/A	N/A

Lemur continued…

200Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

300Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

400Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

Lemur continued…

500 Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

600Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

700Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

Lemur continued…

800Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

900Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

1000Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

LUCENE:

100Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	63.122237	1.574335	65	7.6	N/A	N/A	N/A	N/A
2	52.53453	1.673544	65	7.6	N/A	N/A	N/A	N/A
3	50.054606	1.771834	65	7.6	N/A	N/A	N/A	N/A
4	51.040664	1.946647	65	7.6	N/A	N/A	N/A	N/A
5	51.856621	1.362098	65	7.6	N/A	N/A	N/A	N/A
6	50.256994	1.487982	65	7.6	N/A	N/A	N/A	N/A
7	52.435655	1.522171	65	7.6	N/A	N/A	N/A	N/A
8	52.29573	1.597089	65	7.6	N/A	N/A	N/A	N/A
9	50.053139	1.552935	65	7.6	N/A	N/A	N/A	N/A
10	50.433638	1.702806	65	7.6	N/A	N/A	N/A	N/A

200Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	177.74783	1.565262	131	27.2	210.422472	1.695863	196	35.6
2	175.843915	1.864005	131	27.2	246.77073	1.713195	196	35.6
3	182.186013	1.695161	131	27.2	216.504443	2.304655	196	35.6
4	176.21233	1.948567	131	27.2	213.916549	1.691486	196	35.6
5	174.677282	1.545842	131	27.2	213.875585	1.762199	196	35.6
6	179.959958	2.018718	131	27.2	214.02471	1.676902	196	35.6
7	181.307141	1.587822	131	27.2	216.093115	1.866939	196	35.6
8	177.828092	1.75457	131	27.2	211.677921	1.538332	196	35.6
9	178.267618	2.344814	131	27.2	215.441987	2.248859	196	35.6
10	177.365712	1.651539	131	27.2	213.210575	2.448099	196	35.6

300Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	219.313814	1.645569	175	34.7	260.427139	2.361472	240	42.4
2	218.332557	1.722538	175	34.7	263.148349	1.850508	240	42.4
3	218.756845	1.712257	175	34.7	261.623565	1.728514	240	42.4
4	219.545647	1.958006	175	34.7	260.808574	2.318386	240	42.4
5	221.295259	2.244999	175	34.7	260.514526	1.624422	240	42.4
6	220.054144	1.484102	175	34.7	261.130878	2.241196	240	42.4
7	218.156867	1.704687	175	34.7	259.896126	2.28579	240	42.4
8	220.718989	1.862489	175	34.7	260.636049	1.905749	240	42.4
9	219.482281	1.926081	175	34.7	262.054677	1.63368	240	42.4
10	219.827826	1.872794	175	34.7	258.231901	1.797411	240	42.4

Lucene continued…

400Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	273.298358	2.234558	244	43	316.052404	2.1347	309	50.7
2	273.912062	1.951741	244	43	316.827855	2.549435	309	50.7
3	274.784976	4.253344	244	43	312.990444	2.215043	309	50.7
4	275.247793	2.116409	244	43	315.210787	2.3936	309	50.7
5	273.30187	1.486865	244	43	315.942697	1.744726	309	50.7
6	277.345636	1.878162	244	43	316.442497	2.014278	309	50.7
7	273.185261	2.175504	244	43	312.246406	1.683387	309	50.7
8	272.774318	2.18234	244	43	315.549954	2.136192	309	50.7
9	273.485185	2.180358	244	43	315.33941	1.913755	309	50.7
10	273.99183	1.813105	244	43	317.196429	2.271424	309	50.7

500 Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

600Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	433.439707	2.228462	365	66.1	467.485328	2.998099	430	73.9
2	417.930894	2.044527	365	66.1	463.821936	2.797954	430	73.9
3	417.294259	4.6213	365	66.1	463.665343	2.035507	430	73.9
4	418.465438	4.412922	365	66.1	464.573695	1.925058	430	73.9
5	417.070719	4.10677	365	66.1	457.482407	2.580538	430	73.9
6	415.143221	2.250366	365	66.1	466.438443	3.028964	430	73.9
7	420.432104	2.424891	365	66.1	464.557141	2.79499	430	73.9
8	418.2263	2.428003	365	66.1	469.000874	3.261354	430	73.9
9	419.94844	2.201181	365	66.1	467.361984	3.45548	430	73.9
10	457.713846	1.97908	365	66.1	475.842747	3.154794	430	73.9

Lucene continued…

700Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	491.924544	3.433424	427	76.9	526.493848	4.198918	492	84.7
2	491.408222	2.037986	427	76.9	524.316395	2.693266	492	84.7
3	480.083117	2.003781	427	76.9	521.641515	2.088098	492	84.7
4	476.230437	2.355749	427	76.9	516.183731	2.517055	492	84.7
5	470.927275	2.14118	427	76.9	518.933787	2.015146	492	84.7
6	473.515378	2.162141	427	76.9	514.817749	2.500054	492	84.7
7	472.210825	2.41637	427	76.9	516.627082	2.498374	492	84.7
8	472.686392	2.084741	427	76.9	508.563753	3.505239	492	84.7
9	472.812436	2.144382	427	76.9	513.531059	2.787116	492	84.7
10	474.760332	2.373967	427	76.9	518.6733	2.855572	492	84.7

800Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	524.987048	3.726614	423	87.9	569.571347	2.398389	488	95.6
2	525.974853	2.637416	423	87.9	566.317325	2.48319	488	95.6
3	524.225934	2.565637	423	87.9	565.401243	3.103806	488	95.6
4	526.892503	2.819521	423	87.9	573.986987	2.316458	488	95.6
5	529.786003	2.934821	423	87.9	566.16299	2.837683	488	95.6
6	526.636959	2.610346	423	87.9	573.784522	2.048381	488	95.6
7	524.974221	2.364573	423	87.9	570.299358	3.227141	488	95.6
8	532.608395	4.040977	423	87.9	566.521156	2.447139	488	95.6
9	526.609224	3.675787	423	87.9	569.251651	2.578132	488	95.6
10	526.711607	3.851688	423	87.9	571.318905	2.614779	488	95.6

900Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	557.963035	3.026632	524	90	665.454997	2.148956	589	91.9
2	555.971859	3.592349	524	90	655.544167	1.843052	589	91.9
3	561.575163	2.372933	524	90	655.361662	1.532486	589	91.9
4	558.767274	2.922597	524	90	651.194772	1.55895	589	91.9
5	559.831402	2.283193	524	90	656.329937	1.952142	589	91.9
6	558.505372	4.024078	524	90	664.506683	3.956031	589	91.9
7	560.674398	2.538114	524	90	657.905436	1.628857	589	91.9
8	554.886053	2.207305	524	90	651.506822	1.664549	589	91.9
9	561.431457	2.881173	524	90	649.569176	3.864497	589	91.9
10	557.31002	3.630187	524	90	653.850639	2.255003	589	91.9

Lucene continued…

1000Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	693.245503	1.902618	575	96.9	733.906198	1.654254	640	104.7
2	745.204349	1.991987	575	96.9	730.129896	1.615	640	104.7
3	702.84981	1.570829	575	96.9	737.243484	2.280493	640	104.7
4	691.996003	1.750636	575	96.9	730.98687	2.486036	640	104.7
5	699.975631	1.744646	575	96.9	722.041524	1.774964	640	104.7
6	683.541726	4.018959	575	96.9	734.025456	1.511143	640	104.7
7	695.877367	2.143901	575	96.9	726.952189	2.692475	640	104.7
8	692.015924	2.07762	575	96.9	735.888998	3.196286	640	104.7
9	696.341519	2.140079	575	96.9	733.479274	1.804426	640	104.7
10	689.570663	1.675664	575	96.9	724.697314	2.046188	640	104.7

ZETTAIR:

100Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	10.662934	0.807867	83	10.3	N/A	N/A	N/A	N/A
2	7.37002	0.575654	83	10.3	N/A	N/A	N/A	N/A
3	7.116065	0.522644	83	10.3	N/A	N/A	N/A	N/A
4	7.038981	0.542291	83	10.3	N/A	N/A	N/A	N/A
5	6.159926	0.396657	83	10.3	N/A	N/A	N/A	N/A
6	5.559084	0.352269	83	10.3	N/A	N/A	N/A	N/A
7	6.827147	1.175085	83	10.3	N/A	N/A	N/A	N/A
8	6.123321	0.478121	83	10.3	N/A	N/A	N/A	N/A
9	5.791912	0.516087	83	10.3	N/A	N/A	N/A	N/A
10	5.520724	0.453055	83	10.3	N/A	N/A	N/A	N/A

200Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	29.643824	0.591093	180	35.6	33.1353	0.819006	263	45.1
2	22.932526	1.033671	180	35.6	28.954745	0.704664	263	45.1
3	21.864178	1.309563	180	35.6	29.164891	0.666989	263	45.1
4	21.895306	1.263365	180	35.6	27.697237	0.633883	263	45.1
5	24.183971	1.110876	180	35.6	35.329702	0.610725	263	45.1
6	27.614373	0.616902	180	35.6	29.849075	0.681141	263	45.1
7	26.714664	0.624276	180	35.6	29.110057	1.497084	263	45.1
8	25.934431	0.536882	180	35.6	26.517368	1.457364	263	45.1
9	27.005839	0.791197	180	35.6	30.780945	1.232078	263	45.1
10	25.437725	0.57655	180	35.6	26.734643	1.627276	263	45.1

Zettair continued…

300Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	32.089584	0.65117	239	45.9	36.471797	0.713641	322	55.2
2	28.053022	1.178768	239	45.9	38.131152	0.672798	322	55.2
3	33.651638	0.637774	239	45.9	35.391731	0.677684	322	55.2
4	31.347814	1.435268	239	45.9	38.150113	0.671357	322	55.2
5	32.234022	0.651974	239	45.9	35.210876	0.964134	322	55.2
6	33.742643	0.674958	239	45.9	38.106898	0.683223	322	55.2
7	29.97697	0.610465	239	45.9	33.916971	1.222387	322	55.2
8	28.166596	0.610738	239	45.9	39.544011	0.622055	322	55.2
9	32.779776	0.565738	239	45.9	35.909442	1.205188	322	55.2
10	31.465824	1.376205	239	45.9	39.279978	0.690537	322	55.2

400Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	44.665126	0.583564	318	55.8	45.097391	1.142769	401	65
2	39.644017	0.970495	318	55.8	43.853382	0.745345	401	65
3	40.207798	0.580961	318	55.8	44.278766	0.623913	401	65
4	39.375908	0.578919	318	55.8	44.447743	0.675675	401	65
5	38.199939	1.082533	318	55.8	43.652557	0.629856	401	65
6	38.352504	0.842319	318	55.8	47.30154	0.688997	401	65
7	40.164858	1.510217	318	55.8	45.493774	0.69128	401	65
8	39.80292	1.391845	318	55.8	45.200896	0.643931	401	65
9	41.992684	0.560349	318	55.8	45.222282	0.636847	401	65
10	40.219136	0.570733	318	55.8	46.516012	0.526634	401	65

500 Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	53.492629	0.703514	375	74.6	55.704013	1.1696	458	83.8
2	51.032934	1.23626	375	74.6	55.064168	0.661649	458	83.8
3	50.535037	0.76428	375	74.6	55.584946	0.793498	458	83.8
4	48.944934	1.073663	375	74.6	56.076195	0.64614	458	83.8
5	49.888551	1.31842	375	74.6	54.210531	0.672722	458	83.8
6	51.601548	1.154174	375	74.6	54.559578	0.709884	458	83.8
7	48.721213	0.640161	375	74.6	55.153607	0.721412	458	83.8
8	49.68846	0.642062	375	74.6	56.851008	0.658625	458	83.8
9	49.75042	0.658099	375	74.6	59.351951	0.859355	458	83.8
10	47.421513	1.131184	375	74.6	58.097472	1.107618	458	83.8

Zettair continued…

600Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	58.678855	1.336742	470	84.9	61.101582	0.694991	553	94
2	59.61827	0.661158	470	84.9	62.072583	0.668163	553	94
3	57.374289	1.206898	470	84.9	59.417957	0.63778	553	94
4	59.104311	0.735013	470	84.9	61.885671	0.982694	553	94
5	55.754742	0.636009	470	84.9	61.965238	0.678898	553	94
6	61.2885	0.802432	470	84.9	59.32668	1.008888	553	94
7	58.35032	0.659441	470	84.9	62.335107	0.606134	553	94
8	57.325865	0.841301	470	84.9	59.141629	0.605841	553	94
9	61.751267	0.843442	470	84.9	61.598741	0.758533	553	94
10	58.937917	0.624621	470	84.9	65.685811	0.671667	553	94

700Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	69.063789	0.672927	549	97	70.475312	0.890423	632	106.1
2	65.742785	0.637594	549	97	67.811954	1.516257	632	106.1
3	67.973849	0.767749	549	97	68.478022	1.767273	632	106.1
4	67.788217	1.356424	549	97	72.625338	0.54769	632	106.1
5	68.857221	0.739023	549	97	71.034654	0.657038	632	106.1
6	69.16027	0.740882	549	97	71.41162	0.744336	632	106.1
7	69.497625	0.589703	549	97	71.218746	0.654078	632	106.1
8	69.466338	0.747422	549	97	71.542647	0.693845	632	106.1
9	69.487487	0.664481	549	97	71.070508	0.640145	632	106.1
10	70.413424	0.697796	549	97	69.792912	0.603725	632	106.1

800Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	74.785154	0.633268	581	106.8	78.370741	0.686297	664	115.9
2	73.933126	1.539775	581	106.8	79.262985	0.682378	664	115.9
3	74.499394	0.652924	581	106.8	78.375671	1.75988	664	115.9
4	75.559114	0.685321	581	106.8	78.369158	1.049854	664	115.9
5	74.580933	1.636118	581	106.8	79.936266	0.675547	664	115.9
6	76.740122	0.725666	581	106.8	79.634222	0.714013	664	115.9
7	75.29083	0.675437	581	106.8	78.571699	0.618874	664	115.9
8	73.407301	0.599033	581	106.8	80.307453	0.810868	664	115.9
9	75.358674	0.74395	581	106.8	77.658543	0.707152	664	115.9
10	76.985632	0.608669	581	106.8	79.895535	0.844711	664	115.9

Zettair continued…

900Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	80.025597	0.627615	680	113.4	81.330284	0.705412	763	122.5
2	75.598769	1.162122	680	113.4	82.090854	1.192252	763	122.5
3	79.46144	0.670174	680	113.4	82.757604	0.645213	763	122.5
4	79.249742	1.137273	680	113.4	82.042059	0.847009	763	122.5
5	79.701567	0.763202	680	113.4	79.077766	0.677471	763	122.5
6	79.683081	0.693351	680	113.4	79.719173	0.66734	763	122.5
7	77.072258	0.641642	680	113.4	81.909445	0.787507	763	122.5
8	77.453739	0.819428	680	113.4	80.835435	0.66465	763	122.5
9	77.832633	1.707959	680	113.4	80.140394	0.686345	763	122.5
10	76.927967	1.02638	680	113.4	85.046958	0.660585	763	122.5

1000Docs	Index Creation	First Search	Matches	Index Size	Index Maintenance	Second Search	Matches	Updated Index Size
1	89.727862	1.070275	743	127.9	91.243456	0.837715	826	136.9
2	87.898204	1.790733	743	127.9	92.703015	0.644017	826	136.9
3	88.866871	0.646771	743	127.9	92.947796	0.652492	826	136.9
4	90.491086	1.005634	743	127.9	91.945198	1.000599	826	136.9
5	89.617562	0.590695	743	127.9	90.872818	0.867258	826	136.9
6	87.942989	0.691292	743	127.9	93.469708	0.761637	826	136.9
7	89.241326	0.703243	743	127.9	90.855822	1.357774	826	136.9
8	87.454198	2.143745	743	127.9	90.262373	0.7472	826	136.9
9	86.965054	0.687296	743	127.9	93.31768	0.713921	826	136.9
10	91.471342	0.753063	743	127.9	89.671236	1.83855	826	136.9

Appendix M: 11800-8.txt

An excerpt from 11800-8.txt.

A. E. UHE, by Frederick H. Martens.
(Little biographies: series 1, Musicians)
© 30Dec22, A695089. R59809,
17Mar50, Breitkopf Publications,
inc., successor to Breitkopf & Haertel,
inc. (PWH)

ABBOTT'S DIGEST OF ALL THE NEW YORK REPORTS.
Apr., July, Oct., Dec. 1922.
© 31May22, B528574; 24Aug22, B548911;
2Dec22, B553698; 2Mar23, B571736.
R61801, R61806, R61808, R61815,
28Apr50, The Lawyers Co-operative
Publishing Co. (PCW)

ABRAHAM LINCOLN, an address by G. Lynn
Sumner; Lincoln Day, Kiwanis Club of
Scranton, Pa., Feb. 15, 1922.
© 12Apr22, A673456. R58500, 7Feb50,
G. Lynn Sumner (A)

ADVENTURES OF DOCTOR DOLITTLE, by Hugh
Lofting. (In the New York tribune,
Dec. 24-31, 1922) © 24Dec22, B542207;
25Dec22, B542208; 26Dec22, B542209;
27Dec22, B542210; 28Dec22, B542211;
29Dec22, B542212; 30Dec22, B542213;
31Dec22, B542214. R63718-63725,
23Jun50, Josephine Lofting (W)

THE ADVERTISING MAN, by Earnest Elmo
Calkins. (Vocational series)
© 17Mar22, A659269. R59089, 6Mar50,
Earnest Elmo Calkins (A)