Information Retrieval: A Survey

30 November 2000

by
Ed Greengrass

Abstract

Information Retrieval (IR) is the discipline that deals with retrieval of unstructured data, especially textual documents, in response to a query or topic statement, which may itself be unstructured, e.g., a sentence or even another document, or which may be structured, e.g., a boolean expression. The need for effective methods of automated IR has grown in importance because of the tremendous explosion in the amount of unstructured data, both internal, corporate document collections, and the immense and growing number of document sources on the Internet. This report is a tutorial and survey of the state of the art, both research and commercial, in this dynamic field. The topics covered include: formulation of structured and unstructured queries and topic statements, indexing (including term weighting) of document collections, methods for computing the similarity of queries and documents, classification and routing of documents in an incoming stream to users on the basis of topic or need statements, clustering of document collections on the basis of language or topic, and statistical, probabilistic, and semantic methods of analyzing and retrieving documents.

1. Introduction

This report provides an overview of the state of the art in Information Retrieval (IR), both commercial practice and research. More specifically, it deals with the retrieval of unstructured and semi-structured *text* documents or messages. It doesn't claim to be exhaustive. If it were exhaustive today, it wouldn't be exhaustive tomorrow given the dynamic nature of the field.

The report is divided into five broad areas. Area one (chapters one and two) discusses the basic concepts and definitions, e.g., what IR means, what its goals are, what entities it attempts to retrieve, the criteria by which IR systems are evaluated (and the limitations of those criteria), and how IR differs from retrieval via a traditional DBMS.

Area two (chapters three to ten) discusses each of the major approaches to the generation of queries and their interpretation, by Information Retrieval engines: classical boolean, extended boolean, vector space, probabilistic, and semantic/natural language processing (NLP). It also discusses IR "querying" from the perspective of routing and classification of an incoming stream of documents (as distinct from their retrieval from fixed collections). (In the routing context, queries are often called *topics* or *classifications*.) Finally, it discusses methods of clustering documents within a collection as a form of unsupervised classification, and as an aid to efficient retrieval.

Area three (chapter eleven) discusses the automatic and interactive expansion and refinement of user-generated queries, e.g., on the basis of user "relevance" feedback. Additionally, it discusses the re-use of queries.

Area four (chapter twelve) discusses the "fusion" of streams of output documents resulting from multiple, parallel retrievals into a single ranked stream that can be presented to the user. Two kinds of fusion are discussed: (1) A given query may be issued to multiple document collections using a common IR method. The documents retrieved from each of those collections must be merged into a single stream, ideally the same stream that would have resulted if these separate collections had been integrated into a single collection. (2) The same query may be executed by multiple IR methods (or the same information need may be formulated as multiple queries). In this way, a single query or information need may result in multiple retrievals being applied to the same document collection, each retrieval returning a different set of documents or a different ranking of the documents retrieved. Again, the results of these multiple retrievals must be merged and ranked for presentation to the user.

Area five (chapter thirteen) discusses user interaction with IR systems, i.e., system aid in the formulation and refinement of queries, system display of data and retrieved results in ways that aid user understanding, user browsing of (and interaction with) displayed data and results, etc.

Area six (chapter fourteen) discusses the ANSI/NISO standard Z39.50, initially developed by the library community for searching and retrieving bibliographic records, now emerging as a generic standard for communicating with diverse IR engines. The discussion includes both the existing 1995 standard, and a proposed extension to the Z39.50 query capability, the type 102 query,

which reflects enhancements in IR technology, especially the ability to retrieve documents ranked by the likelihood that they satisfy the user's information need.

Area seven (chapter fifteen) illustrates the state of the art by discussing briefly six actual IR systems — four commercial and two research.

Area eight (chapter sixteen) discusses Web information retrieval, including general concepts, research approaches, and representative commercial Web IR engines.

2. What is Information Retrieval (IR)?

2.1 Definition and Terminology of Information Retrieval (IR)

The term "IR", as used in this paper, refers to the retrieval of *unstructured* records, that is, records consisting primarily of free-form natural language text. Of course, other kinds of data can also be unstructured, e.g., photographic images, audio, video, etc. However, IR research has focused on retrieval of natural language text, a reasonable emphasis given the importance and immense volume of textual data, on the internet and in private archives.

Some points of terminology should be clarified here. The records that IR addresses are often called "documents". That term will be used here. IR often addresses the retrieval of documents from an organized (and relatively static) repository, most commonly called a "collection". That term will also be used here. (The word "archive" is also used. So is the word "corpus". The term "digital library" is becoming very common. But the generic term "collection" is still the term most commonly used in the research literature.) However, it should be understood that IR is not restricted to static collections. The collection may be a stream of messages, e.g., E-mail messages, faxes, news dispatches, flowing over the internet or some private network.

2.1.1 Structured vs. Unstructured vs. Semi-Structured Documents

Records may be *structured*, *unstructured*, *semi-structured*, or a mixture of these types. A record is structured if it consists of named components, organized according to some well-defined syntax. Typically, a structured database will have multiple record types such that all records of a given type have the same syntax, e.g., all rows in a table of a relational database will have the same columns. [Date, 1981, Salton, 1983] Moreover, each component of a record will have a definite "meaning" ("semantics") and a given component of a given record type will have the same semantics in every record of that type. The practical effect is that given the name of a component, a search and retrieval engine (such as a DBMS) can use the syntax to find the given component in a given record and retrieve its contents, its "value". Similarly, given a component and a value, the search engine can find records such that the given component contains ("has") the given value. For example, a relational DBMS can be asked to retrieve the contents of the "age" column of an "Employee" table in a "Personnel" database. The DBMS knows how to find the "Employee" table within the "Personnel" database, and how to find the "age" column within each record of the "Employee" table. And every "age" column within the "Employee" table will have the same semantics, i.e., the age of some employee. The column name "age" may not be sufficient to iden-

tify the column; an “Equipment” table in the same or a different database may also have an “age” column. Hence in general, it may be necessary to specify a path, e.g., database name, table name, column name, to uniquely identify the syntactic component to the search engine. However, the syntax of a well-structured database is such that it is always possible to specify a given syntactic component uniquely and hence it is always possible for the search engine to find all occurrences of a given component. If the given component has a definite semantics, then it is always possible for the search engine to find data with that semantics, e.g., to find the ages of all employees.

By contrast, in a collection of unstructured natural language documents, there is no well-defined syntactic position where a search engine could find data with a given semantics, e.g., the ages of employees. In a random collection of documents, there is no guarantee that they are all about the same topic, e.g., employees. Even if it is known that the documents *are* all about employees, there is no guarantee that they all specify the age of an employee (or that any do). Even if it is known that some documents do specify the age of an employee, there is no simple well-defined way of telling where the age occurs in a given document, e.g., in what sentence or even in what paragraph. This is exactly what is meant by “unstructured;” there is no (externally) well-defined syntax for a given document, let alone a syntax that all the documents share. To the extent that the documents do share a syntax, there is no well-defined semantics associated with each syntactic component.

In some cases, a collection of textual documents may share a common structure and semantics, e.g., in a collection of documents containing facts about countries, each document may contain data about a different country. [CIA Fact Book] The first paragraph may contain the name, location and population of the country in sentences that follow a fairly consistent form. Similarly, the 2nd paragraph may list the principal industries and exports, again in sentences that follow a fairly standard form. Such a collection is called “semi-structured.” Although the data about a country does not occupy well-defined columns in a well-defined table, e.g., name, population, location, etc., as they would do in a structured database, the data nevertheless occupies fairly standard positions in the text of each document with further clues, e.g., key words like “population”, that make it fairly easy to write algorithms or at least heuristics for extracting the data and storing it in structured tables.

2.1.2 Unstructured Documents with Structured Headers

IR documents often are partly structured, e.g., they may have a structured header and an unstructured body. But this header typically contains *metadata,* i.e., data *about* the document, rather than the information content *of* the document. In a bibliographic document, e.g., a book, journal, or paper, this metadata may include author, title, publisher, publication date, subject, abstract, various catalogue numbers, etc. [Z39.50] In other words, the structured metadata is the data that would be found in a traditional library catalogue entry and is now found in an on-line library catalogue. In an E-mail message, the structured header will include the “from” line (originator), “to” line (addressee), subject line, copy line (copy recipients), classification, date, etc. In a fax of a business letter, the structured data may include a corporate letterhead, date, salutation, signature, etc. In all of these cases, the content or “body” of the document remains unstructured natural language text. Hence, a search engine may easily find documents written by a given author or pub-

lished after a given date. But finding documents that contain data on a particular topic is a much harder task. This task is one of the principal problems addressed by IR research.

2.1.3 Structure of a Document as a Document

IR documents are often structured in another way: They have a structure *as* documents. For example, a book may have a structure that consists of certain components by virtue of being a book, e.g., it contains a title page, chapters, etc. The chapters are composed of paragraphs which are composed of sentences, which are composed of words, etc. If the book is a textbook, it will typically have a richer structure including a table of contents, an introduction or preface, an index, a bibliography, etc. The chapters may contain figures, graphs, photographs, tables, citations, etc. This structure may be specified explicitly by a "markup" language such as SGML or HTML. Or the structure may be implicit in the format and organization of the book. A software tool may be able to recover much of this structure by using format clues such as indentation, key words (like "Index" or "Figure") etc., and mark up a document semi-automatically. But in all such cases, the structure is still metadata in the sense that it characterizes the organization of the document, not its semantic content. The search engine may be able to find chapter one, or section one, or figure one, easily. But finding a section dealing with a given topic, e.g., "information retrieval," or containing the value of an attribute of some real-world entity, e.g., the date on which a given organization was incorporated, is a much more difficult and far less well-defined problem.

2.1.4 Goals of Information Retrieval

IR focuses on retrieving documents based on the content of their unstructured components. An IR request (typically called a "query") may specify desired characteristics of both the structured and unstructured components of the documents to be retrieved, e.g., "The documents should be about 'Information retrieval' and their author must be 'Smith'." In this example, the query asks for documents whose body (the unstructured part) is "about" a certain topic and whose author (a structured part) has a specified value.

IR typically seeks to find documents in a given collection that are "about" a given topic or that satisfy a given *information need*. The topic or information need is expressed by a query, generated by the user. Documents that satisfy the given query in the judgment of the user are said to be "relevant." Documents that are not about the given topic are said to be "non-relevant." An IR engine may use the query to classify the documents in a collection (or in an incoming stream), returning to the user a subset of documents that satisfy some classification criterion. Naturally, the higher the proportion of documents returned to the user that she judges as relevant, the better the classification criterion. Alternatively, an IR engine may "rank" the documents in a given collection. To say that document D_1 is higher ranking with respect to a given query Q, than document D_2 may be interpreted probabalistically as meaning that D_1 is more likely to satisfy Q than D_2. Or it may be interpreted as meaning that D_1 satisfies Q more than D_2. The latter could mean that D_1 is more precisely focused on the need expressed by Q than D_2. Or it could mean that more of D_1 satisfies Q than D_2, e.g., D_1 might be entirely devoted to the need expressed by Q while D_2 might deal with a number of topics so that only a single paragraph of D_2 satisfies Q.

2.1.5 Ad-Hoc Querying vs. Routing

A distinction is often made between *routing* and *ad-hoc querying*. [TREC 3 Overview, Harman] In the latter, the user formulates any number of arbitrary queries but applies them to a fixed collection. In routing, the queries are a fixed number of topics; each message in an incoming (and hence constantly changing) stream of messages is classified according to which topic it fits most closely and "routed" to the class corresponding to that topic. (In many routing experiments, there is just one topic or query; hence, there are just two classes, relevant and non-relevant.)

2.1.6 Evaluation of IR Performance

At the heart of IR evaluation is the concept of "relevance". Relevance is an inherently subjective concept [Salton, 83, Pg 173] [Hersh, SIGIR '95] in the sense that satisfaction of human needs is the ultimate goal, and hence the judgment of human users as to how well retrieved documents satisfy their needs is the ultimate criterion of relevance. Moreover, human beings often disagree about whether a given document is relevant to a given query. [Hersh, SIGIR '95] Disagreement among human judges is even more likely when the question is not absolute relevance but degree of relevance. There are other complications: Relevance depends not only on the query and the collection but also on the context, e.g., the user's personal needs, preferences, knowledge, expertise, language, etc. [Hersh, SIGIR '95] {van Rijsbergen, SIGIR '89] Hence, a given document retrieved by a given query for a given user may be relevant to that user on one day but not on another. {Hersh, SIGIR '95] Or the given document may be relevant to one user but not to another even though they both issued the same query. (This may be either because their needs are different or because they "meant" different things by the nominally "same" query.) Or the document may be relevant if retrieved from one collection but not if retrieved from another collection. Or relevance of a document may depend on the order of retrieval, e.g., the second document retrieved is less relevant to a given user if the first document retrieved satisfies the user's need. In general, there is a difference between relevance to the topic of a given query and usefulness to the user who issued the given query. For this reason, some writers [Saracevic, 1997] [Korfhage, 1997] [Salton, 1983] distinguish between *relevance* to the user's query, and *pertinence* to the user's needs.

On the other hand, early experiments "that looked at differing relevance assessments, … found that for 'comparative purposes' (i.e., testing whether a certain technique is better than some other technique) any 'reasonable' set of relevance assessments gave the same ordering of techniques even though absolute performance scores differed." [Voorhees, pc]

There is no way of escaping completely from the concept of relevance. IR is fundamentally concerned with retrieving information (documents, abstracts, summaries, etc.) that match a user-specified need, i.e., that are relevant to the user. One approach to dealing with this subjectivity is to provide or generate "user profiles," i.e., knowledge about the user's needs, preferences, etc. The objective is to give the user not just what he asked for but what he "meant" by what he asked for. Or the profile may be generated automatically, based on statistics derived from documents the user has designated as relevant to his needs. [Yochum, TREC 4]

Two measures of IR success, both based on the concept of relevance [to a given query or information need], are widely used: “precision” and “recall.” Precision is defined as, “the ratio of relevant items retrieved to all items retrieved, or the probability given that an item is retrieved [that] it will be relevant.” [Saracevic, SIGIR ‘95] Recall is defined as, “the ratio of relevant items retrieved to all relevant items in a file [i.e., collection], or the probability given that an item is relevant [that] it will be retrieved.” [Saracevic, SIGIR ‘95] Other measures have been proposed, [Salton, ‘83, pps 172-186] [van Rijsbergen, 1979] but these are by far the most widely used. Note that there is an obvious trade-off here. If one retrieves all of the documents in a collection, then one is sure of retrieving all the relevant documents in the collection in which case the recall will be “perfect”, i.e., one. On the other hand, in the common situation where only a small proportion of the documents in a collection are relevant to the given query, retrieving everything will give a very low precision (close to zero). The usual plausible assumption is that the user wants the best achievable combination of good precision and good recall, i.e., ideally he would like to retrieve all the relevant documents and no non-relevant documents.

But this plausible assumption is open to some very serious objections. It is often the case that the user only wants a small subset of a (possibly large) set of relevant documents. The relevant documents may contain a lot of redundancy; a few of them may be sufficient to tell the user everything he wants to know {Hersh, SIGIR ‘95]. Or the user may be looking for evidence to support a hypothesis or to reduce uncertainty about the hypothesis; a few documents may provide sufficient evidence for that purpose. [Florance, SIGIR ‘95] Or the user may only want the most up-to-date documents on a given topic, e.g., for a lawyer the latest legal opinion or statute may have superseded earlier precedents or statutes.[Turtle, SIGIR ‘94] In general, it is often the case that there are multiple subsets of relevant documents such that any one of these subsets will satisfy the user’s requirement, rather than a single unique set of *the* relevant documents. On the other hand, two relevant documents may present contradictory views of some issue of concern to the user; hence, the user may be seriously misled if he only sees some of the relevant documents.

In practice, some users attach greater importance to precision, i.e., they want to see *some* relevant documents without wading through a lot of junk. Others attach greater importance to recall, i.e., they want to see the highest possible proportion of relevant documents. Hence, van Rijsbergen [1979] offers the E (for Effectiveness) measure, which allows one to specify the relative importance of precision and recall:

$$E \approx 1 - \frac{1}{\alpha\left(\frac{1}{P}\right) + (1-\alpha)\frac{1}{R}}$$

where P is *precision*, R is *recall*, and α is a parameter which varies from zero (user attaches no importance to precision), through one half (user attaches equal importance to precision and recall), to one (user attaches no importance to recall).

Measuring precision is (relatively) easy; if a set of competent users or judges agree on the relevance or non-relevance of each of the retrieved documents, then calculating the precision is straightforward. Of course, this assumes that the set of retrieved documents is of manageable size, as it must be if it is to be of value to the user. If the retrieved documents are ranked, one can

always reduce the size of the retrieved set by setting the threshold higher, e.g., only look at the top 100, or the top 20. Measuring recall is much more difficult because it depends on knowing the number of relevant documents in the entire collection, which means that all the documents in the entire collection must be assessed. [Saracevic, SIGIR '95] If the collection is large, this is not feasible. (The Text REtrieval, i.e., TREC, Conferences attempt to circumvent the problem by pooling samples, e.g., the top 100 documents, retrieved by each competing IR engine; the assumption is made that every relevant document is being retrieved by at least one of the competitors. [Harman, TREC 3 Overview, TREC 4] This works better for comparing engines than for computing an absolute measure of recall.)

Most of the IR systems discussed in this report return an ordered list of document, i.e., the documents are ranked according to some measure, often statistical or probabilistic, of the likelihood that they are relevant to the user's request. The usual assumption is that the user will start with the first document (or some surrogate like a title or summary), the document the system estimates as "best," and work her way down the list until her needs have been satisfied, or her patience exhausted. (But see some alternatives discussed under User Interaction.) Hence, another measure of system effectiveness is how many non-relevant documents the user has to examine before reaching the number of relevant documents she needs or desires. If the system returns 20 documents, and only three are relevant, the precision is 3/20 or 0.15. However, as a practical matter, it makes a considerable difference to the user whether the three relevant documents are the first three in the ordered list (she doesn't have to look at any non-relevant documents if those three satisfy her need), or the last three (she has to look at 17 non-relevant documents before she reaches the "good stuff.") Typically, the situation will be intermediate, e.g., the three relevant documents may appear at positions (ranks) four, seven, and 15. To complicate matters further, it is entirely possible that several documents will be tied according to the given systems measure, e.g., relevant document four and non-relevant documents five and six may receive the same probability of relevance value; in that case, the order of those three documents is arbitrary, i.e., it is equally likely that the relevant document will occupy positions four, five, or six. Hence, Cooper [JASIS, 1968] has proposed the "Expected Search Length (ESL)" for a given query *q*, a measure of the number of non-relevant documents the user will have to wade through to reach a specified number of relevant documents; Cooper's measure takes into account the uncertainty produced by ties.

A more common type of measure, widely used in the research community, e.g., in TREC reports, is *average precision*. This family of measures reflects the recognition that precision varies, generally falls, as recall increases. This variation can be (and frequently is) expressed directly as a graph of precision vs. recall. *Average precision* is an attempt to summarize this kind of curve as a single value, e.g., for the purpose of comparing different IR algorithms, or the same algorithm across different document collections. *Non-interpolated average precision*" corresponds to the area under an ideal (non-interpolated) recall/precision curve." [Harman, TREC-2] This is approximated by "computing the precision after every retrieved relevant document and then averaging these precisions over the total number of retrieved relevant documents" for a given query (or "topic" in TREC terminology). There will be a different average precision, in general, for each query. These averages can then be themselves averaged over all the queries employed by the experimenter.

Another common variation on average precision is the *eleven-point average precision*. The precision is calculated for recalls (in practice, estimated recalls) of zero, 0.1, 0.2,...,1.0. Then, these 11 precisions, computed for uniformly spaced values of recall, are averaged.

3. Approaches to IR - General

Broadly, there are two major categories of IR technology and research: semantic and statistical. Semantic approaches attempt to implement some degree of syntactic and semantic analysis; in other words, they try to reproduce to some (perhaps modest) degree the understanding of the natural language text that a human user would provide. In statistical approaches, the documents that are retrieved or that are highly ranked are those that match the query most closely in terms of some statistical measure. By far the greatest amount of work to date has been devoted to statistical approaches so these will be discussed first. (Indeed, even semantic approaches almost always use, or are used in conjunction with, statistical methods. This is discussed in detail later.)

Statistical approaches fall into a number of categories: boolean, extended boolean, vector space, and probabilistic. Statistical approaches break documents and queries into *terms*. These terms are the population that is counted and measured statistically. Most commonly, the terms are words that occur in a given query or collection of documents. The words often undergo pre-processing. They are "stemmed" to extract the "root" of each word. [Porter, Program, 1980] [Porter, Readings, 1997] The objective is to eliminate the variation that arises from the occurrence of different grammatical forms of the same word, e.g., "retrieve," "retrieved," "retrieves," and "retrieval" should all be recognized as forms of the same word. Hence, it should not be necessary for the user who formulates a query to specify every possible form of a word that he believes may occur in the documents for which he is searching. Another common form of preprocessing is the elimination of common words that have little power to discriminate relevant from non-relevant documents, e.g., "the", "a", "it" and the like. Hence, IR engines are usually provided with a "stop list" of such "noise" words. Note that both stemming and stop lists are language-dependent.

Some sophisticated engines also extract "phrases" as terms. A phrase is a combination of adjacent words which may be recognized by frequency of co-occurrence in a given collection or by presence in a phrase dictionary.

At the other extreme, some engines break documents and queries into "*n*-grams", i.e., arbitrary strings of *n* consecutive characters. [Damashek, 1995] This my be done, e.g., by moving a "window" of *n* characters in length through a document or query one character at a time. In other words, the first *n*-gram will consist of the first n characters in the document, the 2nd *n*-gram will consist of the 2nd through the (*n*+1)-th character, etc. (Early research used $n = 2$, $n = 3$; recent applications have used values of $n=5$, and $n=6$ but the user is free to use the value of *n* that works best for his application.) The window can be moved through the entire document, completely ignoring word, phrase, or punctuation boundaries. Alternatively, the window can be constrained by word separators, or by other punctuation characters, e.g., the engine can gather n-gram statistics separately for each word. [Zamora et al., IP&M, 1981] [Suen, IEEE Pattern, 1979] Thirdly, *n*-grams can be gathered and counted without regard to word boundaries, but then words or phrases can be evaluated in terms of *n*-gram statistics. [Cohen, 1995] In any case, since a single word or phrase can generate multiple *n*-grams, statistical clustering using n-grams has proved to

be language-independent, and has even been used to sort documents *by* language, or by topic within language. For similar reasons, *n*-gram statistics appear to be relatively insensitive to degraded text, e.g., spelling errors, typos, errors due to poor print quality in OCR transmission, etc. [Pearce et al., 1996]

Numeric weights are commonly assigned to document and query terms. A weight is assigned to a given term within a given document, i.e., the same term may have a different weight in each distinct document in which it occurs. The weight is usually a measure of how effective the given term is likely to be in distinguishing the given document from other documents in the given collection. The weight is often normalized to be a fraction between zero and one. Weights can also be assigned to the terms in a query. The weight of a query term is usually a measure of how much importance the term is to be assigned in computation of the similarity of documents to the given query. As with documents, a given term may have a different weight in one query than in another. Query term weights are also usually normalized to be fractions between zero and one.

4. Classical Boolean Approach to IR

In the boolean case, the query is formulated as a boolean combination of terms. A conventional boolean query uses the classical operators AND, OR, and NOT. The query "t_1 AND t_2" is satisfied by a given document D_1 if and only if D_1 contains both terms t_1 and t_2. Similarly, the query "t_1 OR t_2" is satisfied by D_1 if and only if it contains t_1 or t_2 or both. The query "t_1 AND NOT t_2" satisfies D_1 if and only if it contains t_1 and does *not* contain t_2. More complex boolean queries can be built up out of these operators and evaluated according to the classical rules of boolean algebra. Such a classical boolean query is either true or false. Correspondingly, a document either satisfies such a query (is "relevant") or does not satisfy it (is non-relevant"). No ranking is possible, a significant limitation. [Harman, JASIS, 1992] Note however that if stemming is employed, a query condition specifying that a document must contain the word "retrieve" will be satisfied by a document that contains any of the forms "retrieve", "retrieves", "retrieved", "retrieval", etc.

Several kinds of refinement of this classical boolean query are possible when it is applied to IR. First, the query may be applied to a specified syntactic component of each document, e.g., the boolean condition may be applied to the title or the abstract rather than to the document as a whole.

Second, it may be specified that the condition must apply to a specified position within a syntactic component, e.g., to the words at the beginning of the title rather than to any part of the title.

Third, an additional boolean operator may be added to the classical set:, a "proximity" operator. [Z39.50-1995] A proximity operator specifies how close in the text two terms must be to satisfy the query condition. In its general form, the proximity operator specifies a unit, e.g., word, sentence, paragraph, etc., and an integer. For example, the proximity operator may be used to specify that two terms must not only both occur in a given document but must be within n words of each other; e.g., n = 0 may mean that the words must be adjacent. Similarly, the operator may specify that two terms must be within n sentences of each other, etc. A proximity operator can be applied to boolean conditions as well as to simple terms, e.g., it might specify that a sentence satisfying one boolean condition must be adjacent to a sentence satisfying some other boolean condition. A

proximity operator may specify order as well as proximity, e.g., not only how close two words must be but in what order they must occur.

The classical boolean approach does not use term weights. Or, what comes to the same thing, it uses only two weights, zero (a term is absent) and one (a term is present).

The classical boolean model can be viewed as a crude way of expressing phrase and thesaurus relationships. For example, t_1 AND t_2 says that both terms t_1 and t_2 must be present, a condition that is applicable if the two terms form a phrase. If a proximity operator is employed, the boolean condition can be made to say that t_2 must immediately follow t_1 in the text, which corresponds still more closely (though still crudely) to the conventional meaning of a "phrase." Similarly, t_1 OR t_2 says that either t_1 or t_2 can serve as an index term to relevant documents, i.e., in some sense t_1 and t_2 are "equivalent". This is roughly speaking what we mean when we assign t_1 and t_2 to the same class in a thesaurus. In fact, some systems use this viewpoint to generate expanded boolean conditions automatically, e.g., given a set of query terms supplied by the user, "a boolean expression is composed by ORing each … query term with any stored synonyms and then ANDing these clusters together." [Anick, SIGIR '94]

4.1 Automatic Generation of Boolean Queries

The logical structure of Boolean queries, which is their greatest virtue, is also one of their most serious drawbacks. Non-mathematical or novice users often experience difficulty in formulating Boolean queries.[Harman, JASIS, 1992] Indeed, they often misinterpret the meaning of the AND and OR operators. (In particular, they often use "AND," set intersection, when "OR," set union, is intended.) [Ogden & Kaplan, cited in Ogden & Bernick, 1997] This has led to schemes for automatic generation of Boolean queries. [Anick, SIGIR'94] [Salton, IP&M, 1988]

In the Anick approach mentioned above, the query terms (presumably after stemming, removing stop words, etc.) are Ored together. Each OR term is expanded with any synonyms from an online thesaurus. The Salton approach, by contrast, imposes a Boolean structure on the terms supplied by the user. No thesaurus is employed.

Salton starts with a natural language query. The usual stemming and removal of stop words generates a set of user terms, which are ORed together as in the Anick approach. However, Salton then looks for pairs (and triples) of these user-supplied terms that co-occur in one or more documents. Since two or three of these user terms might occur in the same document by chance, Salton then uses a formula for pairwise correlation to determine if any given pair of co-occurring terms T_i and T_j co-occur more frequently than would be expected by chance alone. A similar correlation formula is used for co-occurring triples, i.e., three of the user-supplied words occurring in the same document. Each pair or triple whose computed correlation exceeds a pre-determined threshold is then grouped with a Boolean AND, e.g., if the pair t_i, t_j and the triple t_l, t_m, t_n exceed the threshold, then the automatically generated Boolean query (assuming t terms) becomes:

t_1 OR t_2 OR … OR (t_i AND t_j) OR (t_l AND t_m AND t_n) … OR t_t

It should be stressed again that the pairs are triples are drawn entirely from the terms originally supplied by the user; no thesaurus-based expansion as with Anick, and no query expansion based on relevance feedback (see below) is employed. However, combining these various techniques is certainly feasible.

As a further refinement, Salton ranks the terms (single terms, pairs, and triples) in the automatically generated Boolean expression in descending order by inverse document frequency. (See below for a definition of *idf*. High-*idf* terms tend to be better discriminators of relevance than low-*idf* terms.) He can estimate the number of documents that a given term (or pair or triple) will be responsible for retrieving from its frequency of occurrence in documents. If the total estimated number of documents that will be retrieved by the Boolean query exceeds the number of documents that the user wants to see, he can reduce the estimated number by deleting OR terms from the query starting with those that have the lowest idf ranking. This gives the user, not a true relevance ranking of documents, but at least some control over the number retrieved, something that ordinary Boolean retrieval does not provide.

5. Extended Boolean Approach

Even with the addition of a proximity operator, boolean conditions remain classical in the sense that they are either true or false. Such an all-or-nothing condition tends to have the effect that either an intimidatingly large number of documents or none at all are retrieved. [Harman, JASIS, 1992] Classical Boolean models also tend to produce counter-intuitive results because of this all-or-nothing characteristic, e.g., in response to a multi-term OR, “a document containing all [or many of] the query terms is not treated better than a document containing one term.” [Salton et al., IP&M, 1988] Similarly, in response to a multi-term AND, “[A] document containing all but one query term is treated just as badly as a document containing no query term at all.” [Salton et al., IP&M, 1988] A number of extended boolean models have been developed to provide ranked output, i.e., provide output such that some documents satisfy the query condition more closely than others. [Lee, SIGIR ‘94] These extended boolean models employ extended boolean operators (also called “soft boolean” operators).

Extended boolean operators make use of the weights assigned to the terms in each document. A classical boolean operator evaluates its arguments to return a value of either true or false. These truth values are often represented numerically by zero (false — or in IR terms “doesn’t match given document”) and one (true — or in IR terms “matches given document”). An extended boolean operator evaluates its arguments to a number in the range zero to one, corresponding to the estimated degree to which the given logical expression matches the given document. Lee [SIGIR ‘94] has examined a number of extended Boolean models [Paice, 1984] [Waller et al., 1979] [Zimmerman, 1991] and proved that by certain significant (but not necessarily the *only* significant) criteria, a model called “*p*-norm” [Salton et al., CACM 1983] has the most desirable properties. By “most desirable” is meant that the *p*-norm model tends to evaluate the degree to which a document matches (satisfies) a query more in accordance with a human user’s judgment than the other models. For each of the other models examined, there are cases where the model’s evaluation of the degree of query/document match is at variance with a human user’s intuition. In each of those cases, the *p*-norm model’s evaluation of match agrees with a human user’s intuition.

Given a query consisting of n query terms t_1, t_2, …, t_n, with corresponding weights w_{q1}, w_{q2}, …, w_{qn}, and a document D, with corresponding weights w_{d1}, w_{d2}, …, w_{dn} for the same n terms, the p-norm model defines similarity functions for the extended boolean AND and extended boolean OR of the n terms. The extended boolean AND function computes the similarity of the given document with a query that ANDs the given terms together. Similarly, the extended boolean OR function computes the similarity of the given document with a query that ORs the given terms together. Each similarity is computed as a number in the closed interval [0, 1]. More elaborate boolean queries can obviously be composed from the AND and OR functions. The extended boolean functions for the p-norm model are given by:

$$SIM_{AND}(d, (t_1,w_{q1})AND\ldots AND(t_n,w_{qn})) = 1 - \left(\frac{\sum_{i=1}^{n} ((1 - w_{di})^p \bullet {w_{qi}}^p)}{\sum_{i=1}^{n} {w_{qi}}^p} \right)^{\frac{1}{p}} \qquad ,(1 \le p \le \infty)$$

and

$$SIM_{OR}(d, (t_1,w_{q1})OR\ldots OR(t_n,w_{qn})) = \left(\frac{\sum_{i=1}^{n} ({w_{di}}^p \bullet {w_{qi}}^p)}{\sum_{i=1}^{n} {w_{qi}}^p} \right)^{\frac{1}{p}} \qquad ,(1 \le p \le \infty)$$

The p-norm model has a parameter that can be used to “tune” the model; most of the other models studied by Lee also have such a parameter, though the effect and interpretation of the parameter varies with the model. The parameter p in the p-norm model can vary from one to infinity and has a very clear interpretation. At p= infinity, the p-norm model is equivalent to the classical boolean model; AND corresponds to strict phrase assignment (i.e., all the components of the phrase must be present for the AND to evaluate to non-zero), OR to strict thesaurus class assignment (i.e., presence of any one member of the class is sufficient for the OR to evaluate to one; there is no additional score if two or more are present.). At low to moderate p, e.g., between 2 and 5, AND corresponds to loose phrase assignment, i.e., “the presence of all phrase components is worth more than the presence of only some of the components; terms are not compulsory.” That is, the p-norm AND generalizes the strict boolean AND in the sense that a single low-weighted term substantially lowers the total similarity score, even if all the other terms have high weights. On the other hand, the p-norm AND differs from the strict boolean AND because a single zero weighted, i.e., missing, term does not reduce the total similarity score to zero. Similarly, at low to moderate p, OR corresponds to loose thesaurus class assignment, i.e., “the presence of several terms from a class is worth more than the presence of only one term.” In other words, the p-norm OR generalizes the strict boolean OR in the sense that a single high-weighted term can produce a fairly high total similarity score even if all the other terms are low-weighted or missing (zero weighted). On the other hand, the p-norm OR differs from the strict boolean OR because a single high-weighted term is not enough to maximize the similarity score; additional non-zero terms will increase the total score to some degree. At p = 1, the p-norm model reduces to the pure “vector space” model which is discussed in the next section, i.e., “terms are independent of each other; distinction between phrase and thesaurus assignment disappears.” In fact, at p = 1, AND and OR become identical. [Salton, et al., CACM 1983] They both become identical to cosine similarity, discussed in the next section.

The classical boolean operators AND and OR are binary, i.e., they connect two terms. However, they are also associative, i.e., t_1 AND (t_2 AND t_3) is equivalent to (t_1 AND t_2) AND t_3. This is not true for the *p*-norm model (and some of the other extended boolean models). The *p*-norm model (and the other models with the same problem) circumvent this difficulty by defining the extended boolean operators as n-ary, i.e., connecting n terms, rather than binary. So the above boolean expression becomes AND (t_1, t_2, t_3). {Lee, SIGIR '94] This expression is true if and only if all three terms are present.

The *p*-norm model supports assignment of weights to the query terms as well as the document terms. The *p*-norm formulas extend to this case in a quite straightforward manner. The weights are relative rather than absolute, e.g., the query (t_1, 1) AND (t_2, 1) with a weight of one assigned to each term is exactly equivalent to the query (t_1, 0.1) AND (t_2, 0.1) with a weight of 0.1 assigned to each term. This is so because the *p*-norm formulas normalize the query weights. Relative weights are easier and more natural for the user to assign than absolute weights. It is easier for a user to say that t_1 is more important (or even twice as important) than t_2 than to say exactly *how* important either term is. {Lee, SIGIR '94]

A further degree of flexibility can be achieved in the *p*-norm model by permitting the user to assign a different value of *p* to each boolean operator in a given boolean expression. This allows the user to say, e.g., that a strict phrase interpretation should be given to one AND in the given expression, a looser interpretation to another AND in the same expression, etc. {Salton et al., CACM 1983]

What makes the p-norm model superior to the alternatives surveyed by Lee? Its primary advantage is that *it gives equal importance to all its operands.*This does *not* mean that it ignores document and term weights. On the contrary, the document weights (assigned typically by the program that indexes the document collection), and the query weights (assigned typically by the user who formulates the query, although automatic modification of these weights is discussed in a later section), are essential elements of the *p*-norm functions as given above. What "equal importance" means is that the *p*-norm functions evaluate all term weights in the same way; they do not give special importance to certain terms on the basis of their ordinal positions, i.e., any permutation of term order is equivalent to any other. Moreover, *p*-norm does not give special importance to the terms with minimum or maximum weights, to the exclusion of other terms. For example, one or two high-weighted query terms in a given document will yield a high (relatively close to 1) value of a *p*-norm OR for the given document relative to the given query. It doesn't matter in the least *which* terms are highly weighted. Moreover, if other query terms are also present in the given document, they will add to the value of the OR even if they have neither the maximum nor the minimum weight in the set of query terms matching the given document ("match" terms).

A probabilistic form of extended boolean has been developed [Greiff, SIGIR '97] in which a probabilistic OR is computed in terms of the probability of its component terms, and similarly for AND. See section on "Bayesian Inference Network Model" for further details.

The commercial IR system, Topic, supports a form of extended boolean query called a "topic." These queries can combine strict and extended boolean operators. See discussion in section 6.4.

Experiments have shown that the extended boolean model can achieve greater IR performance than either the classical boolean or the vector space model. But there is a price. Formulating effective extended boolean queries obviously involves more thought and expertise in the query domain than formulating either a classical boolean query, or a simple set of terms with or without weights as in the vector space model.

6. Vector Space Approach

6.1 Building Term Vectors in Document Space

One common approach to document representation and indexing for statistical purposes is to represent each textual document as a set of terms. Most commonly, the terms are words extracted automatically from the documents themselves, although they may also be phrases, n-grams, or, manually assigned descriptor terms. (of course, any such term-based representation sacrifices information about the order in which the terms occur in the document, syntactic information, etc.) Often, if the terms are words extracted from the documents, "stop" words (i.e., "noise" words with little discriminatory power) are eliminated, and the remaining words are stemmed so that only one grammatical form (or the stem common to all the forms) of a given word or phrase remains. (Stop lists and stemming can sometimes be avoided if the terms are *n*-grams — see discussion below.) We can apply this process to each document in a given collection, generating a set of terms that represents the given document. If we then take the union of all these sets of terms, we obtain the set of terms that represents the entire collection. This set of terms defines a "space" such that each distinct term represents one dimension in that space. Since we are representing each document as a set of terms, we can view this space as a "document space". [Salton, 1983] [Salton, 1989]

We can then assign a numeric weight to each term in a given document, representing an estimate (usually but not necessarily statistical) of the usefulness of the given term as a descriptor of the given document, i.e., an estimate of its usefulness for distinguishing the given document from other documents in the same collection. It should be stressed that a given term may receive a different weight in each document in which it occurs; a term may be a better descriptor of one document than of another. A term that is not in a given document receives a weight of zero in that document. The weights assigned to the terms in a given document D_1 can then be interpreted as the coordinates of D_1 in the document space; in other words, D_1 is represented as a point in document space. Equivalently, we can interpret D_1 as a vector from the origin of document space to the point defined by D_1's coordinates.

In document space, each document D_1 is defined by the weights of the terms that represent it. Sometimes, it is desirable to define a "term space" for a given collection. In term space, each document is a dimension. Each point (or vector) in term space is a term in the given collection. The coordinates of a given term are the weights assigned to the given term in each document in which it occurs. As before, a term receives a weight of zero for a document in which it does not occur.

We can combine the "document space" and "term space" perspectives by viewing the collection as represented by a document-by-term matrix. Each row of this matrix is a document (in term space). Each column of this matrix is a term (in document space). The element at row *i*, column *j*, is the weight of term *j* in document *i*.

A query may be specified by the user as a set of terms with accompanying numeric weights. Or a query may be specified in natural language. In the latter case, the query can be processed exactly like a document; indeed, the query might *be* a document, e.g., a sample of the kind of document the user wants to retrieve. A natural language query can receive the usual processing, i.e., removal of "stop" words, stemming, etc., transforming it into a set of terms with accompanying weights. (Again, stoplists and stemming are not applicable if the queries and terms are described using n-gram terms.) Hence, the query can always be interpreted as another document in document space. Note: if the query contains terms that are not in the collection, these represent additional dimensions in document space.

An important question is how weights are assigned to terms either in documents or in queries. A variety of weighting schemes have been used. Given a large collection, manual assignment of weights is very expensive. The most successful and widely used scheme for automatic generation of weights is the "term frequency * inverse document frequency" weighting scheme, commonly abbreviated "*tf***idf*". The "term frequency" (*tf*) is the frequency of occurrence of the given term within the given document. Hence, *tf* is a document-specific statistic; it varies from one document to another, attempting to measure the importance of the term within a given document. By contrast, inverse document frequency (*idf*) is a "global" statistic; *idf* characterizes a given term within an entire collection of documents. It is a measure of how widely the term is distributed over the given collection, and hence of how likely the term is to occur within any given document by chance. The *idf* is defined as "*ln* (*N/n*)" where *N* is the number of documents in the collection and *n* is the number of documents that contain the given term. Hence, the fewer the documents containing the given term, the larger the *idf*. If every document in the collection contains the given term, the *idf* is zero. This expresses the commonsense intuition that a term that occurs in every document in a given collection is not likely to be useful for distinguishing relevant from non-relevant documents. Or what is equivalent, a term that occurs in every document in a collection is not likely to be useful for distinguishing documents about one topic from documents about another topic. To cite a commonly-used example, in a collection of documents about computer science or software, the term "computer" is likely to occur in all or most of the documents, so it won't be very good at discriminating documents relevant to a given query from documents that are non-relevant to the given query. (But the same term might be very good at discriminating a document about computer science from documents that are not about computer science in another collection where computer science documents are rare.)

Computing the weight of a given term in a given document as *tf*idf* says that the best descriptors of a given document will be terms that occur a good deal in the given document and very little in other documents. Similarly, a term that occurs a moderate number of times in a moderate proportion of the documents in the given collection will also be a good descriptor. Hence, the terms that are the best document descriptors in a given collection will be terms that occur with moderate frequency in that collection. The lowest weights will be assigned to terms that occur very infre-

quently in *any* document (low-frequency documents), and terms that occur in most or all of the documents (high frequency documents).

6.2 Normalization of Term Vectors

To allow for variation in document size, the weight is usually "normalized". Two kinds of normalization are often applied. [Lee, SIGIR '95] The first is normalization of the term frequency, "*tf*". The *tf* is divided by the "maximum term frequency," tf_{max}. The "maximum term frequency" is the frequency of the term that occurs most frequently in the given document. So the effect of normalizing term frequency is to generate a factor that varies between zero and one. This kind of normalization has been called "maximum normalization" for obvious reasons. A variation is the formula $0.5 + (0.5*(tf/tf_{max}))$ which causes the normalized tf to vary between 0.5 and 1. In this form, the normalization has been called "augmented normalized term frequency". The purpose and effect of term frequency normalization (in either form) is that the weight (the "importance") of a term in a given document should depend on its frequency of occurrence relative to other terms in the same document, not its absolute frequency of occurrence. Weighting a term by absolute frequency would obviously tend to favor longer documents over shorter documents.

However, there is a potential flaw in "maximum normalization." The normalization factor for a given document depends *only* on the frequency of the most frequent term(s) in the document. Consider a document D_1 in which most of the terms occur with frequencies in proportion to their importance in discriminating the document's primary topic. Now suppose that one term has a disproportionately high frequency, e.g., important terms t_1, t_2, and t_3 each occur twice in D_1 but for some stylistic reason equally important term t_4 occurs six times, the maximum for any term in D_1. Then the frequency of t_4 will drag down the weights of terms t_1, t_2, and t_3 by a factor of three in D_1 relative to their weights in some other similar document D_2 in which t_1, t_2, t_3, and t_4 have equal frequencies. (The same problem arises with the "augmented normalized term frequency" but to a less extreme degree since the high frequency term will have a weight of one as with maximum normalization but it cannot drag the weights of the other terms below 0.5.)

A commonly-used alternative to normalizing the term frequency is to take its natural log plus a constant, e.g., "*log* (*tf*) + 1." This technique, called "logarithmic term frequency," doesn't explicitly take document length or maximum term frequency into account but it does "reduce the importance of raw term frequency in those collections with widely varying document length." It also reduces the effect of a term with an unusually high term frequency within a given document. In general, it reduces the effect of *all* variation in term frequency, since for any two term frequencies, tf_1 and $tf_2 > 0$ such that $tf_2 > tf_1$:

$$\frac{\log(tf_2) + 1}{\log(tf_1) + 1} < \frac{tf_2}{tf_1}$$

The second kind of normalization is by vector length. After all of the *tf***idf* term weights for a given document, i.e., all the components of the document vector, have been calculated, every

component of the vector is divided by the Euclidean length of the vector. The Euclidean length of the vector is the square root of the sum of the squares of all its components. Dividing each component by the Euclidean length of the vector is called “cosine” normalization because the normalized vector has unit length and its projection on any axis in document space is the cosine of the angle between the vector and the given axis.

Augmented maximum (term frequency) normalization and cosine normalization can be used separately or together.

Cosine normalization reduces the problem (described above) of vector component weights for a given document being distorted by a single term with unusually high frequency. (But see the discussion below of pivoted unique normalization which further addresses the problem.) The normalization factor (vector length) is a function of *all* the vector components so the effect of a single term with a disproportionately high frequency is diluted by the weights of all the other terms. Furthermore, the normalization factor is a function of each *tf***idf* term weight, not just the *tf* factor of that weight. So, the weight of a high frequency term may also be lessened by its idf factor.

However, as Lee has pointed out, situations exist in which maximum normalization may actually do better than cosine normalization. Consider a case where document D_1 deals with topic T_A and contains a set of terms relevant to T_A. Now consider document D_2 which deals with T_A and also deals with several other topics T_B, T_C, etc. Suppose that D_2 contains all the terms that D_1 contains, i.e., terms relevant to T_A, but also contains many other terms relevant to T_B, T_C, etc. Since cosine normalization of a given document takes into account the weights of all its terms, the effect is that the weights of the terms relevant to T_A will be dragged down in D_2 (relative to the weights of the same terms in D_1) by the weights of the terms relevant to T_B, T_C, etc. As a result, a user trying to retrieve documents relevant to T_A will be much more likely to retrieve D_1 than D_2 even if they both cover T_A to the same extent. Maximum normalization will do better in this case provided that the maximum frequency term relevant to T_A in D_2 is about as frequent as the maximum frequency term in D_2 relevant to any of the other topics. In that case, none of the other topics will drag down the weights of T_A’s terms in D_2. Lee concludes that in some cases, better precision and recall can be achieved by using each normalization scheme for retrieval separately and then merging the results of the two retrieval runs. (Merging retrieval runs is discussed further below.)

Cosine normalization, as noted above by Lee, tends to favor short documents over long ones, especially in the case where the short document is about a single topic relevant to a given query, and the longer document is about multiple topics of which only one is relevant to the given query. Singhal et al. [SIGIR ‘96] have investigated this problem, and produced a new weighting scheme to correct the problem. They studied fifty queries applied to a large document collection (741,856 documents); queries and documents were taken from the TREC 3 competition. Their study compared probability of retrieval to probability of relevance as functions of document length. The study confirmed the expectation that short documents were more likely to be retrieved than their probability of relevance warranted, while longer documents were less likely to be retrieved than their probability of relevance warranted. This pattern was found to apply to query sets that retrieved relevant documents from six diverse sub-collections of the TREC collection. A natural consequence is that for any collection and query set to which the pattern applies, there will be a

"crossover" document length for which the two probability curves intersect, i.e., a document length for which the probability of relevance equals the probability of retrieval.

These observations led Singhal et al. to develop a "correction factor", a function of document length that maps a conventional "old" document length normalization function, e.g., cosine normalization, into a "new" document normalization function. The correction factor rotates the old normalization function clockwise around the crossover point so that normalization values corresponding to document lengths below the crossover point are greater than before (so that the probability of retrieval for these documents is decreased), and normalization values corresponding to document lengths above the crossover point are less than before (so that the probability of retrieval for these documents is increased). (Remember that term weights for a given document are *divided* by the normalization factor.) The crossover point is called the "pivot". Hence, the new normalization function is called "pivoted normalization."

Note that since the pivoted normalization method described below is based on correcting the document normalization so that the distribution of probability of retrieval coincides more closely with the probability of relevance (as a function of document length), this weighting method could legitimately be called "probabilistic". However, it differs from the probabilistic methods discussed below in section 7, because the probability distributions have been determined experimentally, by observing actual TREC collections, rather than being derived from a theoretical model.

The pivoted normalization is easily derived. Before the normalization is corrected, i.e., pivoted, the relation between *new normalization* and *old normalization* is:

$$new\ normalization = old\ normalization$$

This is a straight line with slope one through the origin of a graph, with a *new normalization* vertical axis, and an *old normalization* horizontal axis. This line is rotated clockwise around the pivot, i.e., around the normalization value corresponding to the crossover document length. Call this value "*pivot*." After the rotation, the form of the new line (by elementary analytic geometry) is:

$$new\ normalization = slope \bullet old\ normalization + K$$

where the slope of the new line is less than one and K is a constant. (Note that although the *old normalization* function, e.g., cosine normalization, is not a linear function of term weights, the *new normalization is* a linear function of the *old normalization.*) K is evaluated by recognizing that since the line was rotated around the pivot point, *new normalization* equals *old normalization* at the pivot point. Hence, substitute *pivot* for both *new normalization* and *old normalization* in the

above linear equation, solve for K and then substitute this value of K back into the equation. The result (with *new normalization* now called *pivoted normalization*) is:

$$pivoted\ normalization = slope \bullet old\ normalization + (1.0 - slope) \bullet pivot$$

where *slope* and *pivot* are constant parameters for a given collection and query set. Since the ranking of documents in a given collection for a given query set is not affected if the normalization factor for every document is multiplied (or divided) by the same constant, these two parameters can be reduced to one by dividing the above normalization function by the constant *(1.0-slope)*pivot*. Singhal et al. found that the optimum value of this parameter was surprisingly close to constant across a variety of TREC sub-collections. Hence, an optimum parameter value learned from training experiments on one collection could be used to compute normalization factors for other collections.

Singhal et al. also examined closely the role of term frequencies and term frequency normalization in term weighting schemes. First, they found (by studying the above experiments) that though, as noted above, cosine normalization favors short documents over long ones, it also favors extremely long documents. This phenomenon is magnified by pivoted normalization. Further, they noted that term frequency is not an important factor in either cosine normalization or document retrieval. This is because (1) the majority of terms in a document only occur once, and (2) *log(tf) + 1* is commonly used in place of raw term frequency, which has a "dampening effect" for *tf > 1*. Hence, the cosine normalization factor for a given document will be approximately equal to the square root of the number of unique terms in the given document, i.e., it increases less than linearly with number of unique terms. But document retrieval is generally governed by the number of term matches between document and query, and hence making the usual simplifying assumption that occurrence of a given term in a given document is independent of the occurrence of any other term, the probability of a match between query and document varies linearly with the number of unique terms. The purpose of the document normalization is to adjust the term weights for each document so that the probability of retrieving a long document with a given query is the same as the probability of retrieving a short document. The conclusion is that cosine normalization reduces term weights by too little for very large documents. Singhal et al. propose to remedy this situation by replacing the cosine normalization value, i.e., the *old normalization*, by *# of unique terms,* in the pivoted normalization function.

Singhal et al. argue further that maximum normalization, i.e., tf/tf_{max}, is not the optimum method of normalizing term frequency because what matters is the frequency of the term relative to the frequencies of all the other descriptor terms, not just relative to the frequency of the most frequently occurring term. Hence, they propose using the function:

$$\frac{1 + \log(tf)}{1 + \log(average\ tf)}$$

for normalized term frequency. This function has the property that its value is one for a term whose frequency is average for the given document, greater than one for terms whose frequencies are greater than average, and less than one for terms whose frequency is less than average.

Hence, Singhal et al. propose weighting each term in a document by the above term frequency normalization function divided by the pivoted normalization, i.e.,

$$\frac{\dfrac{1 + \log(tf)}{1 + \log(average\ tf)}}{slope \bullet \#\ of\ unique\ terms + (1.0 - slope) \bullet pivot}$$

Note that the *idf* is absent from this weighting function. This is because, for reasons explained in the next section, the idf is normally used as a factor in the weights of query terms rather than document terms. That is, if a given term occurs in a query and also in some documents in the collection being queried, the *idf* will be used as a factor in the weight of that term in the query vector rather than in the corresponding document vectors.

Singhal et al. tested this improved term weighting function against a set of TREC sub-collections and found that for optimum parameter values, it performed substantially better than the more familiar product of tf/tf_{max} and *1/cosine normalization*.

It should be noted that this improved weighting scheme compensates for both of the problems noted by Lee. The effect of a term with a disproportionately high frequency in a given document is greatly reduced by the new term frequency normalization function, partly because the frequency of the given term is divided by the average term frequency rather than the maximum term frequency, and partly because both the given term frequency and the average term frequency are replaced by their logs. The advantage of a short document dealing entirely with a topic relevant to a given query, over a longer document dealing with the relevant topic and several nonrelevant topics, is compensated by pivoted normalization which reduces the probability of retrieval of short documents and increases the probability of retrieval of long documents.

All of the normalization schemes discussed above (and in the following section) are based on one underlying assumption: that document relevance is independent of document length. Relevance is assumed to be wholly about how much a given document is *about* a given topic. Variation of document length is viewed as a complication in computing document relevance. Hence, all of the normalization schemes are aimed at factoring out the effects of document length.

If document D_2 is longer than document D_1, relevance computation is assumed to be distorted in one of two ways. If D_2 is largely or entirely about the same topic T_i as D_1, then relevance is distorted by the fact that terms characteristic of the given topic will tend to occur with greater frequency in D_2. If *D2* is about a number of different topics, T_i, T_j, T_k, etc., and only a small part of D_2 is about the same topic T_i as D_1, then the material about the other topics dilutes the effect of the relevant material, making D_2 seem less relevant to T_1 than D_1, even though both documents may contain the same information about T_i. Normalization is largely aimed at overcoming these two kinds of distortion.

Completely ignored is a third possibility: that the user actually prefers either short or long documents about T_i. If the user prefers the longer document, D_2, e.g., she needs all the details it provides, then D_2 is more relevant to T_i relative the users needs. Or to use a term that may be more appropriate, D_2 may be more *pertinent* or more *useful* to the user in meeting her present information need, as expressed by T_i. Of course, D_1 may be more useful; perhaps the user needs a concise summary of the main facts or ideas about T_i, and has neither time nor need for a more detailed exposition. In either case, document size is an important parameter in computing the documents relevance for purposes of selection and ranking in the retrieval set returned to the user. This indicates that either the document and topic vectors should not be normalized, or that the document size should enter explicitly into the document topic similarity computation. This issue is discussed further in section 6.4, which discusses document-topic similarity functions.

6.3 Classification of Term Vector Weighting Schemes

Since the various alternatives discussed above for computing and normalizing term weights can be (and have been) used in a variety of combinations, a conventional code scheme (associated with a popular IR research engine called the SMART system) has been defined and widely adopted to classify the alternatives. [Salton, IP&M, 1988] [Lee, SIGIR ‘95] See Table 1.

The weight of a given term is specified as the product of a *term frequency* factor, a *document frequency* factor, and a *document length normalization* factor. For each of these three factors, two or more alternatives are available. Each alternative for each factor is given a code. See table 1. The codes for the term frequency factor are: “*b*” (term frequency is ignored; the term frequency factor is one if the term is present in the given document, zero otherwise), “*n*” (use the *raw term frequency,* the number of times the term occurs in the given document), “*a*” (use the “augmented normalized term frequency” as defined in the previous section), “*l*” (use the “*logarithmic term frequency*” as defined in the previous section), and “*L*” (use “*average term frequency* based normalization” as defined in the previous section). (A code for the pure “maximum normalization” does not appear to have been defined.) The codes for the document frequency factor are; “*n*” (use 1.0, document frequency factor is ignored), “*t*” (use “*idf*” as the document frequency factor). The codes for the normalization factor are: “*n*” (use 1.0; no document length normalization is used), “*c*” (use cosine normalization, i.e., 1/(*Euclidean vector length*), and “*u*” (use “pivoted unique normalization” as discussed in the previous section). A weighting scheme is constructed in “Chinese menu” form: one from column A (*term frequency* factor), one from column B (*document frequency* factor), and one from column C (*document normalizatio*n factor). For example, “*lnc*” means, “Compute the weight of each term in a given document as the product of the *logarithmic term frequency* (*l*) of the given term, 1.0 (ignore the *idf* of the term), and the *cosine normalization* factor (*c*) of the document vector for which the term’s weight is being computed.” (Multiplying by the cosine normalization factor is equivalent to dividing by the Euclidean vector length as defined in above.) As a further refinement, it is common to use a different weighting scheme for the query than for the documents in the collection being queried. Therefore, the complete specification of the weighting scheme involves two triples, e.g., *lnc-ltc* describes a scheme where the document vectors are weighted as above, and the query vectors are weighted the same except that each query term weight is also multiplied by the idf of the given term in the collection to which the query is being applied. (Note that the query is weighted as a “document” so that the *term frequency* factor

Table 1: Components of schemes for weighting given term in given document

Term Frequency within Document (Unnormalized or Normalized)		
Code	*Formula for Component*	*Description of Component*
b	1.0	Term frequency = 1 if term is in given document, = 0 if term is not in given document.
n	tf	"Raw" term frequency, i.e., number of occurrences of term in given document.
a	$0.5 + 0.5 \cdot \frac{tf}{maxtf}$	"Augmented" term frequency. First, term frequency of given term is normalized by frequency of most frequent term in document ("maximum" normalization) to allow for importance of term relative to other terms in document. Then, it is further normalized ("augmented") so resulting value is in range from 0.5 to 1.0.
l	$ln\ tf + 1.0$	Logarithmic term frequency. This reduces importance of raw term frequency, e.g., if t_2 has twice the frequency of t_1 in given document, the ratio of the logs will be much smaller.
L	$\frac{1 + \log(tf)}{1 + \log(average\ tf)}$	Average term frequency based normalization. See discussion in previous section.
Document Frequency (Number containing Term) within Collection		
n	1.0	Number of documents containing given term is ignored. Original term frequency is not modified.
t	$\ln\frac{N}{n}$	Original term frequency is multiplied by inverse document frequency (idf) where N is the total number of documents in the collection, and n is the number of documents containing the given term. Hence, term that occurs in many documents counts for less than term that occurs in few (or one).

Table 1: Components of schemes for weighting given term in given document (Continued)

Document Length Normalization Component		
n	1.0	Variation in document length is ignored.
c	$\frac{1}{\sqrt{\sum_i w_i^2}}$	Weight of given term in given document is normalized by the length of the document's term vector, so that long documents are not favored over short documents.
u	$\frac{1}{(slope \bullet \#\ of\ \ unique\ terms) + (1 - slope) \bullet pivot}$	Pivoted normalization. See previous section.

measures term frequency within the query, and the *document length* factor normalizes for query length. Only the *idf* factor in the weight of a query term is a measure of the distribution in the collection being queried, not in the query itself.) This scheme has exhibited high retrieval effectiveness for the Text REtrieval Conference (TREC) query sets and collections. [Lee, SIGIR '95] However, *Lnu-ltc* weighting exhibited even better effectiveness against TREC3 and TREC 4 query sets and collections. [Singhal et al., SIGIR '96] [Buckley et al., TREC 4]

The weighting scheme classification described above is open-ended. Indeed, the SMART team, originators of this classification, have only recently added the *L* option to the term frequency factor, and the *u* option to the document length normalization factor.

Note that although the *idf* of a given term is a statistic that characterizes that term relative to a given collection of documents, not relative to a query, it is common to use the *idf* to weight the occurrence of the given term in queries being applied to the collection, not to weight its occurrence in the document vectors that describe the collection itself. The *lnc-ltc* and *Lnu-Ltu* weighting schemes are examples. There are simple reasons for this. First, it is more efficient for purposes of collection maintenance. Whenever new documents are added to the collection (or old documents are removed), the idf must be recomputed for each descriptor term in the affected documents. It would be inefficient to recompute the weight of such a term in every document in which it occurs. Moreover, it is unnecessary for the purposes of a query/document similarity calculation, since the document ranking produced for a given query will be exactly the same whether the idf's enter the computation as factors in the query term weights, or factors in the document term weights, or both.

In a weighting scheme like *tf*idf*, the normalized term frequency of a given term in a given document is *multiplied* by its *idf* so that "good" descriptor terms (which characterize only a relatively small number of documents in a given collection) are weighted more heavily than "bad" descriptor terms (which are so common that they occur in a great many documents in the given collection, and hence are of little value in discriminating between relevant and non-relevant documents).

An alternative approach is to *subtract* from the normalized term frequency of the given term the "average" normalized frequency of the term averaged over all the documents in the given collection. Here, "average" may be "mean", "median", "or some other measure of commonality". [Damashek, Science, 1995] (This is equivalent to subtracting from each document vector a "centroid" vector, i.e., a vector that is the average of all the document vectors in the collection.) Note that a term that occurs in a large proportion of the documents in the given collection will have a larger average term frequency than a term that occurs in only a few documents. Hence, the effect of subtracting the average is to reduce the weight of commonly used terms by more than the weight of rarer terms. The centroid is a measure of commonality, of terms that are too widely used to be good document descriptors.

6.4 Computation of Similarity between Document and Query

Once vectors have been computed for the query and for each document in the given collection, e.g., using a weighting scheme like those described above, the next step is to compute a numeric "similarity" between the query and each document. The documents can then be ranked according to how similar they are to the query, i.e., the highest ranking document is the document most similar to the query, etc. While it would be too much to hope that ranking by similarity in document vector space would correspond exactly with human judgment of degree of relevance to the given query, the hope (borne out to some degree in practice) is that the documents with high similarity will include a high proportion of the relevant documents, and that the documents with very low similarity will include very few relevant documents. (Of course, this assumes that the given collection contains some relevant documents, an assumption that holds in TREC experiments but which can't be guaranteed in all practical situations.) Ranking of course, allows the human user to restrict his attention to a set of documents of manageable size, e.g., the top 20 documents, etc.

The usual similarity measure employed in document vector space is the "inner product" between the query vector and a given document vector. [Salton, 1983] [Salton, 1989] The inner product between a query vector and a document vector is computed by multiplying the query vector component (i.e., weight), QT_i for each term i, by the corresponding document vector component weight, DT_i for the same term i, and summing these products over all i. Hence the inner product is given by:

$$\sum_{i=1}^{N} QT_i \cdot DT_i$$

where N is the number of descriptor terms common to the query and the given document. If both vectors have been cosine normalized, then this inner product represents the cosine of the angle between the two vectors; hence this similarity measure is often called "cosine similarity." The maximum similarity is one, corresponding to the query and document vectors being identical (angle between them zero). The minimum similarity is zero corresponding to the two vectors having no terms in common (angle between them is 90 degrees).

One problem with cosine similarity, noted by both Salton and Lee and discussed above, is that it tends to produce relatively low similarity values for long documents, especially (as Lee points

out) when the document is long because it deals with multiple topics. But Lee's solution (mentioned above and discussed in more detail in section 6.2) is *not* to use a more complicated similarity measure in place of cosine similarity, but rather to merge the result of a retrieval using cosine similarity with the result of a retrieval using term frequency normalization, e.g., maximum normalization. In other words, Lee supplements cosine similarity rather than replaces it, thereby getting the advantages of two relatively simple similarity measures. And the solution of Singhal et al., discussed above in section 6.2, is to develop improved normalization factors for term weighting, factors that do a better job of normalizing for document length and term frequency during a *single* retrieval run, thereby eliminating the need for fusion of separate runs.

In an earlier approach from the same research group, Salton and Buckley [TREC 2, 1994] dealt with the problem of long documents by combining the usual cosine similarity of query and document ("global" similarity) with similarity of the query to parts of the document ("local" similarity). The parts they tried included sentences and paragraphs. In other words, if two documents D_1 and D_2 have comparable similarity to a given query, but D_1 also contains a sentence or paragraph that is particularly similar to the query, then D_1 will be given a higher similarity value than D_2. They have also tried combining multiple local similarity measures, e.g., sentence and paragraph similarity, with the global similarity. However, the Singhal et al. enhanced term weighting (*Lnu*) method, by improving document length normalization, and term frequency normalization, reduces the importance of such local similarity measures. Buckley et al. [TREC 4] report that using *Lnu* weighting reduces the improvement gained by their local/global similarity measures from 15% to 3%. They continue to experiment with sophisticated similarity measures that take into account the actual location of every descriptor term. However, their motivation is now less to correct for document length bias, and more to create a similarity measure to which non-statistical, e.g., linguistic and certainty, factors at the term level can be added.

Another approach to the problem of large, or multi-topic, documents is to break each large document into sections, commonly called "passages," and treat each passage as a "document." [Callan, SIGIR '94] In other words, the system computes the similarity between each passage and the user's query. This enables the system to determine the "best" passage, i.e., the passage in a given document most similar (and hence hopefully most relevant) to the query. This passage then comes to represent the document in any further computations or retrievals involving the same query or any similar information need. Passages have been calculated in a number of ways: *sections* as determined by document markup [Wilkinson, SIGIR '94], passages as delimited by the author, e.g., sections, paragraphs, sentences, etc. [Salton et al., SIGIR '93], clusters of paragraphs, fixed-size passages, etc. A document may be partitioned into fixed size disjoint passages. Fixed length passages may be a fixed number of terms, e.g., 50 words to 600 words each [Kaszkiel et al., SIGIR '97], with each passage beginning immediately after the preceding passage. Zobel et al. [IP&M, 1995] employ passages of 1000 to 2000 words, which they call *pages*; the passages are generated by accumulating paragraphs until the desired length is reached, so that the passages are not exactly fixed length, but always end on a paragraph boundary. Alternatively, fixed-size passages may be overlapped, e.g., if the passages are each p terms long, each passage may start $p/2$ terms beyond the start of the preceding passage. [Callan, SIGIR '94] These fixed size, overlapping passages have been called *windows*, because they can be viewed as obtained by sliding a window p terms long over the text. "[E]xperiments have shown that fixed size passages are at least as effective — and marginally more efficient — than their varying counterparts." [Allan, SIGIR '95]

(The superiority of fixed length, overlapping and non-overlapping, passages is also supported by the experiments of Kaszkiel et al. [SIGIR '97] discussed below.) These passage approaches differs from the Salton and Buckley "global/local approach" in that only local similarities between a query and the passages of a document are computed. Documents are ranked according to the best local similarity in the given document for the given query, i.e., the passage in a each document that is most similar to the given query. Presumably, the benefit of this approach too is reduced by incorporation of an improved document normalization scheme such as *Lnu*. (However, Kaszkiel et al. found that the pivoted document length normalization improved passage length retrieval for all those cases where the passages varied substantially in length.)

Hearst et al. {SIGIR '93] approach global/local similarity in a novel way. First, they attempt to break a given document into *motivated* segments, i.e., variable length segments such that the boundary between one segment and the next is the boundary between one subtopic and the next. These segments, called "tiles," may be multi-paragraph units.

The "tiling" method begins by partitioning the document into "blocks," such that each block is k sentences long. As a heuristic, Hearst chooses k to be the average paragraph length in sentences. Note that this means that the length of a block in sentences may vary somewhat from one document to the next. Moreover, since the block size is a fixed number of sentences, blocks will vary somewhat in size (measured in terms) even within a given document, although they will be approximately equal.

The blocks are combined into topic-based segments, on the assumption that two consecutive blocks are likely to be about the same topic if they are statistically similar, and likely to be about different topics if they are not similar. Similarity is calculated for every pair of consecutive blocks in a given document. The similarity measure employed is the standard cosine similarity, with terms weighted according to the standard *tf***idf* formula. The novelty in employing this formula is that the blocks are treated as the "documents," while the document is treated as the document "collection." Hence, in computing the weight of term t_i in block B_j of document D_k, tf_i is the unnormalized frequency of term t_i in B_j, while idf_i is the *idf* of t_i in the collection of blocks comprising D_k, i.e., it will be zero if it is present in every block of D_k, much higher if it is only present in one or two blocks of D_k. The effect is that two consecutive blocks B_j and B_{j+1} will be very similar if they share a number of terms, the terms are relatively frequent in B_j and B_{j+1}, and are found in few other blocks of D_k. By contrast, terms that are shared by B_j and B_{j+1} but are spread fairly uniformly throughout the entire document, will contribute much less to the inter-block similarity.

These consecutive block similarities are then graphed against block position in the document. The result is a function that peaks where consecutive blocks are very similar, and drops where they are very dissimilar. After this function is smoothed to eliminate small local fluctuations, the result is a function with peaks and valleys. The low points of the valleys are then taken to be the boundaries of the tiles, segments that are each assumed to be about a single local subtopic.

These tiles can be used in a number of ways. Hearst et al. propose a two stage query, where the user can request documents about topic T_1, matching T_1 against an entire document, and about subtopic T_2, matching T_2 against each block in document D_k that satisfies T_1. Note that this is different than combining a local and global similarity into a single similarity measure as Salton and

Buckley do. In the proposed Hearst approach, a document must separately satisfy *both* the global similarity to T_1 *and* the local similarity to T_2. Neither topic will unduly dilute the calculation of similarity for the other topic. As a further enhancement, terms associated with the global or "background" topic T_1 could be eliminated when searching tiles for T_2.

Tiles have also been used by Hearst et al. in a single stage query mode. Here, the idea is that each document in a collection is segmented into tiles, and each tile is then term indexed as a separate "virtual" document. The index for a given tile also identifies the actual document from which it came. Given a query, *Q*, the best (most similar) *N* tiles (*N* = 200 in the reported experiment) are retrieved, and grouped by document of origin. Then, the similarity scores of the tiles in each group, i.e., originating in the same document, are summed. The actual documents are then ranked according to these sums. In the reported experiment using TREC data, retrieval using these tile similarity sums produced substantially better precision than querying directly on indexes of the full text (actual) documents. Apparently, the tile method favored those portions of a document that were most relevant to a given query.

Kretser et al. [SIGIR '99] carry the concept of locality-based similarity even farther than Hearst does with her tiles. They view a document collection as a sequence of words. They calculate a "score" for each word position in the sequence. Hence, the position of a word is its position relative to the start of the collection, *not* relative to the document in which it occurs. Given a query Q defined as a set of terms (words) *t*, the Kretser system computes a score $C_Q(x)$ for every position *x* in the collection. The score for a given word position *x* depends on whether a query term *t* occurs at that position, and on whether query terms occur within a certain distance of that position. The farther a query term occurs from the word position *x* being evaluated, the less its influence, the smaller the "contribution" it makes to $C_Q(x)$. The score $C_Q(x)$ at word position *x* is the sum of the contributions $c_t(x,l)$ where *t* is any query term, *l* is any word position in the document collection at which *t* occurs, and $c_t(x,l)$ is the contribution that query term *t* at word position *l* makes to the score at word position *x*.

Kretser et al. tried out four different shapes for the contribution function $c_t(x,l)$: triangle, cosine, circle, and arc. However, in all four cases, $c_t(x,l)$ was a function of (1) $d = |x-l|$, the maximum distance from *x* at which a query term can make a contribution to the score at *x*, (2) h_t, the "height" or peak contribution, which is the contribution made by a query term *t* at position *x*, and (3) s_t, the "spread" of the contribution function for query term *t*, which determines the maximum distance, d_{max}, at which *t* can contribute to the score at word position *x*. Note that the height and spread are both functions of *t* itself. A scarce term t will have a greater spread; in other words, an occurrence of *t* is able to influence the score at *x* at a greater distance from *x* than an occurrence of a common term. Similarly, a scarce term *t* contributes more proportionately than a common term to the score at *x* if it occurs at *x*, or if it occurs at any distance from *x* within its spread. It is noteworthy that both the spread and the height are collection-based, *not* document-based functions. That is, "scarcity" means scarcity within the collection, not scarcity within any given document. For example, the "spread" is defined as n/f_t where *n* is the number of unique words in the collection, and f_t is the frequency of *t*, i.e., the number of times term *t* occurs in the collection. However, document boundaries *do* play one role in the computation of the score at word position *x*. The spread is not allowed to cross a document boundary. Hence, the score at a word position near the beginning or end of a document can not be influenced by a query term in an adjacent document.

The scheme described above generates a similarity score for every position *x* in the document collection relative to a given query *Q*. The similarities must be computed at runtime, when the query *Q* is supplied. However, as with conventional document-based systems, the collection is indexed in advance as an inverted list, to facilitate the runtime similarity calculation. But Kretser's index must specify *word positions* for every term in the collection, not just document occurrences as for conventional IR systems. Moreover, since it is intended that the query engine will return a ranked list of documents, the document boundaries must also be stored in the index. Various strategies are used to compress all of this information efficiently. The compression ratio varies with the collection. For the four collections tested by Kretser et al., the ratio of index size to collection size varied from 18.9% to 23.1%. This compares with 2.6% to 6.8% for the corresponding document level indexes.

At runtime, given a query *Q*, the system computes the similarity score of every word position relative to *Q*. To rank documents, the system finds the highest scoring word position, and adds its score to an (initially zero) accumulator for the document in which it occurs. Then, it repeats the process for the next highest scoring word position. This 2nd word position may be in the same document, in which case its score is added to the same accumulator. Or, it may occur in another document, in which case, it is added to a new accumulator. The process continues until *r* non-zero accumulators have been generated, where *r* is the number of documents to be returned to the user. The documents are presented to the user in the order of the scores in their accumulators.

Kaszkiel et al. [SIGIR '97] performed experiments to compare various approaches to passages, including Hearst's *tiles*, Zobel's *pages*, Callan's fixed-length overlapping *windows*, non-overlapping fixed length passages, Wilkinson's *sections*, and paragraphs. Twelve different fixed length passage sizes from 50 words to 600 words were tried. The results indicated that fixed length passages, both overlapping and non-overlapping, of 150 words or more were "simple; highly effective; robust" for document retrieval. They scored as well or better than both whole document retrieval, and other passage methods, over the full range of experiments, e.g., for different data sets, and different document normalization schemes. Several limitations of these experiments should be noted, however. Kaszkiel et al. note a couple of limitations themselves: First, their results do not rule out the possibility of further improvement through "combination of passage-level and document-level evidence" Second, they did not exploit (as did some of the passage methods they cite, e.g., tiling) the possibility of achieving better document ranking by combining multiple passage similarities for each document. There are other limitations that they do not mention. Their study focused on using passage similarities to rank the whole documents in which the passages occur. They do not study passages as units of retrieval themselves; indeed, it would be difficult for them to do so, since they employ TREC data for training and testing; TREC data only provides relevance judgments for whole documents, not for passages (let alone the diverse kinds of passages compared here). Similarly, they do not study the interactive uses of passages, e.g., the presentation high-ranking passages to the use, as a basis for selecting either documents, or other similar passages.

Most of the vector space similarity formulas discussed above assume that the vectors have been normalized, with the intended effect that document length is factored out. In other words, the intention is that a short document about topic T_i, a long document containing a short passage

about T_i, and a long document entirely about T_i will yield the same similarity to the topic statement T_i itself. But as was noted in section 6.2, often the user has a preference for either longer or shorter documents about the topic of interest. This problem has not usually been addressed in IR research, but a number of possibilities are available.

One alternative, mentioned back in 6.2, is not to normalize the vectors at all. The effect of taking the inner product of unnormalized vectors (no term frequency normalizations, no vector length normalization) is to generate similarity values that take into account both relevance and size. A short document D_1 relevant to T_i will receive a larger similarity value than a document D_0 totally non-relevant, a long document D_3 containing a short passage relevant to T_i will receive a similarity value roughly equal to that received by D_2, and a long document D_4 largely or entirely about T_i will receive the largest similarity value of all. This works well if the user prefers longer, more detailed documents like D_4. But what if the user prefers shorter documents? Simple functions are available for inverting the similarity function just described so that the similarity values of the inverted function decrease as the similarity values of the original function increase. But this has the perverse effect of giving the highest similarity values to documents that are non-relevant!

Another alternative is to make size a separate and distinct parameter. Size can be made a separate component of both the topic description vector and each document vector. Hence, a match on size category, e.g., small, medium, large, could increase the relevance by an amount that depended on the term weights assigned to that component. Or size could be taken out of the vector space altogether, e.g., the algorithm could test the user's size preference first, and then perform a different vector inner product computation based on the user's preference. The vectors would be unnormalized if the user preferred large documents, normalized (especially pivoted normalization) if the user wanted equal preference to be given to documents about the given topic independently of size. Cosine normalization or some other normalization scheme specifically designed to favor short relevant documents over long relevant documents would be applied to the vectors if the user preference were for "short and sweet."

The inner product and its normalized form, cosine similarity, are not the only similarity functions employed to compare a document vector with a topic vector (although they are by far the most widely used). A variety of "distance" functions, and other term matching functions are available. For example, a family of distance metrics [Korfhage, 1997] is given by:

$$L_p(D_1, D_2) = [\sum_i |d_{1i} - d_{2i}|^p]^{1/p}$$

These metrics compute the distance in vector space between vectors D_1 and D_2 in terms of the components d_{1i} of D_1, the components d_{2i} of D_2, and a parameter p that determines which chooses a specific metric from the family. If p=1, the metric is the city block distance, i.e., distance measured as number of city blocks from one street intersection (corner) to another in a city where the streets are laid out as a rectangular grid. If p=2, the metric is the familiar Euclidean distance, i.e., the straight line distance in the vector space. (This is the same metric that is used for computing the length of a vector for purposes of Euclidean normalization.) If p=∞, the metric is the *maximal direction distance*. That is, as p tends to infinity, the largest difference $|d_{1i} - d_{2i}|$ tends

to dominate all the others, and the function reduces to the absolute value of this maximum difference. Since each vector component corresponds to one dimension, one direction, in vector space, each difference between a pair of corresponding components is the distance between the vectors in a given direction. The maximal direction distance metric is the distance along the dimension where the vectors are farthest apart.

Apart from such distance metrics, there are a host of similarity formulas that "normalize" by avoiding term frequencies altogether, i.e., functions that only count the number of terms that match and (sometimes) the number of terms that don't match. One such popular function is Dice's coefficient [van Rijsbergen, 1979]:

$$Dice = \frac{2w}{n1 + n2}$$

where w is the number of terms common to vectors D_1 and D_2, n_1 is the number of non-zero terms in D_1, and n_2 is the number of non-zero terms in D_2. Note that the denominator here performs a kind of normalization, so that a short document D_1 will get a high score relative to a short topic description D_2 to which it is relevant. A long document D_3 relevant to D_2 will get a lower Dice score provided that the additional text in D_3 contains terms that are not in D_1 and also not in the topic description (greater n_1, same w). This could happen if D_3 contains long sections not relevant to D_2, It could also happen if D_3 contains additional discussion of the topic described by D_2, but this additional discussion uses terms that were overlooked by the user who specified topic D_2. On the other hand, if D_3 and D_1 contain most of the same topic-relevant terms that D_2 contains, but D_3 just uses them more frequently and uses few additional terms that D_1 doesn't use, then D_3 and D_1 will receive similar Dice scores despite their difference in length.

Another common similarity function is Jaccard's coefficient [van Rijsbergen, 1979]:

$$Jaccard(D_1, D_2) = \frac{w}{N - z}$$

where w (as before) is the number of terms common to vectors D_1 and D_2, N is the total number of distinct terms (not term occurrences!) in the vector space (union of *all* document and topic vectors), and z is the number of distinct terms (not term occurrences!) that are neither in D_1 nor in D_2. In other words, N-z is the total number of distinct terms that occur in D_1 or D_2 or both. Note that the value of the Jaccard function is lower, the more distinct terms are either in D_1 but not D_2 or vice versa. It doesn't matter whether the mismatch is caused by non-relevance, or difference in document length. On the other hand, it doesn't matter how frequently a mismatching term (or a matching term) occurs in either D_1 or D_2.

The above schemes for computing query/document similarity assume the existence of a (relatively) static collection of documents to which each query formulated by a user is applied. At the other extreme, we have the "routing" case in which documents arrive in a constantly changing incoming stream, and each document must be "routed" to one of N boxes corresponding to N pre-

selected topics. In sharp contrast to the collection retrieval case, the "queries," i.e., topics or information needs, are fixed while the supply of documents is very dynamic. How can the vector space approach (or any statistical approach) be applied to a situation in which there is no fixed document collection for which collection-wide statistics such as *idf* can be computed? The usual answer is to provide a "training set" of (one hopes) "typical" documents for which statistics can be calculated. Obviously, the hope is that all the subsequent documents received by one's system will have the same statistical properties as the training set. (The alternative is to update the training set regularly, which of course requires retraining the system regularly too.)

6.5 Latent Semantic Indexing (LSI) — An Alternative Vector Scheme

In the traditional vector space approach to IR described above, a vector "space" is defined for a collection of documents such that each dimension of the space is a term occurring in the collection, and each document is specified as a vector with a coordinate for each term occurring in the given document. The value of each coordinate is a weight assigned to the corresponding term, a weight intended to be a measure of how important the given term is in characterizing the given document and distinguishing it from the other documents in the given collection. This approach is an effective first approximation to the statistical properties of the collection, but it is nevertheless an oversimplification. Its major limitation is that it assumes that the terms are independent, orthogonal dimensions of the document space. Adding a new term to the space, e.g., a term that was previously omitted because it wasn't considered a good discriminator, has no effect whatever on the existing terms defining the space. (Adding a new *document* to the collection not only adds new terms to the space but also does affect the weights of the existing terms because it affects their *idf*'s. But this is a term-document relationship, not a term-term relationship.) Hence, relationships among the terms, e.g., the fact that certain terms are likely to co-occur in documents about a given topic because they all refer to aspects of that topic, are ignored. Similarly (and more subtly), the traditional term vector approach ignores the fact that term A and term B may occur in similar contexts in two distinct documents because they are synonyms.

The traditional vector space approach has another feature that can be a drawback in some applications: Since the number of terms that occur in a collection can be large (even after "noise" words have been deleted with a stop list, and variant forms of the same word have been eliminated by stemming), the traditional term-based document space has a large number of dimensions.

A new vector space approach called Latent Semantic Indexing (LSI) [Deerwester et al., JASIS, 1990] attempts to capture these term-term statistical relationships. In LSI, the document space in which each dimension is an actual term occurring in the collection is replaced by (recalculated as) a much lower dimensional document space called k-space (or LSI space) in which each dimension is a derived concept, a "conceptual index," called an LSI "factor" or "feature." These LSI factors are truly independent statistically, i.e., uncorrelated, in a way that terms are not. Hence, LSI factors are "information rich" [SIGIR '94, Hull] in the sense that they capture the term-term relationships that ordinary term-based document space does not. Documents are represented by LSI factor vectors in k-space just as they are represented by term vectors in traditional term-based document space. Vector similarity can be calculated in the same way in k-space as in traditional document space. However, documents and queries dealing with the same topic that would be far

apart in traditional document space (e.g., because they use different but synonymous terms) may be close together in k-space.

As Bartell, et al. [SIGIR '92] explain (they speak of "keywords" rather than "terms"):

> [I]ndividual keywords are not adequate discriminators of semantic content. Rather the indexing relationship between word and document is many-to-many: A number of concepts can be indexed by a single term [polysemy], and a number of terms can index a single concept [synonymy] … [Hence] some relevant documents are missed (they are not indexed by the keywords used in the query, but by synonyms) and some irrelevant documents are retrieved (they are indexed by unintended senses of the keywords in the query). LSI aim[s] at addressing these limitations. This technique maps each document from a vector space representation based on keyword frequency to a vector in a lower dimensional space. Terms are also mapped into vectors in the reduced space. The claim is that the similarity between vectors in the reduced space … may be a better retrieval indicator than similarity measured in the original term space. This is primarily because, in the reduced space, two related documents may be represented similarly even though they do not share any keywords. This may occur, for example, if the keywords used in each of the documents co-occur frequently in other documents.

In other words, if document D_1 uses term t_A and document D_2 uses equivalent term t_B, LSI will effectively recognize this equivalence statistically if t_B and t_A co-occur frequently in similar contexts in other documents. Berry et al. [SIAM Review, 1995] offer a good example:

> Consider the words *car, automobile,* driver, and *elephant.* The terms *car* and *automobile* are synonyms, *driver* is a related concept and *elephant* is unrelated. In most [e.g., traditional term vector] retrieval systems, the query *automobiles* is no more likely to retrieve documents about cars than documents about elephants, if the precise term *automobile* was not used in the documents. It would be preferable if a query about *automobiles* also retrieved articles about *cars* and even articles about *drivers* to a lesser extent. The derived [LSI} *k*-dimensional feature space can represent these useful term relationships. Roughly speaking, the words *car* and *automobile* will occur with many of the same words [i.e., in the same "context"] (e.g., *motor, model, vehicle, chassis, carmakers, sedan engine,* etc.), and they will have similar representations in *k*-space. The contexts for *driver* will overlap to a lesser extent, and those for *elephant* will be quite dissimilar.

(To be fair about it, a good query submitted to a traditional term-based system would use more terms than "automobile", e.g., perhaps some of the other contextual words mentioned in the passage quoted above.)

In other words, the traditional term-based vector space model assumes term independence. "Since there are strong associations between terms in language, this assumption is never satisfied [though it may be] a reasonable first order approximation." [SIGIR '94, Hull] LSI attempts to capture

some of these semantic term dependencies using a purely statistical and automatic method, i.e., without syntactic or semantic natural language analysis and without manual human intervention.

LSI accomplishes this by using a method of matrix decomposition called Singular Value Decomposition (SVD). LSI takes the original document-by-term matrix describing the traditional term-based document space as input. It produces as output three new matrices: *T*, *S*, and *D* such that their product *T***S***D* captures this same statistical information in a new coordinate space, *k*-space, where each of the *k* dimensions represents one of the derived LSI "features" or "concepts" or "factors." "[T]hese factors may be thought of as artificial concepts; they represent extracted common meaning components of many different words and documents." [JASIS, 1990, Deerwester et al.] *D* is a "document" matrix. Each column of *D* is one of the *k* derived concepts. Each row of *D* is the vector for a given document, specified in terms of the *k* concepts. The matrix element for the *j*-th concept in the *i*-th document represents the strength of association of concept *j* with document *i*. Hence, *D* specifies documents in *k*-space. Similarly, *T* is a term matrix. Each column as before is one of the *k* derived concepts. But in *T*, each row is a vector in *k*-space describing a term in the original collection, a term in the original term-by-document matrix that characterized the collection. Hence, "[e]ach term [in this matrix] is then characterized by a vector of weights indicating its strength of association with each of these underlying concepts." {JASIS, 1990, Deerwester et al.] In other words, each term vector (i.e., row) in *T* is "a weighted average of the different meanings of the term." [SIGIR '94, Hull] The diagonal elements in the 2nd matrix *S* assign weights (called "singular values") to the *k* LSI factors according to their significance. This allows the user to have some control over how many dimensions *k*-space is to have.

"[Some of] [t]he power of this decomposition comes from the fact that the new factors are presented in order of their importance (as measured by the diagonal of *S*). Therefore, the least important factors can easily be removed by truncating the matrices *T*, *S*, and *D*, i.e., by deleting some of the rightmost columns of these matrices. The remaining *k* columns [are] called the LSI factors." SIGIR '94, Hull] Note that *k* is a parameter under the user's control. Reducing *k* can eliminate "noise", e.g., "rare and less important usages of certain terms." However, if the number of dimensions (LSI factors) is too low, important information may be lost. The optimum number of dimensions obviously depends on the collection and the task. One report finds that improvement starts at about 10 or 20 dimensions, peaks between 70 and 100, and then decreases. [SIAM Review, Berry et al., 1995] As the number of LSI factors approaches the number of terms, performance necessarily approaches that of standard vector methods. Another report says that the optimum number of dimensions is usually between 100 and 200.

Projection of a set of documents into k space is optimal in the sense that the projection "is guaranteed to have, among all possible projections to a *k*-dimensional space, the lowest possible least-square distance to the original documents. In this sense, LSI finds an optimal solution to the problem of dimensionality reduction." {Schutze et al., SIGIR '97]

What does it mean to say that the *k* factors derived by the LSI procedure correspond to "artificial" concepts? It means that no attempt is made to interpret these k concepts, e.g., to describe them in simple English. Indeed, in many cases, it may not be possible to summarize these concepts, to explain what each one "means." What one *can* say is that a given document is heavily weighted

with regard to concept 1, doesn't deal at all with concept 2, is lightly weighted with respect to concept 3, etc.

What good does it do describe a document in terms of the relative importance to the document of *k* concepts, if one doesn't know what the *k* concepts mean? For a single document, such a description may have no value at all. But if one has a 2nd document also described in terms of weights of those same k concepts, then one *can* say how similar the documents are (in *k*-space). And, if one of those "documents" is a query (or a sample document used as a query, or the centroid of a set of sample documents), then one can say how close the given document is to the given query, in *k*-space of course. Following the usual vector space similarity methods, e.g., calculating the cosine similarities, one can rank documents by how similar they are to the query, in *k*-space. Similarity in *k*-space is more statistically meaningful, and therefore, one hopes, more semantically meaningful, than similarity in conventional term space, because the *k* concepts reflect statistical correlations in the document population, while the original terms do not.

Since one can compute query-document similarities using the *k*-by-document matrix, *D*, alone, what is the value of the term-by-*k* matrix, *T?* One answer is that it allows you to compute term similarities. Presumably, two terms are very similar if they co-occur, i.e., are strongly correlated, with many of the same other terms. Hence, such a similarity can be used to suggest to a user who enters a query, other terms, statistically similar to the terms he used, which could be added to his query. Or the similarities can be used to construct automatically a domain-dependent or collection-dependent thesaurus.

Notice, by the way, that although each row of matrix *T* is called a "term vector," the phrase is used quite differently in LSI terminology than in conventional vector space terminology. In the conventional vector space approach, a "term vector" is a vector in *document space* describing a *document* in terms of weights assigned to each term for the given document. In LSI, both terms and documents are described in LSI factor *k*-space. A term vector is a vector describing a given *term* in LSI *k*-space in terms of the weights assigned to the LSI factors for the given term. A document is described in LSI by a *document* vector specifying the weights assigned to the LSI factors for the given document.

Hearst, et al. [Text Retrieval Conference, TREC 4] point out an additional advantage of LSI with respect to the routing or classification of documents:

> The routing task can be treated as a problem of machine learning or statistical classification. The training set of judged documents is used to construct a classification rule which predicts the relevance of newly arriving documents. Traditional learning algorithms do not work effectively when applied to the full vector space representation of the document collection due to the scale of the problem … In the vector space model, one dimension is reserved for each unique term in the collection. Standard classification techniques cannot operate in such a high dimensional space, due to insufficient training data and computational restrictions. Therefore, some form of dimensionality reduction must be considered … [One approach is to] apply Latent Semantic Indexing (LSI) to represent documents by a low-dimensional linear combination of orthogonal indexing variables.

Berry, et al. {SIAM Review, 1995] discuss another advantage of LSI. It is especially useful for noisy input:

> Because LSI does not depend on literal keyword [i.e., term] matching, it is especially useful when the input text is noisy, as in OCR (optical character recognition), open input, or spelling errors. If there are scanning errors, and a word [name in this case] (*Dumais*) is misspelled (as *Duniais*), many of the other words in the document will be spelled correctly. If these correctly spelled context words also occur in documents that contain a correctly spelled version of *Dumais*, then *Dumais* will probably be near *Duniais* in the *k*-dimensional [LSI factor] space.

On the other hand, LSI has some serious drawbacks too. As Hull [SIGIR ‘94] points out:

> While a reduced representation based on a small number of orthogonal variables might appear to cut storage costs substantially [compared to the traditional term-based vector space model], the opposite is actually true … [The LSI representation requires storage of a substantially larger set of values.] In addition the LSI values are real numbers while the original term frequencies [weights] are integers, adding to the storage costs. Using LSI vectors, we can no longer take advantage of the fact that each term occurs in a limited number of documents, which accounts for the sparse nature of the term by document matrix.

Another disadvantage is that “the LSI solution is also computationally expensive for large collections … [However], it need only be constructed once for the entire collection [assuming a relatively static collection so] performance at retrieval time is not affected.” [SIGIR ‘94, Hull]

In one respect, LSI degrades retrieval time performance too. In a conventional vector space search using an inverted index, only documents that have some terms in common with the query must be examined. If the query is well-formulated, i.e., is composed of terms that are not overly common and serve to distinguish relevant from non-relevant documents, many documents will not contain any terms in common with the query and hence will not need to be examined at all. “With LSI, however, the query must be compared to every document in the collection.” [SIGIR ‘94, Hull] But this is not as great a disadvantage for LSI as it may at first appear. If (as will commonly be the case) the number of terms in a conventional query is greater than the number of factors in its LSI representation, the vector similarity calculation for a given document in conventional term space will take more time to compute than the corresponding calculation in LSI vector space. Moreover, most conventional vector space approaches use some form of query expansion to modify and expand the user’s original query. (This is discussed below in the section on Query Expansion and Refinement.) LSI can be viewed as a special kind of query “expansion”. [SIGIR ‘94, Hull] Conventional query expansion often results in a great increase in the number of terms in the query, whereas LSI may actually reduce the number of terms.

Given that the original LSI solution is computationally expensive, the question arises whether this expense must be incurred repeatedly each time new messages are added to the document collection. (This is an important consideration in any case where the document collection is not static,

but it is especially important if the LSI technique is applied to a routing application; in such an application, the document "collection" is an incoming stream that is continually changing - see section 8.) Fortunately, it is not always necessary to do a complete re-computation every time a new document arises. To begin with, the addition of one document to a large collection is not likely to have a very significant effect on the LSI computation, so it may be possible to ignore the effect. Secondly, there are two approaches to updating the LSI computation without re-doing the entire computation. [Berry et al., 1995]

The first, called "folding in," is the cheapest. Basically, you don't recompute the *k* factors at all. Nor do you recompute the weights of the *k* factors for existing documents or terms. Instead each new document just becomes a new column in the document matrix, described in terms of the original *k* factors. Similarly, if the new document contains some new terms, each new term becomes a new row in the term matrix, again defined in terms of the original *k* factors. So the process is relatively fast and cheap, but there is some degradation; not all the new correlations are being absorbed. Hence, all the original documents and terms occupy the same positions in *k*-space that they did before the folding-in of new documents occurred.

There is a more sophisticated (and naturally more computationally expensive) technique called "SVD updating." It starts with the original LSI database, just like folding in, but the weights associated with the k factors for each document and term are recomputed so that the addition of new documents and terms affects the positions of the existing documents and terms in k-space. Hence, the approximation is much better.

If these two updating procedures prove inadequate, the remaining alternative is to redo the LSI computation from scratch

The fact that (as noted above) an LSI term vector is "a weighted average of the different meanings of the term" can be either an advantage or a disadvantage. It is an advantage if the reduced representation in LSI removes "some of the rare and less important usages" of the given term, i.e., usages that are not relevant to the topic of the query. On the other hand, if the "real meaning [as used in the query and the relevant documents] differs from the average meaning [as captured by the LSI term vector], LSI may actually reduce the quality of the search." [Hull, SIGIR '94]

One more possible drawback of LSI is pointed out by Shutze et al. [SIGIR '95] LSI is not required for, and may actually degrade retrieval of, documents that are well described by a few terms. They give the example of a document about the Hubble Space Telescope, which can be very well retrieved by the single word "Hubble." LSI may actually obscure the key evidence, the presence or absence of the term "Hubble," in this case. On the other hand, LSI is much more effective "if there is a great number of terms which all contribute a small amount of critical information." This is particularly true if each of the documents on the desired topic only contains a subset of these terms. In such a case, LSI is more effective than a term-based classifier at combining evidence to identify documents about the given topic.

Similarity (between query and document) in LSI vector space is usually calculated by the inner product similarity measure as in the traditional document vector space approach. The normalized inner product, i.e., "cosine" similarity is generally used. However, Bartell et al. have shown that

the “inner product similarities between documents in the original [term] space are optimally preserved by the inner products [not normalized inner products] between corresponding vectors in the reduced space.” [Bartell, SIGIR ‘92] Hence, cosine normalization should be computed in the original term space and the LSI calculation then applied to these normalized vectors.

In any case, the value of LSI lies in (1) the elimination of spurious similarities, i.e., due to the same term being used in two different ways, and (2) the detection of similarities in LSI space that are invisible in ordinary term space, i.e., due to different but synonymous terms being used in similar contexts in different documents.

6.6 Vectors Based on *n*-gram Terms

The *n*-gram approach is in some respects the ultimate in vector space (and more generally, in statistical) approaches to IR. In the traditional vector space approaches described above, the dimensions of the document space for a given collection of documents are the words (or sometimes phrases) that occur in the collection; more precisely, they are the terms that remain after stemming, and removal of words that appear on a stoplist. By contrast, in the *n*-gram approach, the dimensions of the document space are *n*-grams, strings of *n* consecutive characters extracted from the text without regard to word length, and often completely without regard to word boundaries. Hence, the *n*-gram method is a remarkably “pure” statistical approach, one that measures the statistical properties of strings of text in the given collection without regard to the vocabulary, lexical, or semantic properties of the natural language(s) in which the documents are written.

The *n*-gram length (*n*) and the method of extracting *n*-grams from documents vary from one author and application to another. “Zamora uses trigram [$n = 3$] analysis for spelling error detection” [Pearce & Nicholas, JASIS, 1996]. Damashek uses *n*-grams of length 5 and 6 for clustering of text by language and topic (see below). He uses $n = 5$ for English and $n = 6$ for Japanese [Damashek, Science, 1995]. Pearce and Nicholas follow Damashek in using 5-grams to support a dynamic hypertext system [JASIS, 1995]. Some authors [Zamora et al., IP&M, 1981]; [Suen, IEEE Pattern, 1979] draw *n*-grams from all the words in a document but use only *n*-grams wholly within a single word. Others [Cavnar, TREC-2, 1993]; [Yannakoudakis et al., IP&M, 1982]; [Damashek, Science, 1995] also use *n*-grams that cross word boundaries, i.e., that start within one word, end in another word, and include the space characters that separate consecutive words [Pearce & Nicholas, JASIS, 1996].

Damashek’s *sliding window* approach [Science, 1995] is one of the most recent and inclusive, and the first to offer “convincing evidence of the usefulness [of the *n*-gram approach] for the purpose of categorizing text in a completely unrestricted multilingual environment.” Minimal preprocessing is required. Numbers and punctuation characters are usually removed. What remains is the alphabet plus the space character (27 characters for English). (Sometimes, no preprocessing is done at all.) The *n*-grams characterizing a document are then obtained by moving a window *n* characters in length through a document or query one character at a time. In other words, the first *n*-gram will consist of the first *n* characters in the document, the 2nd *n*-gram will consist of the 2nd through the (n+1)-th character, etc. Hence, there will be an *n*-gram starting with every character in the document (after preprocessing) except for the last *n*—1 characters. Each document can

then be specified as a vector of *n*-grams, one for each distinct *n*-gram in the document. Each component can be weighted just as the components of a conventional term vector are weighted, e.g., Damashek uses normalized *n*-gram frequency, the number of occurrences of the given *n*-gram in the given document divided by the total number of occurrences of all *n*-grams in the document. Similarly, Damashek computes the similarity between two documents using the familiar cosine similarity measure, which is just as useful in an *n*-gram document space as in a term-based document space.

Once the similarities among the *n*-gram-based document vectors have been computed, the documents can be clustered using a method such as those discussed in section 3.8. Damashek found this approach to be extremely effective for classifying a mixed-language collection of text documents, generating a distinct cluster for each distinct language. Indeed, such methods have been used to cluster documents hierarchically, i.e., by language group and by individual language within group. The beauty of this method is that it amounts to "blind clustering," i.e., the documents are classified without any prior linguistic knowledge. That is, there is no prior knowledge of the individual languages, or even of how many languages or language groups are involved.

A document space of *n*-gram-based vectors can be clustered by topic, as well as language (or by topic within language). However, clustering by topic introduces a problem not usually present in language clustering. When documents are blind-clustered by language, stoplists are not only not available (since they are language-dependent); they are also inapplicable. The very words that are typically placed on stoplists because they occur across most topics within the documents of a given language are the words that best characterize the language statistically. Putting it another way, words that have very little semantic content are good discriminators of a language since the objective is to classify all documents written in the given language regardless of what topic they discuss. Hence, Damashek employs a language-independent method of removing the "noise", the data that is common across topics. He translates the axes of the vector space so that the new origin is at the mean (the *centroid)* of all the document vectors. In that way, the origin becomes "a location that characterizes the information one wishes to ignore," the information common to the collection of documents. This location is equivalent to the "noise," the data that does not serve to distinguish one document from another. Translating the origin is equivalent to subtracting the centroid from each document vector, subtracting the common information one wishes to ignore. The document similarities can then be recalculated relative to this new origin.

But the technique of subtracting the centroid can be used to accomplish more than merely suppressing noise. It can also be used to recognize that two clusters of documents with different primary topics share a secondary topic. (Damashek offers the example of one cluster dealing with the primary topic of health care reform, and another cluster dealing with the primary topic of communicable diseases; they may share a secondary topic of AIDS epidemiology.) If one subtracts from each cluster its centroid (corresponding to its primary topic, the topic that all the documents share), the two clusters may become superimposed in document space, reflecting the secondary topic that they share. In this kind of application, the common data that is removed is not devoid of semantic content. It is merely common to a set of documents and hence not useful for distinguishing them. Hence, it has been called the "context" or the "background" [Cohen, JASIS, 1995] rather than "noise."

A more sophisticated *n*-gram weighting scheme based on the G^2 statistic [Bishop et al., 1975] has been used by Cohen [JASIS, 1995] to distinguish the background of a cluster of documents from the *highlights*, i.e., the words in a document that distinguish the document from its neighbors and hence can serve as an automatically-generated abstract that tells a user very rapidly what a given document is "about." Note that Cohen's method, although it highlights words, identifies the words to be highlighted by statistical characteristics of the *n*-grams of which they are composed. Moreover, Damashek's sliding window approach is employed, enabling phrase as well as word highlights to be identified. As before, the technique is language-independent.

Damashek's use of cosine similarity illustrates a general point: almost any method for weighting terms, normalizing terms, or normalizing term-based vectors is as applicable when the terms are *n*-grams as when the terms are words. Similarly, query expansion based on relevance feedback, discussed in section 3.6 below, is as applicable to *n*-grams as to words. (In practice, query expansion has not usually been combined with Damashek's *n*-gram method, because emphasis has been placed on its power for fast, inexpensive clustering, and interactive browsing.) The converse is also true: A method like centroid subtraction, applied above to *n*-gram vectors, is applicable to word-based term vectors as well. Language-dependent methods such as stoplist removal are not directly applicable to *n*-gram vector representations, but as noted below, can be combined effectively with n-gram analysis.

Because the sliding window used to obtain *n*-grams allows the system to obtain many slices of a given word, the performance of an *n*-gram system is remarkably resistant to textual errors, e.g., spelling errors, typos, errors associated with optical character recognition, etc. Damashek [Science, 1995] artificially corrupted his text (15% of the characters in error) and found that most of the documents in an uncorrupted cluster were still identified as belonging to the cluster (17 of 20). In a hypertext application [Pearce and Nicholas, JASIS, 1996], "the dynamic linkage mechanisms … are tolerant of garbles in up to 30% of the characters in the body of the text." Again, a major virtue of the sliding window *n*-gram approach for tolerance of garbles is that it is language-independent; it does not depend on prior linguistic knowledge.

A pure *n*-gram analysis does not use language-specific and semantic clues, i.e., stemming, stoplists, syntactically-based phrase detection, thesaurus expansion, etc. This theoretically limits its performance compared to methods that make effective use of language-specific as well as statistical clues. However, this limitation is currently more theoretical than actual, because most contemporary retrieval methods make only limited use of language-specific and semantic methods (a situation that may change in the future). Moreover, as noted below, *n*-gram analysis can be combined with word-based, syntactic, and semantic methods in a variety of ways. Further, as discussed above, many methods, e.g., automatic query expansion based on relevance feedback, are as applicable to n-gram analysis as to word-based analysis. On the other hand, precisely because *n*-gram analysis is language-independent, it is especially well-suited (given an adequate training set) to classification or clustering of documents by language. It should be noted here that the "languages" that this technology can classify are not restricted to natural languages, but can also include programming languages, indeed any class of "language" or representation that has distinguishable statistical properties.

The disadvantage of *n*-gram analysis with respect to language-specific processing is actually less serious than it might seem. Algorithms that use *n*-gram counts can be combined with identification of word boundaries to recognize roots shared by different words, conferring some of the same advantage as stemming algorithms, but with the further advantage of language independence. In some respects, these algorithms are better than conventional stemmers that only remove suffixes, e.g., they can recognize the resemblance of “quake” and “earthquake” {Cohen, JASIS, 1995]. (Of course, they can be deceived by spurious resemblances too.) And centroid subtraction, discussed below, provides some of the same benefits as stopword removal, but again in a language-independent fashion. Indeed, it is not always necessary to choose between *n*-gram analysis, and language-dependent methods. They can be combined. For example, language-dependent methods have been used successfully in conjunction with *n*-gram analysis to improve performance, e.g., if the language in which a collection of documents is written is known, words on a stoplist can be removed from each document before n-gram analysis is applied. [Onyshkevych, PC] Similarly, conventional stemming can be applied before *n*-gram analysis. However, the future extension of textual analysis to sophisticated semantic and knowledge-based methods will obviously have to be word-based.

Apart from language independence, the greatest virtue of *n*-gram analysis is that it is very simple to implement, and can run very fast (although the redundancy associated with a sliding window approach may make it expensive with respect to storage). Hence, another very effective way to combine *n*-gram analysis with language-based methods is to perform a two-stage search process. First, *n*-gram methods can be used to zero in rapidly, e.g., in an interactive browsing mode, on document clusters in a domain of interest; then, more refined and expensive language-based methods can be used to refine the search and perform the final retrieval.

But perhaps the feature of the *n*-gram approach that most sets it apart from other statistical methods, is its ability to group documents without any prior knowledge about the documents being grouped. Indeed, in this realm of “blind” clustering, it appears to have no serious competition.

7. Probabilistic Approach

There is no clear line separating probabilistic from statistical methods of IR. Indeed, there is a very close connection since probabilities are often calculated on the basis of statistical evidence. Most of the literature on probabilistic IR assumes that the evidence is so calculated. Of course, given a formula based on a probabilistic model, any source of evidence can be used to compute the probabilities to be plugged into the formula, but as a practical matter the evidence is usually statistical; in fact, it is may sometimes be the very same evidence, e.g., *tf*’s and *idf*’s, used in statistical, e.g., vector space, methods.

7.1 What Distinguishes a Probabilistic Approach?

Then what distinguishes a true probabilistic methodology from other statistical approaches? According to Cooper et al. [SIGIR ‘92]:

> In a thoroughgoing probabilistic design methodology, serious use is made of formal probability theory and statistics to arrive at the estimates of probability of relevance by which the documents are ranked. Such a methodology is to be distinguished from looser approaches -- for instance the "vector space" retrieval model -- in which the retrieved items are ranked by a similarity measure (e.g., the cosine function) whose values are not directly interpretable as probabilities.

7.2 Advantages and Disadvantages of Probabilistic Approach to IR

Cooper et al. [SIGIR '92] list four *potential* advantages of a true probabilistic design methodology:

1. "One has grounds for expecting retrieval effectiveness that is near-optimal relative to the evidence used."

2. There should be "less exclusive reliance on traditional trial-and-error retrieval experiments … to discover the parameter values that result in best performance." (As examples of such trial and error, consider the variety of term weighting schemes that have been tried in varied vector space experiments or the trials required to determine optimum values for the parameters, *A, B,* and *C* in the Rocchio relevance feedback formula, discussed in a later section.)

3. "[A]n array of more powerful statistical indicators of predictivity and goodness of fit [than precision, recall, etc.] become available."

4. "[E]ach document's probability-of-relevance estimate can be reported to the user in ranked output … [I]t would presumably be easier for most users to understand and base their stopping behavior [i.e., when they stop looking at lower ranking documents] upon … a 'probability of relevance' than [a cosine similarity value]." In general, actual estimates for each document of probability of relevance are more useful to a user than mere ranking of documents by probability of relevance (let alone ranking by some other similarity function). {Turtle and Croft, ACM Trans IS, 1991]

Yet, probabilistic methods have not yet been as widely used as these advantages would suggest. Moreover, where they have been used, they have achieved retrieval performance (measured by precision and recall) comparable to, but not clearly superior to, non-probabilistic methods. {Cooper, SIGIR '94] Cooper identifies various reasons for these shortfalls:

1. "[A]dvocates of nonprobabilistic methods … regard the formulation of exact statistical assumptions as an unnecessary theoretical burden on the researcher. They maintain (with some plausibility) that the time and effort spent on such analysis would be better spent on ad hoc experimentation using formalisms looser and friendlier than probability theory."

2. "The estimation procedures used in probabilistic IR are usually based on statistical simplifying assumptions or 'models' of some sort. The retrieval clues that bear on a document's probability of usefulness must somehow be combined into a single relevance probability, and modeling

assumptions are needed to accomplish the combining. Typically, the assumptions adopted for the task are crude and at best only approximately true … The introduction of simplifying assumptions known to be less than universally valid surely compromises to some degree the accuracy of the probability estimates that result." (Of course, similar simplifying assumptions are tacitly used in all other statistical approaches, e.g., the assumption of term independence in the vector space model.)

3. The assumptions underlying some IR models, most notably the widely used (and misnamed) "Binary Independence" IR model, can lead to logical inconsistencies.[Cooper, SIGIR '91, ACM Trans IS, 1995] Successes that have been achieved in spite of the inconsistency of the theoretical model are due to the fact that the actual assumptions used in practice are different than the assumptions of the theoretical model, and stronger than they needed to be.

In a probabilistic method, one usually computes the "conditional" probability *P(D|R)* that a given document *D* is observed on a random basis given event *R*, that d is relevant to a given query. [Salton, ATP, 89] [van Rijsbergen, 1979] If, as is typically the case, query and document are represented by sets of terms, then *P(D|R)* is calculated as a function of the probability of occurrence of these terms in relevant vs. non-relevant documents. The term probabilities are analogous to the term weights in the vector space model (and may be calculated using the same statistical measures). A probabilistic formula is used to calculate *P(D|R)*, in place of the vector similarity formula, e.g., cosine similarity, used to calculate relevance ranking in the vector space model. The probability formula depends on the specific model used, and also on the assumptions made about the distribution of terms, e.g., how terms are distributed over documents in the set of relevant documents, and in the set of non-relevant documents.

More generally, *P(D|R)* may be computed based on any clues available about the document, e.g., manually assigned index terms (concepts with which the document deals, synonyms, etc.) as well as terms extracted automatically from the actual text of the document. Hence, we want to calculate *P(D|A, B, C, …)*, i.e., the probability that the given document, *D*, is relevant, given the clues *A, B, C*, etc. As a further complication, the clues themselves may be viewed as complex, e.g., if the presence of term *t* is a clue to the relevance of document *D*, *t* may be viewed as a cluster of related clues, e.g., its frequency in the query, its frequency in the document, its *idf*, synonyms, etc. This has led to the idea of a "staged" computation, in which a probabilistic model is first applied to each composite clue (stage one), and then applied to the combination of these composite clues (stage two). [Cooper et al., SIGIR '92] This is discussed further below, in the section on Logistic Regression. It has also led to the idea of an "inference net" [Turtle & Croft, ACMtransIS, 1991] in which rules can be specified for combining different sources of evidence (automatically extracted index terms, manually assigned index terms, synonyms, etc.) to compute a "belief" that an information need has been satisfied by a given document. (See below, in the section on the Bayesian Inference Network Model.)

7.3 Linked Dependence

As noted above, the first step in most of these probabilistic methods is to make some statistical simplifying assumption. The objective is to replace joint probabilities (the probability of occurrence of two or more events, *A*, *B*, etc.) by a separate probability for each event. The most widely

used assumption is “Binary Independence” which Cooper [SIGIR ‘91, ACM Trans IS, ‘95] points out should really be called “Linked Dependence” [Cooper et al., SIGIR ‘92]. The model has been *called* “Binary Independence” because it has been assumed that to arrive at the model one must make the simplifying assumption that the document properties that serve as clues to relevance are independent of each other in both the set of relevant documents and the set of non-relevant documents, an “implausible presumption” [Cooper et al., SIGIR ‘92]. Cooper shows that it is sufficient to make the weaker “Linked Dependence” assumption that these properties are *not* independent but that the same degree of dependence holds for both the relevant document set and the non-relevant document set. In symbols (for two properties, *A* and *B*):

$$P(A, B|R) = K \cdot P(A|R) \cdot P(B|R)$$

$$P\langle A, B|\neg R\rangle = K \cdot P(A|\neg R) \cdot P(B|\neg R)$$

where ¬R means non-relevant. *K* is a crude measure of the dependence (the “linkage”) of properties *A* and *B*. If $K = 1$, this reduces to the pure independence property; the joint probability equals the product of the individual properties. Cooper points out that the weaker linked dependence assumption is sufficient to satisfy the requirements of models that have hitherto used the binary independence assumption. “This weaker assumption, though still debatable, has at least the virtue of not denying the existence of dependencies.” Therefore, efforts to remedy the clearly erroneous assumption of pure independence by providing empirical term co-occurrence information are less theoretically important than previously thought; the actual error that needs to be remedied is the assumption that dependencies are the same for relevant and non-relevant document sets. Of course, the latter may still be a significant consideration.

7.4 Bayesian Probability Models

Conditional (“Bayesian”) probabilistic models relate the *prior* probability of document relevance, i.e., the probability that a document selected at random will be relevant, to the *posterior* probability of relevance, i.e., the probability that an observed document is relevant, given the observed (or computed) features of the document. These features may include terms in the document, term statistics, manually assigned descriptors, phrase, etc., indeed all the clues employed in other statistical methods. Bayesian models vary primarily in how the “posterior” probabilities are calculated. They may also vary in how multiple sources of probabilistic evidence are combined, and in how probabilistic and non-probabilistic evidence are combined.

An important feature of Bayesian models is that one starts with a set of prior probabilities that must sum to 1, and one ends up with a set of posterior probabilities that must also sum to 1. [Winkler et al, Stat] (Some probabilistic methodologies may yield document retrieval status values (RSVs) that are not true probabilities, but rather are montone to the corresponding true probabilities. In that case, the RSVs can be used for document ranking but do not sum to one. However, in a true probabilistic model, they can always be converted to true probabilities (normalized) which do sum to one and which will be more meaningful to an end user.) One can view Bayesian probability as applied to IR as a process of "redistributing" probabilities of relevance from the prior probability distribution over all the documents in a given collection to a posterior distribution e.g., over the set of retrieved documents. Crestani and van Rijsbergen {SIGIR '95] call this "probability kinematics." They view this kinematics as a flow of probabilities among the terms serving as descriptors of a document collection. Initially, one has a prior distribution of the probabilities that terms, e.g., are good document descriptors (perhaps based on *idf*'s). Given a particular document D_i, one has a "flow" of probabilities from terms not in D_i to terms in D_i, e.g., perhaps based on simple term occurrence or term frequencies in the given document. These posterior probabilities for the terms that describe D_i also sum to one. The probability of the given document being relevant to a given query is then the sum of the posterior probabilities for the document terms that are also in the query. More generally, this probability may also depend on, e.g., term frequencies or other statistical characteristics of the query.

7.4.1 Binary Independence Model

"The simplest of these models is based on the presence or absence of independently distributed terms in relevant and non-relevant documents," [Shaw, IP&M, 1995] i.e., the distribution of any given term over the collection of documents is *assumed* to be independent of the distribution of any other term. Put another way, the probability of any given term occurring in a relevant document is independent of the probability of any other term occurring in a relevant document (and similarly for non-relevant documents). Hence, the model is referred to as the "binary independent" [BI} model. We saw above, in the preceding section, that this model actually should be called the "linked dependence" model because it actually depends on the weaker assumption that these probabilities are not independent, but rather that the same degree of dependence holds among relevant documents as holds among non-relevant documents, and that this degree of dependence can be captured by a proportionality constant. [Robertson et al, JASIS, 1976] [van Rijsbergen, 1979]. In this model, also known as the "relevance weighting theory" [Efthimiiadis, IP&M, 1995], we start with p_k, the probability that term t_k appears in a document given that the document is relevant and u_k, the probability that t_k appears in a document given that the document is non-relevant. (As always, relevance is defined relative to a given query or "information need.") Using Bayes' rule of inference and the BI (more precisely, linked dependence) assumption, one can derive a function for ranking documents by probability of relevance. [van Rijsbergen, 1979] In this function, each term t_k receives a weight w_k given by:

$$w_k = \log \frac{p_k \cdot (1 - u_k)}{u_k \cdot (1 - p_k)}$$

The "odds" of t_k appearing in a relevant document is $p_k/(1-p_k)$. Similarly, the "odds" of t_k appearing in a non-relevant document is $u_k/(1-u_k)$. Hence, w_k measures the odds of t_k appearing in a relevant document divided by the odds of t_k appearing in a non-relevant document, i.e., the *odds ratio*. Taking the log makes this function symmetric: $w_k = 0$ if $p_k = u_k$, is positive if $p_k > u_k$, and is negative if $p_k < u_k$. Hence, w_k is a good measure of how well a term can distinguish relevant from non-relevant documents. The function w_k is called the "term relevance" weight or "term relevance function," or "logodds," i.e., the log of the odds ratio for t_k. Plainly, w_k will be a very large positive number for a term that has a high probability of appearing in a relevant document and a very low probability of appearing in a non-relevant document (and a very large negative number if the probabilities are reversed). Moreover, if the set of index terms in a document collection satisfies the BI or "linked dependence" condition, the odds ratio (odds of relevance divided by odds of non-relevance) for a given document D is merely the product of the odds ratios of all the t_k appearing in D, i.e., the product of all the corresponding w_k. Hence, the *log* of the odds ratio for D can be computed as the *sum* of the w_k. In other words, the logodds of relevance of D is computed as the sum of the w_k for index terms appearing in D.

The trick is to find a way of computing p_k and u_k. If relevance data is available, e.g. from manually evaluating a previous run with the same query against the same collection or against a training set, then p_k and u_k can be estimated from a 2x2 "contingency" table summarizing the relevance judgments, as given below:

Table 2: Contingency Table of Relevance Judgments

	No. of relevant Documents	No. of non-relevant Documents	Total
No. of documents including term t_k	r	$n-r$	n
No. of documents excluding term t_k	$R-r$	$(N-R) - (n-r)$	$N-n$
Total	R	$N-R$	N

Here N is the total number of documents in the collection, n is the total number of documents that contain term t_k, R is the total number of relevant documents retrieved, and r is the total number of relevant documents retrieved that contain term t_k. From this table, we can estimate p_k as r/R (the proportion of relevant documents containing t_k),and u_k as $(n-r)/(N-R)$ (the proportion of non-rele-

vant documents containing t_k). Obviously, this assumes that "the term distribution in the relevant items previously retrieved [or in the training set] is the same as the distribution for the complete set of relevant items, and that all non-retrieved items [*N-R*] can be treated as non-relevant." [Salton et al, JASIS, 1990]. The latter assumption is necessary to allow us to assume that *N-R* (all retrieved non-relevant documents plus all non-retrieved documents) equals the total number of non-relevant documents. The former assumption allows us to treat the proportion of relevant documents containing t_k in the retrieved sample as characteristic of the proportion in the complete collection, for all t_k.

Equivalently, the odds of t_k appearing in a relevant document is *(r/R)/((1-r/R) = r/(R-r)*, and the odds of t_k appearing in a non-relevant document is *(n-r)/(N-R)*/[*1-(n-r)/(N-R)*] = *(n-r)/(N-R-n+r).* Inserting these values into the formula for w_k, we obtain:

$$w_k = \frac{r(N - R - n + r)}{(R - r)(n - r)}$$

This formula for w_k obviously breaks down if p_k equals one ($r = R$) or zero ($r = 0$). Similarly, w_k breaks down if u_k equals one ($n\text{-}r = N\text{-}R$) or zero ($n = r$). Statistical theory has been used to justify modifying the formulas for p_k and u_k to avoid these singularities by adding a constant c to the numerator and one to the denominator, where $c = 0.5$ or $c = n/N$. [Robertson et al., 1986] However, there are always cases where these constants dominate the computation and distort the results. [Shaw, IP&M, 1995] Shaw proposes to avoid these problems by using the unmodified formulas everywhere but at the singularities, and specifying alternative formulas at the singularities.

If the constant 0.5 is inserted in the formula for w_k, the result is:

$$w_k = \frac{(r + 0.5)(N - R - n + r + 0.5)}{(R - r + 0.5)(n - r + 0.5)}$$

It should be noted that in the term relevance model described above, the probabilities of relevance and non relevance, given t_k, and the corresponding logodds function w_k, are based entirely on the presence or absence of each term t_k in relevant and non-relevant documents. A term t_k is favored, i.e., given a high w_k, if it appears much more frequently in relevant documents than in non-relevant documents. It receives no "extra credit" for appearing more frequently than another term t_j in relevant documents.

The simple term relevance weight given above is based on the contingency table, which is constructed on the basis of a training sample. Hence, the term relevance weight function is based entirely on the assumption that the user possesses a training sample that is adequate in size and representative of the collection(s) to which the function is to be applied. What if one has no training sample? Put another way, what should the "prior" term weight be, before any data from the

collection is sampled? Robertson et al. [SIGIR '97], argue that this prior weight should reflect the fact that terms occurring in a large proportion of documents have little value for predicting relevance. Hence, they advocate using the *i*df as the prior weight, or (equivalently) the weight to be used in the absence of relevance information. On the other hand, if ample term relevance information is available, the term relevance function above applies. Hence, they propose a term weight function that varies smoothly from *idf* (for zero relevance data) to the above term relevance function, w_k, for ample relevance data.

Moreover, ample relevance data means not only an adequate sample of relevant documents, but an adequate sample of non-relevant documents. Hence, Robertson et al. define *S*, the number of known non-relevant documents (analogous to *R* for relevant documents), and *s*, the number of known non-relevant documents containing a given term, t_k (analogous to *r*) Note that *S* is *not* the same as *N*-*R*, and *s* is *not* the same as *n*-*r*. Given these six variables, *R*, *r*, *S*, *s*, *N*, and *n*, Robertson et al. define a function that varies from pure *idf* (for *R*=0, *S*=0, i.e., no data available about relevance and non-relevance) to the above w_k term relevance function for large *R* and *S*. They begin by observing that the logodds formula above, the log of the ratio of the odds of relevance to the odds of non-relevance, stated in terms of p_k and u_k, can be rewritten as:

$$w_k = \log\frac{p_k}{(1-p_k)} - \log\frac{u_k}{(1-u_k)}$$

The first term above is the logodds of relevance given the presence of term t_k. The second term is the logodds of non-relevance, given the presence of term t_k. Robertson et al. express each of these terms as a linear sum of an *idf* term and a term relevance term. The resulting weight function is:
:

$$w^{(1)} = \frac{k_5}{k_5+\sqrt{R}}\left\langle k_4 + \log\frac{N}{N-n}\right\rangle + \frac{\sqrt{R}}{k_5+\sqrt{R}}\log\frac{r+0.5}{R-r+0.5}$$
$$-\frac{k_6}{k_6+\sqrt{S}}\log\frac{n}{N-n} - \frac{\sqrt{S}}{k_6+\sqrt{S}}\log\frac{s+0.5}{S-s+0.5}$$

Here k_4, k_5, and k_6 are "tuning constants" that can be adjusted to tune the weighting function. The function is based on the "assumption... that the effect should be linear in the square root of *R*, on the grounds that the standard error of an estimate based on a sample is proportional to the square root of the sample size." It can be readily seen that if there is no relevance data, i.e., *R*, *S* = 0 (which implies *r*, *s* = 0 too), the weighting function reduces to:

$$w^{(1)} = k_4 + \log\frac{N}{N-n} - \log\frac{n}{N-n}$$
$$= k_4 + \log\frac{N}{n}$$

which is the traditional *idf* function plus a tuning constant. If *R* and *S* are large, the 2nd and 4th terms (the two term relevance function terms) dominate, and the weight function reduces to a pure "evidence-based," or "training-set-based" weight function:

$$w^{(1)} = \log\frac{r + 0.5}{R - r + 0.5} - \log\frac{s + 0.5}{S - s + 0.5}$$

$$= \log\frac{(r + 0.5) \cdot (S - s + 0.5)}{(R - r + 0.5) \cdot (s + 0.5)}$$

7.4.2 Bayesian Inference Network Model

An alternative approach to applying conditional (Bayesian) probability in IR, an approach that "depends less upon Bayesian inversion," is the inference network retrieval model. [Turtle & Croft, ACM Trans IS, 1991] "Inference networks can be used to simulate both probabilistic and Boolean queries and can be used to combine results from multiple queries." They can also be used to combine multiple sources of evidence regarding the relevance of a document to a user query, e.g., a document may be represented by both terms extracted automatically from the document itself, and terms or concepts assigned manually as index terms. The inference network provides a natural way to combine these sources of evidence to determine the probability (in this context, often called the *belief)* that the given document satisfies a given user query or *information need*. The Bayesian inference network approach has been implemented in the INQUERY system. [Callan et al., IP&M, 1995] [Callan et al., DB&ExSysApp, 1992] INQUERY also employs some semantic features, e.g., concept recognizers, which are discussed in a later section. [Callan et al., DB & ExSys, 1992]

An inference network is a probabilistic retrieval model, but it differs from typical retrieval models. [Croft et al., SIGIR '91] A typical model computes "*P(Relevance|Document, Query),* the probability that a user decides a document is relevant given a particular document and query." The inference net model computes "*P(I|Document),* the probability that a user's information need is satisfied given a particular document." This probability can be computed separately for each document in a collection. The documents can then be ranked by probability, from highest probability of satisfying the user's need to lowest. The result is a ranked list of retrieved documents, as with more traditional retrieval models. Moreover, the list of retrieved documents can be cut off at a probability threshold, e.g., only retrieve documents with a probability greater than 70% of satisfying the user's need. Such a probability threshold is likely to be more meaningful to the user than a cosine similarity threshold drawn from the vector space model.

Pearl [Prob Reas, 1988] has an excellent discussion of inference networks in general, and Bayesian networks in particular. He points out that these networks solve an important problem with probabilistic models (and more generally, with "extensional" models):

> Interpreting rules as conditional probability statements, *P(B|A) = p,* does not give us a license to do anything. Even if we are fortunate enough to find *A* true in the database, we still cannot assert a thing about *B* or *P(B*), because the meaning of the statement is "If *A* is true and *A* is the *only* thing that you know, then you can attach to *B*

> a probability *p*." As soon as other facts *K* appear in the database, the license to assert *P(B)* = *p* is automatically revoked, and we need to look up *P(B|A, K)* instead.

The point is that *K* may cause us to revise or retract conclusion *B*. The ability to retract a previous conclusion, called "non-monotonic" reasoning, is forbidden in classical logic but is essential to the kind of plausible reasoning under uncertainty that human beings actually do, and which pervades information retrieval. The great virtue of inference nets is that they organize our knowledge so that all the propositions, *A*, *K*, etc., on which a given conclusion *B* depends are immediately accessible, i.e., they are the parents (directly or indirectly) of *B*. Moreover, a rule for computing the probability of *B* from the probabilities of its parents can be attached to the node for *B*.

The inference network is a graph and consists of nodes connected by directed line segments ("edges"). The nodes are true/false propositions. An edge is drawn from node *p* to node *q* if *p* "causes" or "implies" *q*. We can then call *p* a "parent" of *q*. As the network is applied to IR, the root nodes are documents, i.e., propositions of the form, "Document D_i is observed." There is a document node for each D_i in the given collection. These document nodes are parents of "text" nodes. (See Figure 1.) The *j*-th text node for document D_i is a physical representation of the text of D_i, i.e., the proposition that, "Text representation $T_{i,j}$ of document D_i has been observed." For most purposes and in most IR systems, no distinction is drawn between the document and its text representation. But drawing this distinction allows for the possibility that a text representation might be shared by several documents, e.g., in a hypertext system several documents might have links to the same shared text. It also allows for the possibility that the children of a document node might include not only text nodes but also audio, video, figure, etc., nodes. (Here we are evidently talking about both multiple representations of a document, and multiple components of a multimedia document.) The distinction between document nodes and text nodes will be ignored in the discussion that follows.

The text nodes in turn are parents of "content representation" nodes ("representation" nodes for short). (See Figure 1.) These are the "descriptors" or "index terms." As noted above, the text of a document might be indexed both by terms automatically extracted from the document (as discussed in connection with vector space and Boolean models), and manually assigned descriptor terms or concepts. (For example, as Turtle and Croft point out, there may be two representation nodes associated with the phrase "information retrieval", one corresponding to the actual occurrence of the phrase in one or more documents, the other corresponding to the manual assignment of the concept expressed by the phrase to one or more documents, i.e., the human indexer's judgment that the given documents are "about" information retrieval.) Hence, there may be two or more subsets of representation nodes, each corresponding to a different method of describing or representing documents. A given representation node for term t_k may correspond to the proposition, "t_k is a good document descriptor." If D_1 is a parent of t_k, then the link from D_1 to t_k specifies the probability or "belief" in the proposition that t_k is a good descriptor of the document, given that the document is D_1. A term is a "good descriptor" of a document if its presence in the given document serves to characterize the document and distinguish it, e.g., statistically, from other documents that are about other topics. Note (see Figure 1 below) that if t_k occurs in both document D_2 and document D_i, then nodes D_2 and D_i are both parents of node t_k. And if in addition, concept c_j is assigned manually as a descriptor of D_2, then D_2 is a parent of both t_k and c_j, i.e., document D_2

is described (with some probability or to some extent) by concept c_j and contains term t_k with some probability that t_k is a good descriptor of D_2.

Given a document D_1, one wishes to compute the probabilities of the propositions represented by the child nodes of D_1, the children of *those* nodes, and so on. Hence, an inference net must provide some method of specifying the conditional probability of the proposition at node q, given the probabilities of the propositions at its parents, p_1, p_2, etc. The mechanism chosen by Turtle et al. is called a "link matrix." Each node is assigned such a matrix. The link matrix for node q contains two rows; the first row corresponds to proposition q being false, the second row corresponds to q being true. The link matrix for q has a column for each logical combination of parent truth values. For example, if q has three parents, p_1, p_2, and p_3, each of which can be either true or false, then there are eight columns: a column for all three parents being false, a column for p_1 being true and p_2 and p_3 being false, and so on, up to all three parents being true. Each of these eight combinations is a proposition that may influence our belief that q is true or false. Each entry in the matrix contains a weight corresponding to how strongly we believe the truth or falsity of the corresponding proposition should influence our belief in the truth of q. For example, the cell corresponding to column (i.e., proposition) "p_1 and p_2 true, p_3 false," and row "q true," contains our estimate of how much the truth of the given proposition should influence our belief in the truth of q.

Given the link matrix for node q, $P(q)$ can be computed by summing all the possible prior probabilities. Each prior probability is the probability of some combination of true and false for the parent propositions. Hence, each prior probability corresponds to a column of the link matrix. For example, consider the column "p_1 and p_2 true, p_3 false,." Let P_1, P_2, and P_3 be the probabilities that the propositions p_1, p_2 and p_3 are true. Then the prior probability of the given column (assuming the three parent propositions are independent of each other) is $P_1*P_2*(1-P_3)$. The prior probability is computed similarly for each of the eight columns, e.g., the prior probability for column "p_1 false, p_2 false, p_3 true" is $(1-P_1)*(1-P_2)*P_3$, and so on. These eight prior probabilities are summed to compute the total prior probability for q. As noted above, if some of the parents are more important than others, a weight for each prior probability appears in the link matrix. Each prior probability is multiplied by its weight from the link matrix before summing. Typically, the weight for each prior probability is a function of the weights assigned to the true parents in the given combination. For example, let w_1, w_2, and w_3 be the weights assigned to parent propositions p_1, p_2, and p_3 respectively. Then, each of the eight prior probabilities can be weighted by the normalized sum of the weights for those propositions that are true. Hence, the prior probability for column "p_1 and p_2 true, p_3 false" would be multiplied by $(w_1+w_2)/(w_1+w_2+w_3)$, yielding a weighted prior probability of $P_1*P_2*(1-P_3)*(w_1+w_2)/(w_1+w_2+w_3)$. Then, the total prior probability (sum of the prior probabilities for each column of the link matrix) is multiplied by the probability of q given the prior probabilities, $P(q|p_1, p_2, p_3)$.

Note that the link matrix is a conceptual representation. A pure link matrix contains a column for each logical combination of parent nodes. Hence, if the number of parents is large, the number of columns is *very* large; the number of columns grows exponentially with the number of parents. Specifically, if node q has n parents, its link matrix must (in general) have 2^n columns. Correspondingly, an operator represented by such a matrix runs in $O(2^n)$ time; it "requires $O(2^n)$ floating point operations." Clearly, for large n, a more efficient implementation or an operator with a simpler interpretation, must be employed. An example, discussed below in connection with

extended boolean operators, is link matrices that satisfy the Parent Indifference Criterion, i.e., matrices in which the value of the child node is determined wholly by the number of parents that are true, not at all by which ones are true. For such matrices, the number of columns clearly grows linearly, not exponentially, with the number of parents. Such operators run in $O(n^2)$ time. In certain important cases (see below), the run in linear time.

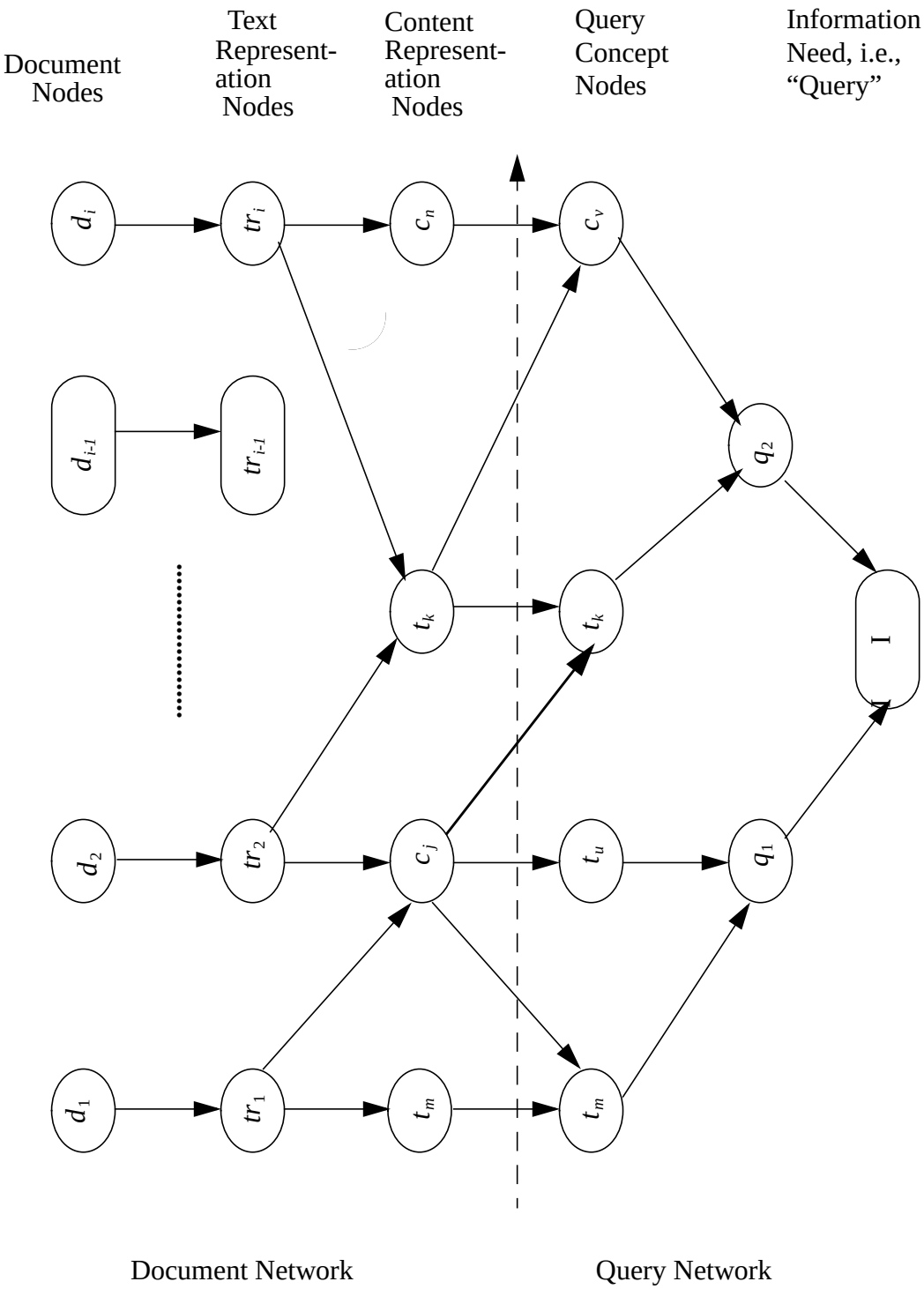

Figure 1: Example of Document Inference Network.

As a further refinement of the inference network structure, term dependencies, e.g., the tendency of two or more terms to co-occur, can be represented by links from one term to another. For example, a link from t_1 to t_2 may represent the probability that term t_2 will occur in a given document, given that term t_1 occurs in the given document. However, the network specifier must be careful to avoid cycles (see discussion below), e.g., a link from t_1 to t_2, a link from t_2 to t_3, and then a link from t_3 to t_1, would be forbidden! Document dependencies can also be represented, e.g., a citation in document D_1 to document D_2. If clustering techniques (see below) are used to establish that a set of documents D_1, D_2, …, D_c are more similar to each other than to any document not in the cluster by some appropriate criterion, then this can be represented by a cluster node which "represents" the cluster. Cluster nodes can themselves have representation nodes. Thus if document D_k belongs to cluster C_i, and C_i has a representation node t_j, then the presence of t_j in the user's query will strengthen belief that D_k satisfies the user's information need even if t_j does not appear in D_k itself.

Once the document network is built, capturing all the dependencies among documents and their representations, we assign probabilities to the nodes. Each document node receives a "prior probability," generally equal to *1/(collection size)*. This is the probability that a given document will be observed, given that a document is selected at random. "Each representation node contains a specification of the conditional probabilities associated with the node given its set of parent text [document] nodes." For example, the conditional probability of term node t_i given parent D_j is the probability that t_i is a good descriptor of D_j, i.e., is effective for the purpose of characterizing D_j, and distinguishing D_j from other documents in the collection. This is specified symbolically as $P(t_j|D_j=true)$. Here," t_j" is the proposition that "t_j is a good descriptor of the observed document," and "$D_j=true$" is the proposition that "D_j has been observed." As described above, the probability of the proposition associated with a given term or manually assigned descriptor, can be specified by a link matrix or its equivalent. However, for the special case where the parent events are observations of documents, the general scheme described above for specifying and evaluating a link matrix is modified in one essential respect: Although many documents may contain a given term, t_j, and hence may be parents of t_j's representation node, each document D_i is observed separately (the observed document is said to be "instantiated"), and hence its probability of satisfying the user's information need, *I*, is computed separately. The documents can then be ranked by the probability that they satisfy *I*. Hence, the link matrix of a representation node t_j corresponds to observation of a single document containing t_i and observation of *no* other documents. There are no columns in t_j's link matrix corresponding to several documents observed at the same time. Hence, the link matrix for t_j contains only two columns: "Document D_i observed," i.e, $P(t_j|D_i=true$ and all other parents = *false*) and "No document observed," i.e., $P(t_j|$all parents = *false*).For convenience, these conditional probabilities are usually expressed for short as: $P(t_j|D_i=true)$ and $P(t_j|D_i=false)$. This two-column matrix can readily be represented in closed form.

The conditional probability of the proposition "t_j is a good descriptor" given the event "Document D_i is observed" can be based on any available evidence. However, as observed in the section on Building Term Vectors, "*tf***idf*" is a good statistical measure of the ability of a given term to distinguish a given document. Turtle et al. employ a variant of this measure. However, their variant, 0.4 + 0.6**tf***idf*, departs from conventional vector term weighting schemes in one striking respect. The probability (or "belief" as the inference net folks like to call it) is non-zero even when the

term frequency for the given document is zero, i.e., even when the term doesn't occur in the document! The constant component, 0.4 in the above function, is called the "default probability." It corresponds to the view that a given term may have some probability of being a valid descriptor of the document, even if it is not observed in the document. In other words, $P(t_j|D_i=false) = 0.4$. For example, the given term itself may not be present, but a synonym may occur in the document. Or, terms that frequently co-occur with the given term may be present. Or observation may be restricted to the title or abstract, or a summary of the document; the given term may be absent from these surrogates, yet present in the full text of the document. The constant 0.4 was arrived at, not by any deep theory, but purely by experiment. Turtle et al. tried a wide variety of linear functions of the form $A + B*tf + C*idf + D*tf*idf$, where *tf* was normalized by maximum within-document term frequency, and *idf* was normalized by collection size. They found that the variant above gave the best results.

It might have been expected that the default probability would be dependent on *idf*, but experiment showed that a constant default performed as well. To complicate matters further, Greiff et al. [SIGIR '97] developed a computationally tractable, probabalistically motivated soft [extended] boolean operator based on a link matrix. (See below.) They found that to achieve performance comparable to the *p*-norm model (see section on Extended Boolean Approach), they had to set the default probability to zero!

So far, we have discussed that part of the inference network that describes a collection of documents. Not surprisingly, it is called the "document network' and it is calculated once for a static collection. Naturally, it is updated if the collection it describes is updated. The document nodes are the top or "root" nodes of the document network; the representation nodes are the bottom or "leaf" nodes. (The document network is not a tree since it has multiple roots and a text or representation node can have multiple parents. But it is directed and acyclic, i.e., no "loops," so it is possible to use graph tree terminology and talk about root, parent, and leaf nodes. For short, a directed acyclic graph is called a "DAG.") Note: There is a very good reason why loops (cycles) must be avoided in Bayesian inference networks. As Turtle and Croft point out, "evidence attached to any node in the cycle would continually propagate through the network and repeatedly reinforce the original node."

The other part of the inference network is the "query network." (See Figure 1.) It too is a DAG. Its multiple roots are the concepts that express the user's information need. These concept nodes may be the parents of multiple intermediate "query" nodes. Each query node is a parent of the single leaf node representing the user's information need. The use of those intermediate query nodes allows an information need to be expressed as multiple queries. Naturally, a separate query network is constructed for each user information need. "The single leaf representing the information need [of a given query network] corresponds to the event that an information need is met."

To apply a given information need to a given document collection, the query network corresponding to the given need must be attached to the document network corresponding to the given collection. This is accomplished (see Figure 1) by specifying parent-child links from the content representation nodes of the document network to the concept nodes of the query network, i.e., the leaf nodes (or at least some of them) of the document network become parents of the root nodes (at least some of them) of the query network. In the simplest case, a representation node from the

document network and a concept node from the query network may be identical, e.g., the former may be a term found in certain documents, and the latter may be a term specified in a user query. (In Figure 1, t_m is a term in document D_1 and also a term in the query I. Similarly, term t_k is a term in both document D_2 and document D_i, and also a term in the query I.) The link from the one to the other then expresses the relationship that the observation of the given term in a given document contributes evidence to the belief that the document satisfies the given information need. In a more complex case, multiple representation nodes may be parents of a single query concept node which is not identical to any of its parents. This occurs for example, when documents are represented in multiple ways, and queries use "concepts that do not explicitly appear in any document representation." For example in Figure 1, concept c_n is a (perhaps manually assigned) concept descriptor of document D_i, and t_k is a term or phrase occurring in D_i. Both c_n and t_k are parents of query concept c_v. Document descriptor concept c_n might be the concept "information retrieval," query concept c_v might be the concept "textual retrieval," and document term descriptor t_k might be the phrase "information retrieval" actually occurring in D_i (and also D_2). So, the given structure tells us that both the presence of the phrase "information retrieval" in a document (as a string in its text representation) and the presence of the concept "information retrieval" (as a manually assigned concept descriptor in its semantic representation), contribute to the belief that the document is about textual retrieval, which contributes, in turn, to a belief that the user's information need I is satisfied.

Once a query network for a given information need, I, has been attached to a document network for a given collection C, it becomes possible to compute the "belief" (the probability) that the information need has been satisfied by a given document or subset of documents. We must specify an operator or estimation rule, e.g., using a link matrix or its equivalent, for every non-root node of the total retrieval inference network, specifying how that node's probabilities are to be estimated given the probabilities of its parents. Thus we specify how the probability of I being satisfied depends on its parent query nodes, how the probability of each query node being true depends on the probabilities of its parent concept nodes, how the probability of each query concept node being true depends on the probabilities of its parent document representation nodes, and so on. If we select some particular document D_i, we set D_i's node to true and all other document nodes to false. This "evidence" percolates down through the network as we calculate the probability of each representation node given the evidence of its parent document nodes, the probability of each query concept node given the probabilities of its parent representation nodes, the probability of each query node given the probabilities of its parent concept nodes, and finally the probability (the "belief") that the user's information need, I, is satisfied, given the probabilities of its parent query nodes. Hence, by selecting some particular document D_i, we can compute the probability (the "belief") that the user's information need, I, is satisfied by D_i. We can repeat this process for each document in the collection, thus computing a probabilistic ranking for the documents.

Note that the component terms or concepts that comprise a query can be combined probabalistically, as described above. On the other hand, link matrices can also be used to specify non-probabilistic, e.g., boolean, operators.

For example, a strict boolean AND means that the belief in the truth of node q depends on the truth of all its parents. If, as above, q has three parents, p_1, p_2 and p_3, then the link matrix row for q = *true* will have a zero for every column except the last one; the last column, "p_1 and p_2 and p_3

true" will contain a one. The row for AND false is, of course, the complement: a one in every column except the last. Hence, the meaning of the link matrix is that *q* is true if and only if p_1, p_2, and p_3 are all true. Similarly, the *q = true* row of the link matrix for boolean OR will contain a one for every column except the first column, "p_1, p_2, and p_3 all false," a zero in that first column. Note: the presence of all ones and zeros in the link matrix for a strict logical OR or AND is a way of saying that the parents are unweighted, i.e., each term in a strict boolean AND or OR is just as important as any other.

Link matrices can even be specified efficiently for *soft* (also called *extended*) boolean operators. (See section on Extended Boolean Approach.) [Greiff et al., SIGIR '97] Instead of specifying a column for each logical combination of parent truth values, Greiff et al. specify a column for each *number* of true parents, independently of *which* parents are true. So, if *q* has four parents, there will be five columns, corresponding to no parents true, one parent true, two parents true, three parents true, and all four parents true. The weight in the *true* row for, e.g., three parents true, is an estimate of the probability that *q* is true, given that exactly three of its parents are true, *any* three of its parents. In other words, the weight for the *true* row of column *i* is the conditional probability of *q* given *i* parents true, *P(q| i parents true*). Any link matrix of this type is said to satisfy the *Parent Indifference Criterion* (PIC). Understandably, the cases they explore are those in which the conditional probability is non-decreasing as the number of true parents increases, since the more parents are true, the more support there is for belief in *q*. Hence, the weights in the *true* row for *q* either remain the same or increase as the number of parents increases, i.e., if $j > i$ and row *0 = true* in link matrix *m*, then $m(0, j) \geq m(0, i)$. As before, the unconditioned probability of *q*, *P(q)*, is computed by multiplying the prior probability of each logical combination of parents by its corresponding weight from the link matrix, and summing these products. The difference is that for PIC operators, there is a single weight (and hence a single column in the link matrix), for all logical combinations in which the same number of parents are true. This fact not only reduces the matrix to $n+1$ columns; it also makes possible a much more efficient algorithm for operator evaluation. [Greiff, SIGIR '97] The curve of *P(q) = true* vs. # of true parents can be linear or non-linear. Separate matrices must be specified for the soft boolean OR and the soft boolean AND respectively. Commonly, but not necessarily, the parents are terms in the document being evaluated, that are also in the query. The probabilities in the link matrix for the soft boolean OR operator are set so that the curve rises rapidly for a small number of parents, and then increases more slowly; this corresponds to the idea that for a generalization of OR, a small number of query terms present in a given document count a lot toward the total probability that the document is relevant, but additional terms present add a little more. Similarly, for a soft boolean AND operator, the probabilities in the link matrix are set so that the curve increases slowly until most of the parents are true, then increases rapidly; this corresponds to the idea that for a generalization of AND, a small number of query terms present in a given document count a little, but presence of most or all of the query terms add a lot more to the total probability that the document is relevant. If the parents are terms in a given document, then truth of a parent is presence of the term in the given document. The parents can be weighted as before, e.g., using some variant of *tf*idf*. The child *q* can be the query (Information Need in the Turtle et al. terminology), or a concept within the query (see below).

Greiff et al. show that PIC operators can be implemented to run in $O(n^2)$ time. Moreover, if the probability vs. number of parents curve is piecewise linear, and "all but one of the pieces of the function is of constant width," the operator can be evaluated in $O(n)$ time. (Note, by the way, that

the strict boolean operators are also defined by PIC matrices. However, for the strict booleans, the link matrix can be represented by a closed form operator, so the PIC algorithm is not necessary.)

The operator at any given non-root node, N_i, computes a belief (probability) in N_i in terms of operands which are the beliefs in N_i's parent nodes. A set of operators have been implemented in the INQUERY system that can be specified at the level of the query network, i.e., that are effectively part of the INQUERY query language. They include strict boolean AND, OR, and NOT, extended (soft) boolean operators such as those discussed above, weighted sum (similar to those used for computing the cosine similarity of a document in document vector space), unweighted sum (i.e., mean), and maximum (maximum of operand values). There are also proximity operators which return not a belief but "true" if the proximity condition is satisfied or "false" if it is not satisfied. Proximity operators include unordered text window proximity operators (operands are terms that must occur in any order in a text window of *size* $\leq n$), and ordered interword proximity operators (operands are terms that must occur in a specified order with interword *separation* $\leq n$). A proximity value can be converted into a belief value by an operator such as "PHRASE" (i.e., if the operands satisfy the ordered proximity condition with n = 3, calculate the unweighted sum of their beliefs).[Turtle and Croft, ACM Trans IS, 1991] [Callan et al., IP&M, 1995] [Callan et al., DB & ExSys, 1992] The set of possible operators is clearly open-ended. However, every operator evaluates the probability (the belief) of a given node in terms of the probabilities (beliefs) of its parents. This is the basic "update procedure for Bayesian networks."

For example, suppose that the user's query includes an OR condition (strict or extended), e.g., she is looking for documents that contain t_1 or t_2 or t_3. This is represented, in the query network, by a node Q with three inputs, corresponding to t_1, t_2, *and* t_3. The t_1 input may represent the probability that t_1 is a good content descriptor, i.e., a good descriptor for purposes of distinguishing relevance, given that some document D has been observed in the current collection, and similarly for t_2, and t_3. These probabilities may be computed by a widely used ad hoc, statistical measure like *tf*idf* (see section on Building Term Vectors in Document Space), or by a measure based on some theoretical model, e.g., the BI model (see section on Binary Independence Model). Then, the output of the OR node will represent the joint probability that t_1 OR t_2 OR t_3 is a good content descriptor, given the probabilities associated with t_1, t_2, and t_3 separately. If the OR node is a concept node, c_1, then its output is the probability that the c_1 is present, given D, i.e., that D is "about" c_1. The separate boolean probabilities associated with the ANDs, ORs, NOTs, etc. of which the query is composed are then combined into one total probability for the user's query, the probability that the user's information need, expressed as a logical combination of concepts, has been satisfied by the observed document D. This same process can be applied to each document in the collection, thus generating a probabilistic ranking of the documents.

As a simple special case, the OR node or AND node of the preceding paragraph could be the user's actual query, i.e., there may be no intermediate concept layer. In that case, evaluation of the boolean node, given document D, would give the probability that D was "about" the user's query, i.e., satisfied the user's information need.

On the other hand, the user's query could be a term vector (either supplied directly by the user) or extracted from a free-text specification of the user's need. If the vector consists of the three terms t_1, t_2, *and* t_3, then these three terms are parents of the user's query node. The link matrix for the

query, *Q*, would contain (as before) a column for each of the eight logical combinations of truth value for the three parents. Note that "truth" of t_1 given *D* does *not* mean that t_1 is observed in *D*. If t_1 is present in *D*, truth of t_1 is the proposition that t_1 is a good descriptor of *D* (or in other words, contributes evidence that D is relevant to queries containing t_1). The value of t_1 is a measure of the probability that t_1 is indeed a good descriptor, i.e., some measure of its "goodness." The corresponding value of the proposition that t_1 is false is then $1\text{-}P(t_1 = true|D)$. On the other hand, if t_1 is not present in *D* at all, the value of the proposition "t_1 is a good descriptor" is zero, because *D* offers no support for the proposition. Correspondingly, the value of the proposition "t_1 is not a good descriptor is 1-0 = 1. (But note that if the concept of a *default probability*, discussed above, is employed, then the probability that t_1 is a good descriptor, given *no* document support, is not zero; using Turtle's function, given above, its value = 0.4.)

One essential virtue of the inference network approach is that it allows one to represent a complex set of dependencies, dependencies in a document collection, dependencies in a complex information need specification, and dependencies between the concepts that represent the document collection and the concepts used to express the information need. The belief that a given information need is satisfied by a given document or set of documents in the given collection can then be estimated by evaluating the operator at each non-root node of the network. Different sources of evidence (automatically extracted terms, phrases, paragraphs, and manually assigned descriptors), can be combined. Different query types, e.g., natural language and Boolean, "can be combined in a consistent probabilistic framework. This type of 'data fusion' has been known to be effective in the information retrieval context for a number of years." [Callan et al., IP&M, 1995] (Note: This "data fusion" is fusion of document representations and evidence, and fusion of queries. The next section deals with fusion of results, i.e., fusion of retrieved documents from different sources or different queries.) Last but not least, inference net evaluation does not require a complex closed form expression that captures all the dependencies. Instead, the logic of evaluation is spread over the network. However, the one problem that neither inference networks nor any other probabilistic representation can solve is the difficulty of ever knowing/estimating the dependencies and prior probabilities in a complex relationship among documents and queries.

7.4.3 Logical Imaging

Another approach to computing posterior probabilities, based on "non-classical logics," is described by Crestani and van Rijsbergen [SIGIR '95]. Given a document D_i, the key to "Logical Imaging" is to determine for every term t_k that indexes the given document collection and is *not* in D_i, the term in D_i that is "most similar" to t_k. Then posterior term probabilities for D_i are computed by transferring the prior probability of each term not in D_i to its most similar term in D_i. The transfer is additive, i.e., if terms t_j, t_k, and t_l are terms not in D_i, and all three are "most similar" to D_i term t_{i1}, then we add the prior probabilities of t_j, t_k, and t_l to the prior probability of t_{i1} to obtain its posterior probability. Probability is neither "created" nor "destroyed" in this process, merely transferred from non-document terms to document terms. Hence, if the prior probabilities for the entire term space sum to one, the posterior probabilities for a given document must also sum to one. The sum of posterior probabilities of document terms that are also in a given query then becomes the probability of relevance for the given document relative to the given query.

The obvious question is how to compute term "similarity" in order to compute for each term t_i the degree of similarity of every other term. (Given this ranking, it is then straightforward to determine for any term t_k that is not in a given document D_i, the term in D_i that is "most similar" to t_k.) The measure chosen by Crestani and van Rijsbergen is the "Expected Mutual Information Measure" (EMIM) between terms. As applied to IR, "[t]he EMIM between two terms is often interpreted as a measure of the statistical information contained in the first term about the other one (or vice versa, being a symmetric measure." [Crestani & van Rijsbergen, SIGIR '95]. Let T_i be a variable that takes on two values: T_i=1 means that term t_i occurs in a randomly chosen document and T_i=0 means that it does not. Similarly T_j is a variable for the occurrence or non-occurrence of another term t_j, $P(T_i)$ is the probability of event T_i, $P(T_j)$ is the probability of event T_j and $P(T_i, T_j)$ is the joint probability of events T_i and T_j. Then the EMIM is:

$$I(t_i, t_j) = \sum_{T_i, T_j} P(T_i, T_j) \cdot \log \frac{P(T_i, T_j)}{P(T_i) \cdot P(T_j)}$$

where the summation is over the four possible values of T_i and T_j together, i.e., T_i occurs and T_j does not, T_j occurs and T_i does not, they both occur, or neither occurs, in a given document. Note that if T_i and T_j occur independently, $P(T_i, T_j) = P(T_i)P(T_j)$ and $I(T_i, T_j)$ =0, so the EMIM is a measure of how much the two events, in this case co-occurrence of T_i and T_j in a document, departs from stochastic independence. [van Rijsbergen, 1979]

Crestani and van Rijsbergen also describe an extension of Logical Imaging called "General Logical Imaging." The difference is that instead of identifying the term t_l in D_i most similar to a given term t_k not in D_i, one identifies the set of terms, e.g., t_l, t_m, t_n, in D_i most similar to t_k. Then one transfers the prior probability of t_k to the set t_l, t_m, t_n according to a transfer function (called an "opinionated probability function") which prescribes how much of t_k's prior probability is transferred to the most similar term t_l, how much to the next most similar term, t_m, and so on.

7.4.4 Logistic Regression

Cooper et al. [TREC, 1994] [SIGIR '92], use "a probabilistic model … to deduce the general form that the document-ranking equation should take, after which regression analysis is applied to obtain empirically-based values for the constants that appear in the equation … The probabilistic model is derived from a statistical assumption of 'linked dependence.'" (See discussion of Linked Dependence in earlier section.) Logistic regression is the regression analysis method employed. That is, the "probability of relevance vs. document evidence" curve is fitted to a logistic regression function. The values of the "empirically-based" constants are derived from a training set.

Cooper et al. explain the reasons why logistic regression is more appropriate than standard (non-logistic) linear regression for predicting the probability of relevance. The most important reason is that the probability of relevance to be modeled is two-valued, i.e., every document in the training set is either known to be relevant (probability of relevance equals one), or known to be non-relevant (probability of relevance equals zero). Hence, the desired probability curve must fit a set of

data points all of which reside either on the horizontal p=0 line, or the horizontal p=1 line. Clearly, the sloping straight line generated by linear regression will fit these points very poorly for any possible slope. On the other hand, logistic regression generates an "*S*-shaped" curve. With appropriate parameters, the lower arm of the "*S*" can be made to approximate p=0, while the upper arm can be made to approximate p=1.

To obtain this "S-shaped" curve, we express the probability of relevance as a logistic regression ("logit") function, as follows:

$$P(R|X_1, \ldots, X_M) = \frac{e^{c_0 + c_1X_1 + \ldots + c_MX_M}}{1 + e^{c_0 + c_1X_1 + \ldots + c_MX_M}}$$

where the X_j are individual facts about a given document. Cooper et al. group the facts (the "evidence") from a given query/document pair into "composite" clues, A_i, (i = 1 to N), where each composite clue is composed of a set of facts X_j (j = 1 to M). For example, "if … A_i is a word stem, then X_1 might be (say) the relative frequency of the stem in the query, X_2 its relative frequency in the document, and X_3 its inverse document frequency in the collection." Hence, in contrast to vector space methods, and most other probabilistic approaches (except inference networks), which describe each document with a single level of features, a "feature vector," Cooper et al. model each document and query more accurately as *two* levels of feature: At the outer level, each document and query is described by a conventional feature vector of "composite" features, A_i. At the second (inner) level, each composite feature is expanded into a set of elementary features, X_i. Each composite feature, A_i, is related to its elementary features by logistic regression. The probability of relevance of the document as a whole is related to its composite features, the A_i, by the traditional linked dependence assumption.

The calculation turns out to be easier in terms of the log of the "odds" of relevance. The "odds" of relevance is defined as the ratio of the probability of relevance to the probability of non-relevance. Hence, the odds that a document is relevant, given a single composite feature, A_i, can be expressed in terms of its corresponding elementary features, as:

$$O(R|A_i) = O(R|X_1, \ldots, X_M) = \frac{P(R|X_1, \ldots, X_M)}{P(\neg R|X_1, \ldots, X_M)}$$

$$= \frac{P(R|X_1, \ldots, X_M)}{1 - P(R|X_1, \ldots, X_M)}$$

If one replaces the probability by the corresponding logistic regression function in the above identity, and takes the natural log of both sides, one obtains:

$$\log O(R|A_i) = \log O(R|X_1, \ldots, X_M) = c_0 + c_1X_1 + \ldots + c_MX_M$$

This equation gives the logodds of relevance given a single composite clue, A_i. To extend it to a set of N clues, A_1,..., A_N, one uses the linked dependence assumption, discussed above in an earlier section. This assumption is that the same degree of dependence holds for both the relevant document set and the non-relevant document set. This degree of dependence is expressed (crudely) as a common proportionality constant. Hence in symbols, we have:

$$P(A, B|R) = K \cdot P(A|R) \cdot P(B|R)$$

and

$$P\langle A, B|\neg R\rangle = K \cdot P(A|\neg R) \cdot P(B|\neg R)$$

Note that linked dependence, like pure binary independence, breaks a complex joint probability into a product of simpler separate probabilities for the individual "composite" clues. Cooper et al. point out that since each clue, each piece of evidence, is a separate factor in the linked dependence formulation, and hence makes a separate contribution to the total probability of relevance, the effect is that the probability of relevance computed for high-ranking documents, documents containing many clues matching the given query, will tend to be too high. They offer a couple of relatively crude methods of compensating for this effect, discussed below.

Dividing the second linked dependence equation by the first (which causes the proportionality constant K to cancel out), and using the identity,

$$\frac{P(A|R)}{P\langle A|\neg R\rangle} = \frac{O(R|A)}{O(A)}$$

we have:

$$\frac{O(R|(A, B))}{O(R)} = \frac{O(R|A)}{O(A)} \cdot \frac{O(R|B)}{O(B)}$$

or, generalizing to N composite clues, multiplying by $O(R)$, and taking the log of both sides, we have:

$$\log O(R|A_1, A_2, ..., A_N) = \log O(R) + \sum_{i=1}^{N} [\log O(R|A_i) - \log O(R)]$$

where the A_i are the "composite features" to be used to characterize any randomly chosen query-document pair, $O(R|A_i)$ is the odds that the document is relevant to the query given the composite feature A_i, $O(R)$ is the odds that the document is relevant to the query in the absence of any evidence, and $O(R|A_1, A_2, \ldots, A_n)$ is the odds that a document is relevant given all the composite features, $A_1, A_2, \ldots, A_n$. The sum in the above equation is taken over all the clues, i.e., from i =1 to i = N. Note that the composite features employed are those that match in at least one query-document pair in the training set. This is the "fundamental equation" employed by Cooper et al. for document ranking by logistic regression. Note that the regression is "logistic" in the sense that we

express the probability of relevance as a logit function. The actual regression performed (see below) is linear regression.

If all the composite features, A_i, for a query/document pair are of the same type, i.e., defined in terms of the same independent variables, X_j, then the set of composite features forms a matrix with $M + 1$ columns. For any query-document pair in the training set, there will be a set of rows in the matrix with one row for each composite feature, e.g., each word stem, that occurs in both query and document.(These are called "match terms.") Hence, each row in the training set corresponds to a query-document-term "triple." The row for a given composite feature of a given query-document pair will contain the M values of the elementary features, X_j, comprising that composite feature, e.g., the relative (normalized) frequency of the term in the given document, the relative frequency of the term in the query, etc. The given row will also contain one additional value: the logodds of relevance given the presence of A_i, $log\ O(R/A_i)$. This value is computed from the training set, by observing the proportion of documents relevant to the given query (as judged by the humans who constructed the training set) that contain A_i, the proportion of documents not relevant to the given query that contain A_i, dividing the former by the latter, and taking the log of this quotient. This value is the $(M+1)$th value in the row for the given query-document-composite clue triple. The entire set of such rows in the training set forms a matrix. If this training set matrix, or a matrix composed of a representative sample of rows drawn from the training set, is submitted to a statistical program package capable of performing ordinary linear regression, the package will compute values for the coefficients $c_0, c_1, \ldots, c_m$. Cooper et al. argue that the resulting linear function of the X_j with the c_j computed by the regression program will be a better predictor of $log\ O(R/A_i)$ than direct computation from the training set. Note that each coefficient, c_j, is the coefficient applied to any value of elementary clue type, X_j, e.g., if X_1 is relative frequency of a given match term in a given document, then c_1 is the coefficient applied to the relative frequency of any term in any document for purposes of computing its logodds of relevance to any query containing the same term.

At retrieval time, the system is given a new query Q against an operational document collection it is hoped will have similar statistical properties to the training set. Given any document D from this operational collection, the retrieval system can estimate $log\ O(R|A_i)$ of D for each of the A_i using the coefficients computed from the training set (as described above), and the values of the X_i obtained from the given document and query. The $log\ O(R)$ can be estimated straightforwardly from the proportion of relevant documents in the training set. Summing the $log\ O(R|A_i)$- $log\ O(R)$ contributions, the system obtains from the above equation an estimate for $log\ O(R|A_1, A_2, \ldots, A_n)$ for the given document. These logodds estimates can be used to relevance rank the retrieved documents relative to the given query. Moreover, a logodds estimate can be converted into a probability of relevance for the benefit of the end user.

In a subsequent TREC experiment [TREC-2], Cooper et al. employ a variant of the method described above. They call the earlier method the "triples-then-pairs" approach. The term "triples" refers to the fact that coefficients are computed separately for each of the composite clues, the A_i; each of the rows in the training set used to compute the coefficients for a given A_i corresponds to a query-document-composite clue triple, i.e., to the values of the elementary clues, the X_i, for a given A_i, obtained from a given query and document. By contrast, Cooper et al. call their TREC version the "pairs-only" approach. A single linear regression is applied to a set of rows

derived from the entire training set. Each row corresponds to a single query-document pair; hence, each row contains the *M* elementary clues, X_i, for each of the *N* composite clues, A_i. Therefore, the elementary clues have a double index, $X_{n,m}$, where *n* indexes the composite clue, and *m* indexes the elementary clues for a given composite clue.Hence, instead of obtaining coefficients for computing the logodds of relevance given a single A_i (as in the earlier "triples" approach), Cooper et al. obtain at once the coefficients for computing the logodds of relevance of any given document to any given query given *all* the A_i. In other words, they apply regression to obtain the coefficients for:

$$\log O(R|A_1, A_2, ..., A_N) = c_0 + c_1 \sum_{n=1}^{N} X_{n,1} + \ldots + c_M \sum_{n=1}^{N} X_{n,M}$$

As noted earlier, the effect of the "linked dependence" assumption (or the even stronger binary independence assumption) is to overstate the probability of relevance as *N*, the number of composite clues in the query that match the document being ranked, increases. In other words, the effect is to overstate the probability of relevance for high-ranking documents. In their TREC-2 work, Cooper et al. compensate for this by multiplying each of the $X_{n,m}$ terms by a "function *f(N)* that drops gently with increasing *N*, say $1/(\sqrt{N})$ or $1/(1 + \log N)$."

For the TREC2 experiment, three "composite" clues were employed: (1) normalized word stem frequency in query, (2) normalized word stem frequency in document, and (3) normalized word stem frequency in collection. Note that, whereas for the "triples-then-pairs" approach, each composite clue might be, e.g., a stemmed match word, and the facts comprising the composite might be its various (in this case, three) frequencies, in the "pairs-only" approach, each composite clue is a type of frequency, and the facts comprising a given composite are the values of the given frequency for each of the *M* match word stems. For word stem *m*, these were defined in TREC2 as follows:

1. $X_{m,1}$ = number of times the *m*-th word stem occurs in query, divided by (total number of all stem occurrences in query + 35);

2. $X_{m,2}$ = number of times the *m*-th word stem occurs in the document, divided by (total number of all stem occurrences in document + 80);

3. $X_{m,3}$ = log(number of times the *m*-th word stem occurs in the collection, divided by the total number of all stem occurrences in the collection).

Note that Cooper et al. assume above (in both the "triples" and "pairs only" methods) that a training set is available for a sample of "typical" queries, but not for the actual future queries for which retrieval is to be performed. Hence, regression coefficients, the c_i above, are computed for the elementary clues associated with each composite match term A_i, e.g., each word stem that matches in at least one query-document pair of the training set. When a document *D* is to be ranked against a new query, *Q*, the A_i employed are those which occur in both *Q* and *D*, i.e., they are match terms for the particular query-document pair being evaluated, not the complete set of match terms for the training set as a whole. (On the other hand, *Q* and *D* might contain a match term that never

occurred in the training set. Such a match term could not be used in computing the logodds of relevance of *D* to *Q*, because.the coefficients c_j would not be known for this "new" A_i.) That is, the predictors whose values are used to compute the logodds of relevance of a document *D* to a new query *Q* include the elementary clues, the $X_{n,m}$, for every term, e.g., every word stem, that occurs in both *Q* and *D*, and is also a match term of the training set. In the "triples" method, the value of each of those elementary clues, e.g., the relative frequency of a given word stem in *D* (that also occurs in *Q*), is multiplied by the coefficient for that type of elementary clue, as computed from the training set. Then these products are summed to compute the logodds of relevance for the given A_i, e.g., the given word stem. These logodds values are then summed to compute the value of the logodds of relevance of *D* to *Q*, given all the matching terms. In the "pairs only" method, the values of all the elementary clues of a given type, e.g., the relative frequencies of all match terms for *Q* and *D* are summed first, and then multiplied by the coefficient for that clue type.

By contrast, if training data is available for the actual queries to be evaluated operationally (which is commonly the case in routing applications), then, given one of those actual queries, *Q*, Cooper et al. [TREC-2] develop an equation that predicts the logodds of relevance of any given document *D* to *Q*. This equation is derived, as before, from their "fundamental equation." However now, each of the A_i corresponds to presence or absence of the *i*-th retrieval clue, e.g., the *i*-th term of *Q*, in *D*. Note that the odds of relevance for each term A_i in *Q* can now be estimated directly from the training set. If A_i is in *D*, one computes the ratio of relevant to non-relevant documents containing A_i. Similarly, If A_i is not in *D*, one computes the ratio of relevant to non-relevant documents *not* containing A_i. Hence, the composite clues A_i need not be expressed in terms of the elementary clues, as in the earlier cases. However, it may be that, in addition to the query-specific training set, there is also a larger non-specific training set for "typical" queries as before. In that case, the logodds of relevance of *Q* to *D* can be expressed as a linear combination of a query-specific function of all the A_i in *Q*, and a non-specific predictive function of the A_i in *Q* that are also in *D*, obtained by the earlier method where the queries to be evaluated operationally are not known. The non-specific component will be expressed in terms of elementary clues, as before. How should the query-specific and non-specific components be weighted? According to Cooper et al. [TREC-2], this remains a subject for research.

7.4.5 Okapi (Term Weighting Based on Two-Poisson Model)

Robertson et al. [SIGIR '94] has developed a term weighting scheme based on the Poisson distribution. This scheme was fist presented in the City University of London Okapi system. As it has proved to be one of the most successful weighting schemes in TREC competitions, it has been adopted by other TREC participants, and is generally identified by the system in which it was introduced, as Okapi weighting.

The Okapi approach starts with the view of a document as a random stream of term occurrences. Each term occurrence is a binary event with respect to a given term *t*. That is, there is a (typically small) probability *p* that the event will be an occurrence of *t*, and a probability $q = 1-p$ that the event is *not* an occurrence of *t*. Then, the probability of x occurrences (commonly called "suc-

cesses") of *t* in *n* terms is given by the binomial distribution. [Hoel, 1971] For very small *p* and very large *n*, the binomial distribution is well approximated by the Poisson distribution:

$$p(x) = e^{-\mu}\frac{\mu^x}{x!}$$

where μ is the mean of the distribution. To incorporate within-document term frequency *tf*, Robertson makes the fundamental assumption that the term frequency of a given term *t* is also given by a Poisson distribution, but that the mean of this distribution is different depending on whether the document is "about" *t* or not. It is assumed that each term *t* represents some "concept," and that any document in which *t* occurs can be said to be either about or not about the given concept. Documents that are about *t* are said to be "elite" for *t*. Hence, Robertson assumes that there are two Poisson distributions for a given term *t*, one for the set of documents that are elite for *t*, the other for documents that are not elite for *t*. (This is why the Okapi weighting is said to be a 2-Poisson model.) The Poisson distribution for a given term *t* becomes:

$$p(tf) = e^{m}\frac{m^{tf}}{(tf)!}$$

where *m*, the mean of the distribution, is either μ or λ depending on whether the distribution is for documents elite for *t* (mean = μ), or documents that are not elite for *t* (mean = λ). Note that these two Poisson distributions give the probability of a given term frequency for a given term *t* in terms of document eliteness to *t*, *not in terms of relevance to a given query.* A query can contain multiple terms. A document contains many terms, and may be about multiple concepts. The usual assumptions about term independence, or Cooper's "linked dependence," are extended to eliteness; that is the eliteness properties of any term t_i are assumed to be independent of those for any other term t_j.

Robertson defines the weight *w* for a given term *t* in terms of a logodds function:

$$w = \log\frac{p_{tf}q_0}{q_{tf}p_0}$$

where p_{tf} is the probability of *t* being present with frequency *tf* given that the document is relevant to a given query, and q_{tf} is the probability of *t* being present with frequency *tf* given that the document is non-relevant to the given query. The p_0 and q_0 are the corresponding probabilities with *t* absent. Hence, P_{tf}/P_0 is not the odds of *t* being present in a relevant document as before, but the odds of *t* being present with a given *tf* as compared to not being present in a relevant document at all. (And similarly, for q_{tf}/q_0 with respect to non-relevant documents.) When the Poisson distributions of *t* relative to document eliteness/non-eliteness given above are incorporated into this logodds function of *t* relative to document relevance/non-relevance, the result is a rather complex function in terms of four difficult-to-estimate variables: *p'*, *q'*, μ and λ. Here, *p'* is the probability

that a given document is elite for *t*, given that it is relevant, i.e., *P*(document elite for *t*|*R*). Similarly, *q'* = *P*(document elite for *t*| not *R*).

Robertson converts this difficult-to-compute term weight function into a more practical function His basic strategy is to replace complex functions by much simpler functions of term frequency that have approximately the same shape, e.g., the same behavior at *tf*=0, the same behavior as *tf* increases, and grows large, etc. His approximation starts with the traditional logodds function for presence/absence of *t*, as derived from the relevance/non-relevance contingency table in 7.4.1 (Binary independence). This is multiplied (in effect, "corrected" or "improved") by a simple approximation function for term weight in a document as a function of *tf*, a function that approximates the shape of the true 2-Poisson function. The approximation contains a "tuning constant," k_1, in the denominator, whose value (determined by experimentation) influences the shape of the curve. Then, the weight function is multiplied by a similar approximation function for the query, i.e., a function of within-query term frequency, *qtf*. This function also contains a tuning constant, k_3.

To improve the approximation further, Robertson takes document length into account. He offers two broad hypotheses to account for variation in document length: The "Verbosity hypothesis" is the hypothesis that longer documents simply cover the same material as corresponding shorter documents but with more words, or (more fairly) cover the same topic in more detail. (This is the hypothesis that underlies most document vector normalization schemes discussed above.) The "Scope hypothesis" is the hypothesis that longer documents deal with more topics than shorter documents. (This is the hypothesis that underlies most work with document "passages.") Obviously, each hypothesis can be correct in some cases, and indeed, in other cases, both hypotheses may be correct, i.e., a document may be longer than another both because it uses more words to discuss a given topic, and because it discusses a greater number of topics. Hence, Robertson refines his approximation to allow the user to take either or both hypotheses into account, as appropriate. First, on the basis of the Verbosity hypothesis, he wants the weight function to be independent of document length. On the simple common assumption that term frequency is proportional to document length, he multiplies k_1 by dl_i, the length of the *i*-th document, the document D_i under consideration, so that all terms will increase proportionally with document length, and the weight function will remain unchanged. Then, on the assumption that the value of k_1 has been chosen for the average document, he further normalizes k_1, dividing it by dl_{avg} the average document length for the collection under consideration. Then, he modifies this normalization factor with another tuning constant, *b*, into a composite constant $K = k_1((1-b) + b(dl_i/dl_{avg}))$. The constant, *b*, also determined by experiment, controls the extent to which Verbosity hypothesis applies (*b*=1) or does not apply (*b*=0).

To compute document-query similarity for a given document, D_i, the term weights determined by the above approximation function are added together for all query terms that match terms in D_i. Finally, to this sum Robertson adds a "global correction term" that depends only on the terms in the query, and not at all on whether they match terms in D_i. This correction term reflects the influence of document length variation, departure from the average length, with respect to the weight of each query term. The correction term contain yet another tuning constant, k_2.

The final result, first used in TREC3 [Robertson et al., 1995] and TREC4 [Robertson et al., 1996], is called BM25; the BM stands for "Best Match" and the 25 is the version number, reflecting the evolution of this term weighting scheme. The BM25 function for computing the similarity between a query Q, and a document D_i is:

$$S(Q, D_i) = \sum_{t \in Q} \left(\log \frac{(r + 0.5)/(R - r + 0.5)}{(n - r + 0.5)/(N - n - R + r + 0.5)} \right) \frac{(k_1 + 1)tf}{K + tf} \frac{(k_3 + 1)qtf}{k_3 + qtf} + k_2 |Q| \frac{avdl - dl}{avdl + dl}$$

where

Summation is over all terms t in query Q
r = number of documents relevant to Q containing term t
R = number of documents relevant to Q
n = number of documents containing t
N = number of documents in the given collection
tf = frequency (number of occurrences) of t in D_i
qtf = frequency of t in Q
$avdl$ = average document length in the given collection
dl = length of D_i, e.g., the number of terms, or the number of indexed terms, in D_i
$|Q|$ = number of terms in Q
k_1, k_2, k_3, and K are tuning constants as described above.
K = $k_1((1\text{-}b) + b(dl_i/dl_{avg}))$ where b is another tuning parameter.

Varieties of Okapi BM25 have continued to be used down through TREC-9, both by its originators [Robertson et al., 2000] and others, due to its effectiveness. According to Robertson et al., "k_1 and b default to 1.2 and 0.75 respectively, but smaller values of b are sometimes advantageous; in long queries k_3 is often set to 7 or 1000 (effectively infinite)." k_2 has often been set to 0, e.g., in TREC-4 and TREC-9.

8. Routing/Classification Approaches

In theory, the "routing" or "classification" problem is identical to the information retrieval problem: to identify documents that match, i.e., are relevant to, a specified query or information need. Hence, in principle the same methods are applicable to both problems. [Belkin & Croft, CACM, 1992] However, the practical differences between the two problems affect which methods are practical for each.

In information retrieval, the user has at any given time one or more relatively static collections. The collections may be updated, but not so rapidly as to change their basic, e.g., statistical or semantic, properties overnight. New collections may come on-line, but they will have the same relatively static characteristics as the existing collections. The user generates many queries against these collections. In other words, the collections are relatively static; the queries are not.

In the classification environment, there is no fixed collection. Instead, there is a steady (perhaps high volume) stream of incoming documents. There is a well-defined set of topics of interest, or a well-defined set of users, each with his own well-defined set of interests and concerns. The problem is to classify each document according to which topic(s) it is "about" or which user(s) the document would interest, and then route the document to the appropriate "bin(s)." Documents that are not about any topic of interest are thrown away. The set of documents to be classified and routed is not static at all; rather it is constantly changing. Moreover, these documents are not available initially for the purposes of studying their statistical or other properties. Rather, they will arrive over a (perhaps long) period of time. On the other hand, the set of user needs are presumed to be (relatively) stable [Belkin & Croft, CACM, 1992], although new needs and users will arrive over time, and old needs will become obsolete. Hence, the queries/information needs are relatively static; the document population is not.

The term "routing" is often applied to a classification system in which there is only a single topic of interest. Hence, the objective is to distinguish and pass on all documents relevant to the given topic, and discard all other documents. In that case, the router is often called a "filter". Filtering can be "negative" as well as positive, i.e., the purpose of a user's "profile" may be to specify "junk" that he wants to throw away. [Belkin & Croft, CACM, 1992]

Of course, this distinction between routing and information retrieval is an idealization. In practice, the distinction is not necessarily so clear-cut. The collections to which information retrieval is applied may not be as static as one would wish. The information needs in the routing application may also change rapidly. Routing and information retrieval may be viewed as opposite ends of a spectrum with many actual applications in between.

In the routing application, one does not start with a large static collection of actual documents to which classification and routing are to be applied. Hence, it is common to employ a "training set" of documents which are (it is hoped) statistically typical of the documents to be encountered in practice. The routing system is trained against these documents. Training a system against a "training set" is analogous to expanding or refining a query with relevance feedback. In the latter case, the retrieved documents judged relevant are, in effect, the "training set". In the former case, the user supplies relevance judgments for the documents in the training set, e.g., document D_1 is relevant to class C_1, document D_2 is relevant to classes C_1 and C_2, D_3 is non-relevant, etc. The effect of training is to build a query or set of queries that classify the incoming documents correctly. Hence, the desired effect is that the query for class C, will match or rank documents according to their degree of relevance to class C.

The biggest practical difference between routing and information retrieval is that in routing, the training, i.e., the query expansion and refinement, are performed in advance, i.e., before the system goes "operational." Hence, computationally expensive and time-consuming training methods that would not be acceptable in real time become practical. (In short, routing permits preprocessing, perhaps slow, of the relatively static queries. In contrast, ad hoc querying permits preprocessing, perhaps slow, of the relatively static document collections.) The essential requirement is that once the system has been trained, the amount of time required to perform the classification at run time is moderate. Moreover, even at run time, the user is not sitting at his screen waiting for a response to a query he has just issued. Hence, response time may not be as critical an issue as in

information retrieval. However, the volume of documents that a routing application must classify may be very large in some applications. In such cases, it may be impossible for the system to "keep up" with the traffic volume unless the run-time classification algorithm is very fast.

Even during a pre-operational training phase, a very large "feature space" can present problems. As noted above, classification or learning algorithms break down if the number of features required for classification, the number of dimensions of the feature space, is very large. Hull [SIGIR '94] notes that "[i]f there are too many predictor variables [i.e., features used to classify the documents], the classifier will overfit the training data … there must be significantly fewer predictor variables than relevant documents before it is possible to obtain good estimates of the parameters in the classification model." Similarly, Schutze et al. [SIGIR '95] consider "classification techniques which have decision rules that are derived via explicit error minimization … Error minimization is difficult in high-dimensional feature spaces because the convergence process is slow and the models are prone to overfitting." In particular, if the feature space consists of all significant terms in a given collection, or even in the relevant documents of a collection, the number of features will certainly be far too large. Hence, even though the training is performed "off-line," i.e., before classifying the actual traffic, methods to reduce the number of features are essential. Note that overfitting is the same problem that Buckley and Salton [SIGIR '95] encountered when they used relevance feedback to expand queries, thereby increasing greatly the number of query terms, i.e., "features." They developed their technique of Dynamic Feedback Optimization to avoid that problem (see above).

Two methods [Schutze et al., SIGIR '95] have been used to reduce the dimensionality of the feature space: reparameterization (which replaces the original feature space by a lower dimension feature space derived from the original features), and feature selection (which selects from the complete set of features a small subset of the "best" features, i.e., the ones most likely to distinguish relevant from non-relevant documents). A popular method of reparameterization (LSI) is discussed above. Methods of feature selection (more specifically, term selection but the methods are generalizable to other statistical features) are discussed above in the section on query expansion.

Another problem with large feature spaces or computationally expensive classification algorithms is that the "real world" that generates the documents to be classified may change rapidly, resulting in a document population with rapidly changing statistical characteristics. Hence, it may be necessary to re-train the system frequently, which becomes unacceptably expensive if the training algorithm is very slow or requires enormous computational resources. This issue seems to have been largely ignored in the literature, which generally assumes that user needs and the statistical characteristics of the document population to be classified are stable.

If the population of incoming documents is completely stable, then the training set (if it is a representative sample of that future population) is sufficient to train the classifier. If the training set is imperfect (or non-existent), relevance feedback can be applied to the accumulating collection of documents. [Buckley et al., SIGIR '94] Documents that have satisfied a user can become, over time, a very large and effective training set for that user or for a given topic. [Belkin & Croft, CACM, 1992] "Over the life of the query [i.e., a long-standing information need], thousands of

documents could conceivably be returned to the user for relevance judgments." [Buckley et al., SIGIR '94]

On the other hand, Lewis [SIGIR '95] is one of the few to consider the case where "the classifier is applied to time-varying data such as news feeds or electronic mail." In such a case, *relevance feedback* (i.e., query refinement and expansion based on user identification of those documents retrieved by the original query which are relevant to her need - see below for a more detailed discussion) can be applied to the incoming documents to update the query, enabling it to track a population of incoming documents whose statistical characteristics are (slowly) changing. (Buckley et al. consider the application of relevance feedback to a routing application, but they do *not* specifically address the issue of changing statistical properties; they assume that the purpose of continuing feedback is to enable the classifier to approach more closely a fixed target.) Lewis *does* consider time-varying data but he explicitly does *not* consider relevance feedback. Instead, he considers the case where the classifiers are "autonomous" systems, i.e., there is little feedback from end users, and hence the systems must estimate their own effectiveness and re-optimize themselves as the incoming data changes. He assumes probabilistic classifiers, i.e., classifiers that, given a document, output both a classification and a probability that the document belongs in that classification, given the set of features being used for classification. His method is to specify an "effectiveness" measure for the classifier as a function of the classifications and associated probabilities assigned to N previous documents. The classifier then re-tunes itself to maximize the effectiveness measure. Lewis studies three possible effectiveness measures.

Yu et al. [CIKM 98] address the issue of routing a stream of (possibly) time varying data, in an interactive environment where each incoming document judged relevant by the system is presented to the user for a possible relevance judgment. Their "adaptive text filtering" algorithm maintains a pool of terms, $Pool_R$, that have occurred in documents judged relevant by the user, and another pool of terms, $Pool_N$, that have *only* occurred in documents judged non-relevant. (In other words, a term that has occurred in relevant documents will be in $Pool_R$, regardless of whether it has also appeared in non-relevant documents. By contrast, a term will be in $Pool_N$ only if it has appeared in non-relevant documents, and has never appeared in relevant documents.) An incoming document is retrieved and presented to the user as possibly relevant under two conditions: (1) The document is retrieved if the sum of the weights of terms in the document that are *also* in $Pool_R$ exceeds a specified threshold. The weight of a term in this calculation is actually the sum of two weights, its *feature* weight, based on all the documents in which it has occurred so far that have been presented to the user and judged relevant, and its *document* weight, its weight in the current document computed by the traditional *tf***idf* function. (2) The document is also retrieved if the proportion of *new* terms in its document vector plus the proportion of terms in the document vector that are also in $Pool_R$ exceeds a specified threshold. A *new* term is a term that is neither in $Pool_R$ or $Pool_N$. In other words, a document is presented to the user as a relevance candidate according to the 2nd criterion if some proportion of its terms have already been seen in documents he has judged relevant, *and* a proportion have not been seen at all. A weighting factor is used to determine the relative importance of presence in $Pool_R$ versus novelty. $Pool_R$ and $Pool_N$.are updated based on the user's relevance judgment. The feature weights of terms in the document are also updated based on the user's relevance judgment. (Note that the presence of many novel terms can cause a document to be presented to the user, but only his relevance judgment can cause the feature weights and pool contents to be updated.) The weight of a given term is increased if the

document was retrieved on the basis of the 2nd (novelty) criterion, and the user judges it relevant. The weight of a term is decreased if the document was retrieved on the basis of the first (relevance) criterion, and the user judges it non-relevant.

Leaving the realm of adaptive classification methods applied to time-varying data, let us consider methods of classification where the training set is static. As noted above, the first consideration is how to limit the feature space. Once the feature space has been reduced sufficiently, a variety of classification training or learning methods are available. Let's consider some of these methods briefly.

Relevance feedback weighting (e.g., the Rocchio formula discussed in a later section), can be applied to the training set. If nothing is known about the classification weights initially, then the weight of the original "query" to be refined or expanded by feedback in the can be set to zero. In other words, we are using relevance feedback here to *generate* the classification "query" rather than to *refine* it as in an ad hoc query application. Schutze et al.[SIGIR '95] tried Linear Discriminant Analysis (LDA), Logistic Regression, and Neural Networks. In contrast to Rocchio, all three of these methods "have decision rules that are derived via explicit error minimization." Note that logistic regression and Rocchio formulas both have constant parameters that need to be determined but with logistic regression, the parameters are computed directly to optimize the formula whereas in the Rocchio approach, the parameters (*A, B,* and *C*) can only be determined by systematic trial and error. LDA classifies the population into two (or more) distinct groups. The separation between the groups is maximized by maximizing an appropriate criterion, e.g., the "separation of the vector means" of the groups. [Hull, SIGIR '94] For the present case, LDA computes a linear function *z* of the document descriptors that distinguishes the two groups of interest, relevant documents and non-relevant documents, "as widely as possible relative to the variation of values of z within [each of] the two groups." [Hoel, 1971] Again, this is a direct rather than a trial and error procedure. Schutze et al. found in their experiments that all three of these "classifiers perform 10-15% better than relevance feedback via Rocchio expansion for the TREC-2 and TREC-3 routing tasks." Hull [SIGIR '94] also employed discriminant analysis. Hull applied this technique to a feature space whose dimensionality was reduced using LSI. As a further refinement, LSI was applied not to the entire collection used for training but to the set of relevant documents for a given query. Hence, LSI must be applied separately for each query but the document term matrix to which it is applied in each case is much smaller than the matrix for the entire collection.

Dumais et al. {CIKM '98] compared five classification methods that work by learning from a training set: Rocchio relevance feedback, decision trees, Naive Bayes, Bayes Nets, and Support Vector Machines (SVM). They trained on the "so-called Reuters-21578 collection," a collection of news stories. They "used 12,902 stories that had been classified into 118 categories (e.g., corporate acquisitions, earnings, money market, grain, and interest)." (Note that news stories, though an important, and widely used, type of text corpus, are in some respects particularly easy, because of their relatively standard organization, conventions, and vocabulary.)

Dumais et al. represented the documents in the traditional way as vectors of words. Then, they performed feature selection to reduce the dimension of the vectors. They used mutual information, $MI(x_i, c)$ as the feature selection measure, where x_i is the *i*-th feature, and *c* is the category

for which the various classifiers are being trained. (See the section on Logical Imaging for the definition of "mutual information.") They selected the 300 "best" features (highest MI value) for SVM and decision trees, and 50 features for the other three classification methods.

Since Rocchio's method is here being used for classification rather than query refinement, there is no "initial query" term, as noted above. Dumais et al. also elected to discard the negative examples, i.e., the documents that were not relevant to the given category for which the classifiers were being trained. Since the training set starts with relevance judgments for each category, there are no interactive relevance judgments. Hence, the Rocchio formula for a given category was reduced to computing the centroid (the average) of the documents labeled relevant to the given category. At test time, a new document was judged relevant to a given category if its similarity to the category's centroid (as measured by the Jaccard similarity measure) exceeded a specified threshold.

Lewis and Gale [SIGIR '94] use a variation on traditional relevance feedback which they call "uncertainty sampling." In any situation where the volume of training data is too large for the user to rate all the documents, some sampling method is required. In traditional relevance feedback, the sample the user is asked to classify consists of those documents that the current classifier considers most relevant. Hence, Lewis and Gale call this approach "relevance sampling". It has the notable virtue, especially if the relevance feedback is taking place while the system is operational, that the documents the user is asked to classify are the ones that (as far as the classifier can tell) he wants to see anyway. However, if the training is taking place before the system is operational (or in a very early stage of operation) and the primary objective is to perfect the classifier, then uncertainty sampling (derived from "results in computational learning theory") may work better. The method assumes a "classifier that both predicts a class and provides a measurement of how certain that prediction is. Probabilistic, fuzzy, nearest neighbor, and neural classifiers, along with many others, satisfy this criterion or can be easily modified to do so." The sample documents chosen for the user to rate are those about which the classifier is most uncertain, e.g., most uncertain whether to classify them as relevant or non-relevant. For a probabilistic classifier (such as the one they actually describe and test in their paper), the most uncertain documents are those that are classified with a probability of correct classification close to 0.5. Lewis and Gale obtained substantially better classification for a given sample size when the classifier was trained by uncertainty sampling of the training set than when it was trained by relevance sampling (and far better than with training on a random sample).

Yang [SIGIR '94] addresses the classification problem that there is often a wide gap between the vocabulary used in documents to be classified and the terms used in the class or topic (here called "category") descriptions, i.e., the "queries." The training set consists of documents to which users have manually assigned category descriptions. (Yang is using a medical application so examples of category descriptions are "acquired immunodeficiency syndrome" and "nervous system diseases".) A given category may be assigned to many documents. More surprisingly, the same document, e.g., a common diagnosis, may occur multiple times in the training set. Even more surprisingly, the same category may be assigned multiple times to the "same" document; the way this comes about is that the category is assigned to two distinct documents which become identical as the result of aggressive application of a stoplist. The problem is to classify, i.e., assign appropriate category descriptions to, a new document. Yang's approach consists of two stages. In the first stage, she computes the conventional cosine similarity between the new document to be

classified and the documents in the training set, e.g., the similarity $sim(X, D_j)$ between the new document X and a training document D_j. In the second stage (the novel part of the method), she estimates the conditional probability $Pr(c_k|D_j)$ that a given category c_k is relevant to a training document D_j. $Pr(c_k|D_j)$ is estimated as the "number of times category c_k is assigned to document D_j" (see above) divided by the "number of times document D_j occurs in the training sample". Then $sim(X, D_j)$ is multiplied by $Pr(c_k|D_j)$ for each D_j and this product is summed over the N top-ranking D_j's, i.e., the ones most similar to X. Experimentally, Yang found that the optimum value of N for her collection was $N = 30$. The result is $rel(c_k|X)$, a relevance score for c_k.These scores are not probabilities but they provide a ranking of categories for the given document X similar to what would be obtained with probabilities. This ranking can then be used to assign the highest ranking categories to document X.

DR-LINK [Liddy et al., ACMIS, '94] deals with an issue that arises whenever a collection or stream of documents must be categorized and routed according to multiple topics. The distribution of relevant documents with respect to computed topic similarity scores will tend to vary widely from one topic to another. For example, suppose that a categorization engine ranks a document population with respect to two topics, T_1 and T_2, producing two separate document rankings, one with respect to T_1 and the other with respect to T_2. In the T_1 ranking, 95% of the documents actually relevant to topic T_1 (as judged by human users) may be found in the top-ranking 5%. On the other hand, in the T_2 ranking, it may be necessary to traverse the top-ranking 35% to find 95% of the documents actually relevant to topic T_2. Hence, quite different topic similarity thresholds are required for T_1 and T_2 respectively. DR-LINK deals with this problem by developing a multiple regression formula on document training sets for each of the topics to be categorized. (Clearly, this approach only applies to an application such as routing, where a fixed number of topics will be applied to a changing population of documents, and a training set of typical documents is available for each topic.) The regression formula for each topic has two independent variables, (1) the desired recall level, e.g., 95% in the example above, and (2) the top-ranked document's similarity. The dependent variable returned by the regression formula is the estimated topic-document similarity score threshold needed to achieve the desired level of recall. Hence, a different threshold can be computed for each topic. In tests involving TREC-2 topics and data (173,255 Wall Street Journal articles from 1986-1992), this formula proved quite effective at computing a threshold appropriate for a given topic and desired recall level. Moreover, it was found that because a few relevant documents ("stragglers") tend to be low-ranking, the number of documents that need to be examined can be drastically reduced by lowering the recall level from an unrealistic 100% to a more realistic 80%;

9. Natural Language Processing (NLP) Approaches

The phrase "Natural Language Processing" (NLP) approaches" to IR refers here to all methods based on knowledge of the syntax and/or semantics of the natural language in which document text is written, or knowledge of the world, e.g., the application domains, to which the documents refer. Hence such approaches may also be broadly characterized as *semantic* approaches, in the sense that they attempt to address the structure and meaning of textual documents directly, instead of merely using statistical measures as surrogates. However, as discussed below, there are three sources of terminological confusion. First, the term "semantic" is sometimes used to refer to one particular level of NLP, although in reality it is applies to (at least) five different levels. Secondly,

many NLP techniques, especially in the realm of “shallow” (and correspondingly, computationally efficient) NLP methods employ statistical techniques, e.g., to determine the most likely sense or part of speech of a given word in a given context. Third, NLP techniques are rarely used by themselves in IR. More commonly, they are used to supplement statistical techniques.

Human beings find it amazingly easy to assess the relevance of a given document based on syntax and semantics. They find statistical and probabilistic methods much more difficult, tedious and error prone. For automated systems, the situation is the reverse. They can perform statistical calculations easily. Developing automated systems that can understand documents in the syntactic/semantic sense is much more difficult. As a result, most IR systems to date have been based on statistical methods. Increasingly however, syntactic and semantic methods are being used to supplement statistical methods. The reason is plain. Even the best statistical or probabilistic methods will miss some relevant documents and retrieve some (often quite a bit of) junk. The hope is that an appropriate combination of traditional statistical/probabilistic methods and syntactic/semantic methods will perform better than the statistical methods alone. Ideally, the combination would approach human performance. This ideal is a long way from realization. [Faloutsos & Oard, UMd-CS, 1995].

Note, by the way, that a technique like Latent *Semantic* Indexing (LSI) (discussed above) is not a semantic method in the sense used here despite the presence of the word “semantic” in its name. Rather, it is a statistical method for capturing term dependencies that it is hoped have semantic significance.

Liddy [BASIS, ‘98] classifies NLP techniques according to the level of linguistic unit processed, and (correspondingly) the level and complexity of the processing required. She identifies the following levels: phonological, morphological, lexical, syntactic, semantic, discourse, and pragmatic. The *phonological* level is the level of interpreting speech sounds, e.g., phonemes. It is mainly of interest in speech to text processing, rather than textual IR.

Several traditional IR techniques do use NLP techniques, almost entirely at the *morphological* and *lexical* levels. The morphological level is concerned with analysis of the variant forms of a given word in terms of its components, e.g. prefixes, roots, and suffixes. Hence, traditional stemming techniques that reduce variants of a word to a common root form for query-document term matching, exemplify morphological IR processing. The lexical level is concerned with analysis of structure and meaning at the purely word level. For example, traditional lexical IR processing includes construction of stop lists of words believed to have low semantic content. [Faloutsos & Oard, UMd-CS, 1995] (But see below!) Similarly, generation and use of thesauri for query expansion, and of controlled vocabulary lists for indexing and query formulation, are other traditional examples of lexical IR processing. Proper noun identification is another, somewhat newer, form of IR lexical processing. Tagging words with their parts of speech is also a kind of lexical processing, common and well-established in NLP, but rare in traditional IR.

The *syntactic* level is the level at which the syntactic structure of sentences is determined, in terms of the parts of speech of the individual words. In practice, a single sentence can have many possible structures. Determining the correct structure from these alternatives requires knowledge at the higher levels (or statistics based on a training set). For this reason, and more generally because it

is relatively expensive computationally, syntactic level processing has been little used in traditional IR. Some use of syntax has been made to identify units larger than single words, e.g., phrases, but even here, statistical co-occurrence and proximity rather than NLP, have been the preferred methods in IR.

The *semantic* level is the level at which one tries to interpret meaning at the level of clauses, sentences, rather than just individual words. Note that the disambiguation of words having multiple senses is a semantic-level task, because a word can only be disambiguated in the context of the phrase, sentence, or sometimes larger text unit in which it occurs (and also because disambiguation may require real-world knowledge, generic, user-specific, or domain-specific). Because of the difficulty and NLP sophistication required (and the need for the aid of the higher levels), traditional IR has avoided semantic-level processing in favor of statistical keyword matching, as discussed in earlier sections.

The *discourse* level is the level at which one tries to interpret the structure and meaning of larger units, e.g., paragraphs, whole documents, etc., in terms of words, phrases, clauses, and sentences.

The *pragmatic* level is the level at which one applies external knowledge (that is, external to the document and original query). The knowledge employed at this level may include general knowledge of the world, knowledge specific to a given application domain, and knowledge about the user's needs, preferences, and goals in submitting a given query.

The most important source of semantic content in traditional IR is relevance feedback, the refinement and expansion of a query based on human judgments of which of the documents retrieved by the query are relevant to the given query. Naturally, these human judgments are based on the user's understanding of the semantic content (in the broadest sense) of the retrieved documents, and her understanding of her actual needs. Hence, the feedback is implicitly at the higher NLP levels. However in traditional IR, this feedback is typically used merely to improve statistically the set of term descriptors and the weights assigned to those descriptors. Only the human user "understands" or "processes" the documents at any semantic or natural language level. However, the methods discussed below can be extended to relevance feedback in various ways, e.g., the descriptors extracted from documents identified as relevant can be higher-level linguistic units such as phrases, or even concepts that do not actually appear in the documents themselves.

This section discusses research into the application of the higher levels of NLP, i.e., syntactic, semantic, discourse, and pragmatic, to the classic problems of IR. It also discusses advances at the lexical levels, e.g., improved proper noun recognition and classification.

It should be stressed that, almost without exception, the NLP methods discussed below are used in conjunction with, not in place of, traditional boolean, vector, and statistical term weighting techniques for document-topic matching and document categorization. [Lewis et al, CACM] Semantic methods can be used to extend the terms to which matching is applied from keywords to key phrases. They can be used to disambiguate terms that have multiple meanings, or fit multiple parts of speech. They can be used to map keywords, phrases and proper nouns into conceptual terms, e.g., subject category terms, that express more naturally the user's interests, but that will not necessarily co-occur in both the user's topic statement and the relevant documents. They can be used

to supplement the terms in a user's query with candidate synonyms. They can be used to identify semantic relationships among the keywords or phrases occurring in a topic, or in a candidate document. Hence, topics and documents can be matched not only on whether the specified keywords occur in both, but on whether they occur in the same (or similar) relationship in both topic and document. Semantic methods permit the identification of relationships other than the purely boolean, e.g., given the keywords "company" and "investigation," semantic methods can distinguish a query about a company performing an investigation from a document about a company being investigated. Note that statistical or user-specified weights can be applied to all of these semantically derived terms, i.e., to phrases, conceptual terms, synonyms, relationships, etc.

Semantic methods can significantly enhance term normalization techniques. Term normalization reduces query and document descriptor terms to a common form for matching purposes. Stemming is the most common type of normalization in traditional IR systems. Another traditional technique is the manual assignment of index terms to documents from a controlled vocabulary. Semantic methods permit more sophisticated forms of automated normalization. Varied syntactic forms can be mapped to a standard syntax, e.g., "investigation by the company," and "the company is investigating" can be mapped into the common noun phrase "company investigation." Related words, e.g., house, apartment, and hut, can be mapped into a common subject category, "dwelling." Varied forms of a proper noun can be mapped into a standard form. And so on.

9.1 Phrase Identification and Analysis

A common use of syntactic (and to some degree semantic) methods is phrase identification. [Riloff, SIGIR '95] [Jacqemin & Royaute, SIGIR '94] [Kupiec, SIGIR '93] [Anick & Flynn, SIGIR '93] [Strzalkowski & Carballo, TREC-2] [Evans & Lefferts, TREC-2] Phrases are typically identified in IR so that they can be used as descriptor terms, i.e., so the descriptors of a document are not limited to single words. Traditional methods identify phrases by statistical co-occurrence, e.g., co-occurrence of pairs of terms in documents at a rate greater than would be expected by random chance. Co-occurrence can be combined with adjacency, e.g., if two (or more) terms co-occur within a few words of each other at a rate greater than chance, the probability that they are related semantically certainly increases. If the terms in question are also related syntactically, the chance that they form a phrase is still greater. Syntactic analysis can identify phrases even when the terms of which they are composed are not adjacent, or do not co-occur with greater than chance frequency. However, extraction of phrases by purely syntactic means alone is seldom effective since it is likely to extract many phrases of little value for characterizing the topic(s) of a given document or query. [Croft et al., SIGIR '91] A combination of syntactic and statistical methods is more effective. {Lewis, SIGIR '92] [Lewis et al., SIGIR 96] Lewis et al. suggest that statistical weighting techniques should be applied to phrase descriptors, even if they are generated by NLP or combined NLP/statistical techniques. However, they suggest that "[w]eighting for phrases may differ from weighting for single-word terms to allow for their lower frequency and different distribution characteristics."

Lewis et al. urge that phrase descriptors should be "linguistically solid compounds," e.g., noun phrases, etc. However, they also stress that phrase matching should reflect the variety of forms a phrase can assume, and the varying degrees of evidence provided by each form. For example, if the noun phrase "prefabricated units" is extracted from a document, it is likely that it signifies the

corresponding concept. The presence in a given document of the verb phrase “[they] prefabricated units” would provide weaker evidence for the presence of the concept. The co-occurrence in a given document of the two words “prefabricated” and “units” in close proximity, e.g., in the same sentence or paragraph, but not in the same syntactic phrase, would provide still weaker (but not non-zero) evidence that the concept was present. The co-occurrence of the two words in different paragraphs of the given document would provide much weaker evidence still, and so on.

The above example also illustrates that NLP identification and extraction of phrases can have an important effect on the traditional approach of word stemming. A stemmer would reduce “prefabricated” and “prefabricate” to the root “prefabricat.” But, as we see above, the distinction between “prefabricated” and “prefabricate” (or prefabricating, for that matter) may be the difference between a noun phrase and a verb phrase, and hence a corresponding difference in the evidence that the respective phrases contribute about the topic(s) discussed by a document that contains them.

Lewis et al. also note that the degree of sophisticated NLP and statistical processing applied to extraction of phrases and other compound terms might be considerably greater for user queries than for documents. There are several reasons for this: First, the number of queries is normally very much less than the number of documents, so that an IR system can afford to process the queries more carefully. Second, it is very important to understand what the user’s requirements are; this is complicated by the fact that queries are generally much shorter than documents, and often more carelessly formulated by users who are not professional IR searchers. (For the same reasons, it is very desirable to support interactive query refinement, using thesauruses, NLP analysis of user queries, relevance feedback with regard to both good terms and good documents, as assessed by the user, etc.) Third, any error in extracting phrases and compound terms from documents can be corrected (or at least compensated for) during the query-document matching process, since the matching process will take into account not merely a single phrase that may have been extracted from a given document incorrectly, but the context of other words and phrases that have been extracted from the same document.

The burden of applying expensive NLP to large document collections can be further eased by a two-step process: First, coarse ranked retrieval of candidate documents using statistical and shallow NLP techniques. Second, more sophisticated NLP applied only to the much smaller list of highly ranked documents retrieved by the first stage.

Shallow or coarse NLP refers to techniques for extraction, based on local contexts, of noun and verb phrases, compound proper nouns (discussed later), simple basic propositions, e.g., produce(factory, house), complex nominals, e.g., “debt reduction,” “health hazards,” and simple concept-relation-concept (CRC) structures, e.g., the sentence fragment “the company’s investigation of the incident...” generates two CRC triples, each relating a noun to the verb: [investigate] -> (AGENT) -> [company], and [investigate] -> (PATIENT) -> [incident], which convey the information that the company acts in the investigation, and the incident is the object of the investigation. CRC triples and structures constructed from them are discussed further below. Here, it should be stressed that shallow, localized extraction of such “solid” linguistic components is contrasted with complete parsing of sentences and paragraphs, a very much more difficult and expen-

sive process, requiring the support of elaborate knowledge bases (KB's) and typically generating multiple legal parses.

Lewis et al. further argue that, although NLP may be used to supplement word descriptors with phrase and compound term descriptors, these compound term descriptors should *not* be combined into higher level structured descriptors, e.g., frames, templates. Note that they do not object to such higher-level descriptors for *knowledge* representation and retrieval. Rather, they object to them as index descriptors for document retrieval. Their argument is that such higher-level structures are labor-intensive to produce, and that an exact query-document match at the level of such elaborate structures is very unlikely. In subsequent sections, there is a discussion of certain higher-level document structures. In particular, there is a discussion of exciting work in the area of document *discourse structure*, the structure of clauses, sentences, and paragraphs that determines the narrative or expository flow of a document. It is this discourse structure that enables a reader to follow the flow of the writer's argument, and understand what the writer is saying. Some research indicates that combining term descriptors with this discourse structure can enhance document retrieval. It can also be used for forms of knowledge retrieval, e.g., document summarization. But note that the discourse structure is not a structure composed of conventional document descriptors such as words and phrases. Rather, it is a wholly separate, higher-level structure of the document as a whole. The words and phrases are related to the discourse structure either by being located *in* identifiable components of the discourse structure, or by serving as linguistic clues *to identify* the components of the discourse structure.

Strzalkowski and Carballo [TREC-2] extract phrases syntactically from a large collection, but then apply a variety of statistical techniques to these phrases before formulating queries.

> The extracted phrases are statistically analyzed in syntactic contexts in order to discover a variety of similarity links between smaller subphrases and words occurring in them. A further filtering process maps these similarity links onto semantic relations (generalization, specialization, synonymy, etc.) after which they are used to transform a user's request into a search query.

As an example of "statistical analyz[ing] in syntactic contexts," they filter out poor terms with a statistical measure called Global Specificity Measure (GTS) "similar to the standard inverted [sic] document frequency (*idf*) measure except that term frequency *is measured over syntactic units* [e.g., phrases such as verb-object, noun-adjunct, etc.] *rather than document size units*." (italics mine). (See below for further discussion of GTS.) As an example of using a derived semantic relation "to transform a user's request into a search query," they offer the example of adding the compound term "unlawful activity" to "a query … containing the compound term 'illegal activity' via a synonymy link between 'illegal' and 'unlawful.'"

Note that the above example illustrates another use of combining syntactic with statistical techniques: the automated creation of thesauri. Traditionally, thesauri have been constructed either manually, e.g., WordNet, or statistically. (Because of the many semantic relations it supports, WordNet, described below, is more properly described as a semantic network.) Stzalowski and Carballo illustrate that thesaurus type information (not only synonymy but other semantic rela-

tions like specialization and generalization) can be derived by applying statistical techniques, e.g., co-occurrence, similarity, etc.) to syntactic units derived by automated parsing (and not only to the syntactic units themselves but to their internal structure). Note that even when the thesaurus generation, e.g., determination of synonymy, is wholly statistical [Evans & Lefferts, TREC-2] [Milic-Frayling et al. TREC-4], the extraction of candidate phrases syntactically allows these phrases to be units to which the statistical methodology can be applied.

Phrases are commonly extracted to serve as document (query) descriptors, i.e., as index terms. However, they are also used in a variety of other, related ways. For example, phrase extraction is used as a key feature of a system, MURAX [Kupiec, SIGIR ‘93], that answers natural language questions that require a noun phrase as answer. The answers are obtained from a large, general purpose, on-line encyclopedia. The method does not depend on domain-specific knowledge; the questions can be on any topic whatever, as long as they require noun phrase answers. But the method does make use of other simple semantic and syntactic heuristics, e.g., the question can begin with “who”, “what”, “where”, or “when” because such questions usually require persons, things, places, or times (which can be expressed as noun phrases) as answers; questions beginning with “why” or “how” (which usually require procedural answers) are forbidden. Phrases are extracted from the question, and the question is reformulated as a structured boolean query with proximity constraints. The query returns a number of encyclopedia articles (possibly none). The number of hits (retrieved articles) can be increased by broadening the query, e.g., relaxing proximity constraints, dropping phrases, or reducing a phrase to a sub-phrase. The number of hits can be decreased by narrowing the query, e.g., adding a term such as the main verb. Retrieved articles are ranked by number of matches with the query. Phrases or words that match phrases or words in the query are then extracted from the retrieved articles. An answer to the original question may require information from multiple retrieved articles. A variety of term matching rules are used; the phrase that appears in an article need not have exactly the same form as the corresponding phrase in the question, e.g., “was succeeded by” can match “who succeeded.” This system demonstrates that “shallow syntactic analysis can be used to advantage in broad domains.”

Syntactic phrase extraction is performed and used somewhat differently by Riloff [SIGIR ‘95]. The goal is one of those specified for the Message Understanding Conference (MUC) series rather than the Text REtrieval Conference (TREC) series. In this case, the goal is extraction of data about a particular topic, e.g., terrorist events or joint ventures, from each relevant document, and the filling out of a topic-specific template for each such document. For example, given a document reporting a terrorist event, the goal might be to extract such key elements as the name of the perpetrator, the name of the victim, the location at which the event took place, etc. Since the objective is not merely to identify relevant documents but to “understand” each document well enough to extract key elements, Natural Language Processing (NLP) naturally assumes greater importance. So does knowledge of the domain to which the extraction task applies, e.g., terrorism. Hence, Riloff extracts instances of phrase patterns called “concept nodes.” A concept node is a specific linguistic pattern containing a specific word. For example, each occurrence of the concept node called “$murder-passive-victim$” is a string of the form “<X> was murdered” (or “<X> were murdered” for multiple victims).

An important result demonstrated by Riloff (also noted by others) is that such common IR techniques as stop-lists and stemming, which are intended only to remove noise words and noise vari-

ation of a given term, can in fact remove clues crucial for judging the relevance of a document or the semantics of its content. For example, the presence of the term “dead” in a document was not a reliable guarantee that the document described a murder, but the phrase “was found dead” proved to be an extremely reliable descriptor for that purpose. It evidently “has an implicit connotation of foul play.” Similarly, the presence of the term “assassination” (singular) in a document proved a much more reliable indicator that the document described a specific assassination event than the presence of the term “assassinations” (plural). The plural often referred to assassinations in general. Prepositions, passive vs. active verb form, positive vs. negative assertion, also proved significant to determining the significance of a phrase as a descriptor, at least within a specific domain. For example, the term “venture” by itself in a document was not a good indicator that a document described a joint venture. But the phrases “venture with” and “venture by” proved very good descriptors indeed, e.g., 95% precision for the test collection.

9.2 Sense Disambiguation of Terms and Noun Phrases

Another area (besides phrase extraction and phrase analysis) where researchers have tried to use syntactic/semantic techniques to improve IR performance is “word sense disambiguation.” [Chakravarthy & Haase, SIGIR ‘95] [Sanderson, SIGIR ‘95] [Voorhees, SIGIR ‘93] It is commonplace that natural language words have multiple meanings. More precisely, a string of characters representing a word (a “lexical token” in the jargon of the field) can represent multiple words, each with a different meaning (a different “sense”). (The technical term for such a word is *polysemou*s.) Sometimes, the meanings are closely related meanings of what we tend to consider the “same” word. Often, they are two (or more) completely different words that happen to be spelled the same, e.g., “bank” can mean either a financial institution or the border of a river, “bat” can mean either a flying mammal or a sporting implement, etc. It is plausible to suggest that if one can use syntactic/semantic methods to determine which “sense” of a word is intended by each occurrence of the word in any given document or query, one will be better able to retrieve the documents relevant to a given query (better recall) and to reject non-relevant documents (better precision). In particular, one would be able to avoid matching one sense of a word in a query with a completely different sense of the word in a document.

One crucial issue that arises in sense disambiguation is how fine-grained the sense distinctions are to be. This may depend on the resources available, e.g., an on-line Machine Readable Dictionary (MRD) such as the Longman Dictionary of Contemporary English (LDOCE) [Procter, 1978] may provide a relatively large number of senses relative to what may be generated by hand from a training corpus. On the other hand, the number of senses to distinguish depends heavily on the task for which the sense disambiguation is being performed. The number of senses that need to be distinguished for IR, may be considerably less than the number that need to be distinguished, e.g., for polished machine translation.

One reason is that senses of a single word that are closely related in one language may (and often will) correspond to different words in another language.

Moreover, for IR purposes, it may often be sufficient to distinguish fairly broad subject categories. For example, the 1978 LDOCE distinguishes 13 senses of the word *bank*. [Guthrie, 1993] Yet, these thirteen senses may be grouped into several broader categories. One coarse sense of *bank* is

a repository to which one can make deposits and from which one can make withdrawals. At a finer grain, one can distinguish a financial bank from a blood bank or a leave bank. Similarly, another quite different coarse sense of *bank* is a heap of material. At a finer grain, one can distinguish a snow bank, a sand bank, a cloud bank, and the bank of a river. For many IR purposes, it may be sufficient to distinguish the senses of bank at the coarser level, or to distinguish a financial bank from other repositories.

Guthrie et al. [1991] disambiguates words like *bank* by using word co-occurrence statistics derived from an electronic reference source, in this case the LDOCE MRD. The problem is to disambiguate polysemous words like *bank*, occurring in a test corpus. Given an occurrence in the test corpus of *bank*, they extract the local context of the given occurrence. The context may be the sentence in which the given word occurs, e.g., "We got a bank loan to buy a house," or some specified number of words on either side of the given word. This context is then matched against each *neighborhood* of the given word, derived from the LDOCE.

The neighborhood of a given word is the set of words (excluding stop words) that co-occur in the definition of one sense or group of related senses of the given word. The electronic version of LDOCE specifies a set of relatively broad subject categories, each category designated by a Subject Field Code (SFC). (The use of SFC's is discussed in greater detail below, in the section on Concept Identification.) SFC's are assigned to some of the sense definitions of a given word. For example, the sense of bank as "a place in which money is kept and paid out on demand" is assigned an SFC of "EC" which stands for the subject category "Economics." (The verb sense of *bank*, "to put or keep (money) in a bank" is also assigned the code "EC.") At the simplest level, the neighborhood of the Economics sense of *bank* is the set of words (excluding stop words) in all the definitions of *bank* that are assigned an SFC of "EC." Since the definitions are relatively short, a co-occurrence neighborhood is computed for each non-stop word in each 'EC" definition of *bank*. In other words, since the word "money" occurs in one or more EC definitions of *bank*, a neighborhood is computed for *money.* The union of these neighborhoods becomes *the* neighborhood for the EC sense of *bank*. Guthrie et al. then compute the overlap (number of words in common) of this neighborhood with the context of the occurrence of *bank* to be disambiguated. In the same way, the overlap is computed between the context and the neighborhood of every other sense (or subject category) of *bank*. The sense of the neighborhood with the greatest amount of overlap with the context is chosen to be the sense of *bank* in the occurrence being disambiguated. The Guthrie procedure is automatic, but the best success rate achieved in this and similar approaches is only about 70% accuracy.

From the sentence used in the above example, i.e., "We got a bank loan to buy a house," it is easy to see why sense disambiguation using MRD definitions might be relatively ineffective. A key word in the above sentence is "loan." Although a great many loans involve money, it is entirely possible that none of the definitions of money in the MRD involve the use of the word "loan." The word "loan" may co-occur with the word "money" with significant frequency in a news or business corpus, yet not occur in definitions of "money."

Voorhees [SIGIR '93] uses the large general purpose semantic network, WordNet [Miller, Jlex, 1990]. WordNet groups words into strict synonym sets called "synsets." The synsets are divided into four main categories: nouns, verbs, adjectives, and adverbs. Within each category, synsets are

linked together by a variety of semantic relations appropriate to the category, e.g., for nouns the relations include hypernymy/hyponymy (the "is-a" or class/subclass relation, e.g., a car *is a* vehicle), antonymy (word A means the opposite of word B), and several "part-of" relations. Voorhees, in the reported research, disambiguates nouns only. The approach is to group noun synsets in WordNet so that the synsets in a given group all contain words with closely related senses. The groups are called "hoods". A hood is effectively similar to a class in Roget's thesaurus. Given a document, she can then count the number of nouns in the document that are found in a given hood, i.e., the number of nouns that can have the sense (or related senses) of a given hood. Of course, a given noun can belong to multiple hoods, corresponding to multiple senses. In such cases, the noun is assumed to have the sense it possesses in the hood which contains the largest number of words (or word occurrences) from the document. In other words, it is assumed that in a given document, a given noun will tend to co-occur most with other nouns having related senses. (The word "bat" is ambiguous, but if it co-occurs in a document with the also ambiguous words, "base," "glove," and "hit," it is very likely that the word is being used in its baseball sense.)

Leacock et al. [Corpus Proc, 1996] have investigated sense disambiguation of words in a large text corpus by statistical classification based on term co-occurrence in the contexts in which the given words occur. In this study, a context was defined to be a sentence in which a given word occurred, and the preceding sentence. The preceding sentence was included in the context because a given word is often used anaphorically. If the preceding sentence also contained the given word in the same sense, then the sentence preceding *that* sentence was also included in the context. A single polysemous word, "line," was studied., A training set was constructed consisting of contexts for each of six different senses of "line." These included: line of text ("a line from Hamlet"), a formation ("a line at the box office"), an abstract division ("the narrow line between tact and lying"), a telephone connection ("the line went dead"), a thin, flexible cord ("A fishing line"), and a class of product ("a product line"). Elements varied included: type of classifier (Bayesian, vector space, and neural network), number of senses (two, three and six), and number of contexts in training set (50, 100, and 200). The machine classifiers were also compared to human classifiers. All of the machine classifiers performed best with 200 contexts (71% to 76% correct answers). Performance of the three classifiers tended to converge as the number of contexts went up. With only two senses to distinguish (between product line and formation), the accuracy was over 90%. However, with three senses (product line, line of text, formation), the classifiers did only a little better than with six (mean accuracy of 76%). In general, some senses were harder than others for all three classifiers to identify; the hardest were: line of text, formation, division, in that order. The three easiest for all three classifiers were product, phone, and cord, although the order of ease for these three varied with the classifier. Moreover, for any given sense, some contexts made sense identification easier than others. Interestingly, the humans, when given the same information as the machine classifiers, e.g., a vector of stemmed substantive words for the vector space classifier, found the same senses (and context within a sense) easy to distinguish, and the same senses difficult. In general, the humans did better than the machine classifiers. The exception was the "easy" contexts, chosen because the machine classifiers made no errors on them; the humans had a 15% error rate on these contexts. Most significant, when the humans were given the original sentences comprising each context, their performance was nearly perfect. On the other hand, when they were given the same input as the Bayesian classifier, i.e., the complete set of words in the context, with no stemming or stop-word removal, but with the words in reverse alphabetical order (because none of the classifiers used word order), their performance was much the same as when stop-

words (also called *functi*on words) were removed. The stop words were of no value to the human classifiers except when they were presented in the proper syntactic order. Plainly, the machine classifiers would do much better if they could make effective use of order and proximity as the human classifiers do.

Evaluation of sense disambiguation can be very tedious since it is necessary not only to evaluate the retrieved output for relevance, but to examine the retrieval process at the level of individual words to determine how it was affected by the disambiguation. Moreover, it is sometimes difficult to determine what the intended sense of a word, e.g., in a short query, actually *is*. Sanderson [SIGIR ‘94] reports on a novel method of evaluation, attributed to Yarowsky [Hum Lang, 1993]. The approach is to create ambiguous pseudo-words artificially by replacing each occurrence of a pair of distinct words in the document (any pair, not just adjacent words) by a pseudo-word formed by concatenating the two actual words. The effect is to create a new document with half as many word occurrences, each having a known-in-advance ambiguity. Still greater ambiguity can be created by generating pseudo-words composed of N real words where *N* may be, 3, 4, …, 10, etc. Of course, indexes and queries have to be modified correspondingly. “The disambiguator is then applied to each occurrence of [each] new word. Evaluation of the disambiguator’s output is a trivial matter as we know beforehand the correct sense of each occurrence of the word.”

The surprising result of such research into disambiguation is that it seems to improve IR performance very little. Indeed, in most cases, it actually degrades performance. Sanderson found that adding quite large amounts of controlled ambiguity, e.g., size ten pseudo-words, had little effect on IR retrieval. Removing a controlled amount of ambiguity seemed to make IR performance worse. Voorhees too found that in most cases, performance was degraded by disambiguation. The following reasons were identified: If the number of terms in the query is not very small, the term combination itself appears to have the effect of disambiguating documents and queries (e.g., as in the “base, bat, glove, hit” example above). This phenomenon is also noted by Lewis et al. [CACM ‘96] On the other hand, if the query is very small, then it provides very little context for the disambiguator to use. Hence, the disambiguator is likely to introduce errors, resulting in missed query document matches. According to Voorhees, “[t]hese results demonstrate that missing matches between the documents and query degrades performance more than eliminating spurious matches [by disambiguation] helps retrieval for small homogeneous collections.” Sanderson used a less homogeneous collection with similar results. He found that the disambiguator had to be at least 90% accurate to avoid performance degradation. Supplying a domain-specific context for short queries (as humans do) and selecting the most frequent sense for an ambiguous term may alleviate the problem. However, there are some queries for which Voorhees found that disambiguation was very helpful.

9.3 Concept Identification and Matching

In all of the previous discussion of methods for matching documents against queries or topics, the query and document terms being matched are words or phrases extracted from the given queries and documents, or (in the case of techniques like LSI) factors derived statistically from the words. The words may be subject to pre-processing, e.g., stemming to reduce variants to a common form, but in essence query words are being matched against document words. However, the words are essentially being used as surrogates for the concepts they express. Boolean expressions, vectors,

SVD factors and the like may be used to capture the intended concept more precisely, by specifying the words that co-occur in a given semantic context. However, what the user really wants is to retrieve documents that are *about* certain concepts. Of course, in many cases the user is looking for documents about a specific named entity, e.g., a given person, or a given book. Even then, certain concepts are usually implicit, e.g., if she is looking for documents about a person named "Baker," she wants to specify that the documents must be about a person named "Baker" and not about the profession, "baker." Moreover, she may want to see documents about an author named "Baker," or a CEO named "Baker," or a Baker who participated in a certain criminal action, etc.

Liddy et al., [ACM Trans IS, 1994] have developed a technique for matching topics (or topic profiles) against documents at the concept or subject level. They have implemented a text categorization module based on this technique; the module provides a front-end filtering function for the larger Document Retrieval through LINguistic Knowledge (DR-LINK) text retrieval system, although it can also serve as a stand-alone routing or categorization system and has been tested in this mode, e.g., using TREC 2 data. The text categorizer described in TREC-2 used a machine-readable dictionary (MRD), the *"Longman's Dictionary of Contemporary English"* (LDOCE). The second edition (1987) of LDOCE contains 35,899 "headwords", i.e., words for which the LDOCE contains an entry. Each headword may contain multiple definitions, corresponding to multiple senses of the word. The LDOCE contains 53,838 senses of the 35,899 headwords. For each word, the LDOCE specifies one or more "parts of speech." For each part of speech associated with a given word, the LDOCE specifies all its senses, and assigns a "Subject Field Code" (SFC) to each sense. For example, "earth" can be both a noun and a verb (the latter chiefly in British terminology). As a noun, it can refer to the planet on which we live (SFC = ASTRONOMY), or to the soil in which we plant crops (SFC = AGRICULTURE), or to a class of chemicals (SFC = SCIENCE), etc. In all, the LDOCE assigns six SFC's to "earth" as a noun. (Current versions of DR-LINK use a proprietary MRD, containing a proprietary, and larger, set of SFC's. [Liddy, PC] Since this MRD is specifically designed to serve the needs of DR-LINK, it omits many features of stand-alone, commercial MRD's such as the LDOCE.)

The categorizer uses the SFC's to construct SFC vectors, instead of term vectors. First, it assigns one or more parts of speech to each word in the document or topic statement, using a probabilistic part-of-speech tagger, POST. (As in most vector space approaches, the topic or query is treated as just another document; a vector is developed for the topic in exactly the same way as for the documents to be categorized.) Then, it looks up each part-of-speech tagged word in the LDOCE, and attaches to the word all SFC's associated with it. For example, if a particular occurrence of "earth" in a given document is tagged as a noun, it would have six SFC's attached to it, including ASTRONOMY, AGRICULTURE, SCIENCE, etc. If the word is not found in the LDOCE, stemming is applied, and a second LDOCE lookup is performed. Note that some words may not be assigned any SFC at all, either because the word is not in the LDOCE, or because it does not appear in the LDOCE as the part-of-speech with which it has been tagged. Some words may require a larger, or a more specialized dictionary resource than the LDOCE.

Since a given word, even when tagged with a given part of speech, may still have multiple SFC's assigned to it (as in the example of "earth" above"), the word sense must be disambiguated. However, in contrast to the word sense disambiguation discussed in the preceding section, here disambiguation is conducted entirely at the SFC, i.e., concept, level. Disambiguation is first performed

at the level of the local context, here interpreted as the sentence level. SFC frequencies are computed for the given sentence. There are two reasons why a given SFC may be assigned more than once in a given sentence. First, two or more words in the given sentence may each be assigned the same SFC as one of their respective senses. Second, two or more senses of the same word occurrence, even tagged with the same part of speech, may be assigned the same SFC. Liddy et al. give the example of the sentence, "State companies employ about one billion people." The word "state" is assigned the SFC "POLITICAL SCIENCE" four times, corresponding to four different senses of the word, e.g., state" as nation, "state" as subdivision of a nation, "state" as in "separation of church and state," state" as in "secretary of state," etc. The word "people" is assigned the SFC "POLITICAL SCIENCE" twice, e.g., "people" as in "the people of New York," and "people" as in "the people have chosen a president," etc. Two important cases of SFC frequency are identified for purposes of sense disambiguation: If a given word within the given sentence is only assigned a single SFC, this is called a "unique SFC." Obviously, for such a word, no sense disambiguation is required. On the other hand, an SFC is considered "highly frequent" if it is assigned more than three times over all the words in the given sentence. If a word is assigned multiple SFC's and one of these SFC's is highly frequent, that SFC will be assigned to the given word. In the illustrative sentence above, "Political Science" is a highly frequent SFC since it is assigned six times (four for "state" and two for "people"). Hence, it will be assigned as *the* SFC for both "state" and "people" in that sentence. Note that a word like "people" has other senses, e.g., the LDOCE also assigns the SFC's "SOCIOLOGY" and "ANTHROPOLOGY" to the word "people." But *in the given sentence*, 'POLITICAL SCIENCE" is a much more frequent SFC than either "SOCIOLOGY" or "ANTHROPOLOGY," so it is chosen *in that sentence* as the preferred SFC to disambiguate "state" and "people."

If a word in the given sentence remains ambiguous, i.e., the word is assigned multiple SFC's but none of the SFC's assigned to the given word is highly frequent, then an SFC correlation matrix is used. This matrix is built from a training corpus; in the research described here, the corpus consisted of 977 newspaper articles. The matrix is 124 X 124 because there were 124 SFC's in the LDOCE edition used in the research described here. The given word is disambiguated by choosing that one of its assigned SFC's that has the highest correlation (tendency to co-occur in the same document) with the unique and highly frequent SFC's in the same sentence. Note that the correlation matrix measures correlations among SFC's at the document level. Hence, at this stage, document as well as sentence level knowledge is being used for disambiguation.

After every ambiguous word in every sentence in the given document has been disambiguated, a document-level SFC vector is created by summing the single SFC's assigned to each word. For example, if the SFC "POLITICAL SCIENCE" is assigned as the SFC of one or more words in one or more sentences of the given document, then it will become one of the terms of the SFC vector for the given document, with a term weight determined by its normalized term frequency, or some other appropriate term weighting formula. For example, the reported research uses Sager's term weighting formula, f_{in}/k_n, where f_{in} is the SFC frequency of SFC i in document n and k_n is the number of tokens (SFC occurrences) in document n.

Once an SFC vector has been created for every document in the collection being categorized, and every topic for which documents are to be categorized, the similarity of each document to each topic can be computed, and all documents above a specified threshold for a given topic can be

assigned to its category. However, in DR-LINK, the SFC vectors are employed in a more sophisticated manner, to take advantage of the discourse structure of the document. This is described in the next section. Moreover, similarity at the SFC vector level, though it can be used in a stand-alone mode, can also (and in DR-LINK is) combined with similarity at other levels, e.g., matching at the level of proper nouns, relationships, etc. These issues are discussed in sections that follow.

The Liddy topic matching scheme described above is a novel form of "controlled vocabulary" methodology. It is customary to distinguish IR systems as "controlled vocabulary" or "free text vocabulary." In the former, documents are indexed manually by their authors or by professional indexers from a preset vocabulary of index terms. In the latter (as in most of the research described in this report), the index terms for a given document are generated automatically from the content of the document. The great advantage of the former is that expert human judgment is applied to choose appropriate document descriptors. The great disadvantages are that it is extremely labor-intensive, and that users must know and use the controlled vocabulary in formulating their queries and topic profiles. The Liddy scheme offers "the best of both worlds." On the one hand, documents are indexed automatically, and the user can formulate queries and topics in terms of her own vocabulary. On the other hand, document and topic terms are mapped from a large dictionary lexicon into a controlled vocabulary of concept terms, the SFC's. This controlled vocabulary has been developed and refined over a considerable period of time by professional lexicographers.The only drawback is that machine-readable resources such as the LDOCE need to be extended (along with the set of SFC's) to deal with documents in specialized domains.

Note that *relevance feedback* (see below) can readily be applied to these SFC vectors [Liddy et al, Online, 1995]. The user's original query is converted into an SFC vector and matched against SFC vectors representing each document in the given collection, as described above. A list of documents, ranked by SFC vector similarity to the query and above a specified similarity threshold, is returned to the user. The user then identifies those documents on the list (or paragraphs or sentences within those documents) that she considers relevant to the given query. DR-LINK can generate a Relevance Feedback (RF) SFC vector from each of the user-selected documents, and aggregate these vectors into one RF-SFC vector. Since the documents (or paragraphs or sentences) selected by the user should be good examples of the kind of document for which she is looking, the resulting RF-SFC vector should exemplify (better than the original query) the user's requirements at the conceptual (subject-category) level. Hence, the RF-SFC vector can be used in place of the SFC vector generated from the original query, to calculate similarity at the conceptual level, and return a new ranked list of documents. As with other forms of relevance feedback, this process can be repeated to refine the query further, until no more improvement is observed. Of course as with all relevance feedback, improvement depends upon the existence within the collection of documents that are truly relevant to the user's requirements.

The advantages of an NLP-based, subject-based system such as DR-LINK, over the far more common keyword-based systems is well-illustrated by the comparison of three IR systems reported by Feldman. [ONLINE, 96] For example, she cites a common experience of IR searchers, expressed by the rule, "Search for wars, and you will also retrieve sports." It is not hard to see why keyword-based systems, whether boolean or vector space, are vulnerable to such errors. The vocabularies of sports and war overlap to a significant degree, e.g., terms like "battle," "conflict," "win," "loss," "victory," "defeat," and many more. If the user employs some of these terms in for-

mulating her query, a keyword based system will have difficulty distinguishing sports stories from war stories. Feldman encountered this difficulty when she asked for “information about African countries which had civil wars, insurrections, coups, or rebellions.” She encountered this difficulty with both the boolean DIALOG system, and the relevance ranking TARGET system. She encountered this difficulty even though the terms she used (or stemmed), terms like “insurrections,” “coups,” “rebellions, “wars,” are not the most common terms to appear in sports stories. Presumably, some of these terms, or synonyms for them, e.g., synonyms for “war,” do appear in sports stories. The only way to exclude such stories from a boolean query without excluding legitimate stories too, would be to add “AND NOT....,” where the “NOT” is followed by a set of sports-specific terms, e.g., “NOT rugby,” “NOT soccer,” etc. A relevance ranking system might be expected to do somewhat better, if the specified terms occur more frequently in war stories than in sports stories. However, TARGET did *no*t do better on this query than DIALOG in Feldman’s test. DR-LINK did much better on this query, retrieving 50 relevant war stories, and excluding the “false drop” sports stories altogether.

It is easy to see why DR-LINK was much more successful on this query. The SFC vector approach provides several advantages in such an example. For instance, a sports story would almost always contain some sports-specific terms. Hence, the corresponding SFC vector for such a sports story would contain sports-specific SFC’s. The presence of such SFC’s would help DR-LINK to disambiguate terms that belong to both the sports and war domains correctly, choosing the sports sense of such terms. On the other hand, the fact that all the terms in the user’s query had a war sense and none had a pure sports sense would favor disambiguating the query terms as having their war senses at the SFC level.

It should be noted that DR-LINK is not the only system that indexes a large document collection automatically using a controlled vocabulary. For example, CONSTRUE-TIS [Hayes et al., 1990] assigns subject category terms automatically from a controlled vocabulary to a large collection of news stories maintained by Reuters. The subject categories include 135 economic categories, e.g., mergers and acquisitions, corporate earnings, interest rates, various currencies, etc. 539 proper name categories (people, countries, companies, etc.) are also supported. Rapid automatic indexing enables the system to keep up with the rapid addition of new stories. Indexing is based on shallow knowledge base techniques. Each subject category is recognized by if-then rules. The “if” conditions are specified in terms of “concepts.” The concepts, in turn, are defined in terms of a pattern language. The pattern language specifies keyword patterns. The patterns may include such features as keyword order, boolean combinations of keywords, gaps for a specified number of arbitrary words, etc. Hence, CONSTRUE-TIS conceptual indexing by subject category employs much more than simple boolean conditions, but much less than true natural language processing. However, the subject category terms that are assigned automatically to each story (the terms that comprise the controlled vocabulary) are the same terms that users must employ to retrieve the story. By contrast, the user of DR-LINK never sees or needs to know its controlled vocabulary. Documents and natural language topic requests are both mapped automatically into the common, controlled vocabulary of SFC’s.

9.3.1 Formal Concepts

The preceding section dealt with concepts assigned to word senses by human beings, e.g., domain experts or lexicographers. In this section, we describe how "formal concepts" can be defined for a given domain, or a given body of text, using a mathematical method for knowledge representation, exploration, and processing called Formal Context Analysis (FCA).

FCA is an unsupervised learning technique that "allows implications between attributes to be determined and visualized." [Cole et al., Comp Int, 1999]

FCA models a domain as being composed of individual objects (called *formal objects*) and their attributes (called *formal attributes*). The set of objects and attributes chosen to model a domain is called a *formal context*. Formal contexts relate objects to their attributes. A context may be represented as a matrix in which the rows are objects, the columns are attributes, and each cell specifies whether a given object has (yes) or doesn't have (no) a given attribute. If a formal context represents a textual document (or collection of documents), the objects may be the individuals mentioned in the text, e.g., specific persons, companies, locations, buildings, etc., whether named or unnamed. Correspondingly, the attributes of a given individual object will be those mentioned explicitly or implicitly in the text as characterizing the objects, e.g., a building may be small, tall, or ornate, etc. The object type is also considered an attribute of the object, e.g., the attributes of the Empire State Building include both the descriptive adjective "tall," and the object type "building."

Semantic relations can also be formalized by a context matrix. Each row of a relation matrix is an ordered pair of objects (classes or instances) that the text specifies (explicitly or implicitly) as in some relation to each other. The columns are the relations, and the attributes of the objects participating in those relations. Note that a pair of objects may be in more than one relation to each other. If row *i* represents the ordered object pair {*O1*, *O2*}, and column *j* represents the relation *R*, then a "yes" in cell {*i, j*} means that *O1* is in the directed relation *R* to *O2*.

The formal contexts of FCA can be translated into Sowa's conceptual graphs (CG's) [1984] and vice versa.[Wille, 1997] Both CG's and FCA have been used to represent and process knowledge. Both are methods of formalizing logic, and relating logical concepts to real-world concepts. Since CG's were developed to represent the syntax and semantics of natural language, FCA can be used for the same purpose. The objects of FCA map into Sowa's concept nodes. The relations map into the relations that connect Sowa's concepts. These relations are generic rather than application-domain-specific, e.g., Sowa defines such relations as "agent" (AGNT), "patient" (PTNT), "location" (LOC), etc. Agents act upon objects, patients (not necessarily the medical kind!) are objects that are acted upon, and so on. Hence, in the above example, if John sued Mary, *O1* could be John, *O2* could be Mary, *O1* would be the AGNT with respect to suing, the one who is *doing* the suing, and Mary would be the PTNT with respect to the suing, the one who is *being* sued. The context would show "yes" for the cell {John, AGNT}; it would also show "yes" for the cell {Mary, PTNT}. The context would also contain cells for attributes of John and Mary individually, e.g., John may be tall and antagonistic, Mary may be blonde, reasonable, etc.

An FCA *formal concept* is a pair (*A,B*) where *A* is a subset of the objects in a context, and *B* is a corresponding subset of attributes applicable to all the objects in the *concept*. Eklund et al. [proposal, 1999] offer a very simple example of a context that describes the solar system. The objects,

A, are the planets, the attributes, *B*, are size (small, medium, large), distance (near, far), and moon (yes, no). One of the concepts that FCA can discover is {{Earth, Mars}, {small, near, yes}}. A defining characteristic of the formal concept (*A*, *B*) is that "*A* must be the largest set of objects for which each object in the set possesses all the attributes of *B*. The reverse must be true also of *B*." [Cole et al., 1999] In other words, the set of attributes, *B*, must be the largest set of attribute values that characterize the set of objects, *A*.

The relationships among concepts and individuals that are captured in a formal context or a conceptual graph, can also be represented as a "concept lattice." Each node of a concept lattice corresponds to the maximum set of objects that possesses a given attribute. This set of objects is called the extent; the number of objects comprising this extent may be attached to the node. (This set may also be viewed as the maximal concept associated with the given attribute.) If a parent node representing objects with attribute m_1 branches to two child nodes representing objects with attributes m_2 and m_3 respectively, then the extent of the m_2 node represents objects having both attribute m_1 and attribute m_2. Similarly, the extent of the m_3 node represents objects having both attribute m_1 and attribute m_3. Now, since a lattice is a more general graph than a tree, it is quite permissible for two nodes, e.g., the m_2 and m_3 nodes of the above example, to meet in a common node below. This common meet node represents the set of objects that possess attributes m_1, m_2 *and* (not *or*!) m_3. If a node having attribute m_4 is immediately below this meet node, various implications involving m_2, m_3 and m_4 (given m_1) can be represented. If the extent of m_4 is greater than or equal to the extent of the meet of m_2 and m_3, this represents "m_2 and m_3 implies m_4" (in other words, every object having both attribute m_2 and attribute m_3 is one of the objects having m_4). Similarly, If the extent of m_4 is less than or equal to the extent of the meet of m_2 and m_3, this represents "m_4 implies m_2 and m_3" (in other words, every object having attribute m_4 is one of the objects having both attribute m_2 and attribute m_3). If the extent of m_4 is exactly equal to the extent of the meet of m_2 and m_3, then there is equivalence between m_4 and the meet (intersection) of m_2 and m_3.
Partial implications (conditional probabilities) can also be represented. The probability of m_1 given m_0, denoted $\Pr(m_1 \mid m_0)$, is extent(m_1)/extent(m_0). If the m_0 node is the root of the lattice, representing the entire population of objects under consideration, then the implications described in the preceding paragraph are unconditional. However, one can still compute $\Pr(m_1 \mid m_2)$ as extent(meet(m_1 and m_2))/extent (m_2). That is, the probability of an object having attribute m_1 given that it has attribute m_2, is computed as the quotient of the number of objects having both m_1 and m_2 (objects having m_1 given that they also have m_2) divided by the total number of objects having m_2.

Given that a rich context may involve a large number of objects and attributes, the user will normally want to focus on a more manageable subset. Cole et al. [Comp Int 1999] provide this capability. That is, the user can focus on certain specific objects and attributes of interest; the system will generate the concept lattice for the selected entities. Cole calls this a "scale." As a further refinement, their system allows the user to generate a lattice involving one set of concepts within a lattice involving another set of concepts, i.e., to nest one scale within the nodes of the lattice of another scale.

For example, Cole et al. have applied FCA to the analysis of medical discharge summaries. In both cases, the objectives are to represent the relationships and implications among the concepts.

"The relationships will often be mundane, but occasionally surprising and new." For example, they generate a concept lattice exhibiting relationships among classes and subclasses of disease, e.g., "respiratory tract diseases," is a broad disease class, "asthma" is a disease subclass within that broader class. Another broad class may contain such subclasses as "carcinoma." The lattice may indicate (via "meet" nodes) the co-occurrence of diseases, e.g., how many patients suffer from both asthma and carcinoma. Within each node of this "disease" lattice, another lattice may be nested, exhibiting, e.g., patient behaviors such as smoking, and drug therapies that have been applied. For example, the "smoking" node of the behavior lattice can be found in each node, e.g., the "carcinoma" node, of the disease lattice. The extent of the "smoking" node within the "carcinoma" node gives the number of patients diagnosed with carcinoma who smoke. By studying these lattices, relationships among diseases, patient behaviors, and therapies may be displayed and analyzed.

Recently, Cole et al. [KDD99] have focused on semi-structured text (XML-based and HTML-based). The FCA objects studied are text documents, e.g., email messages. The attributes are keywords and patterns found in the documents. The patterns are specified as regular expressions. For example, email attributes may include the originator and addressees (from and to lines), and a date condition, e.g., all dates in the range from September to November 1994. The attributes may also include keywords, e.g., names, mentioned in the textual body of the email document. Analysis of the conceptual lattice associated with an email collection leads to the discovery of patterns, e.g., that a high proportion of the emails addressed to a given addressee mention a given person or combination of persons by name.

9.3.2 Concepts and Discourse Structure

Many researchers have recognized that a document, especially a large document, may not be the ideal unit for matching against queries or topics. A document may deal with multiple topics. The matters of concern to a given user, or the key words that identify her interests, may be localized to a small portion of a document. Hence, a variety of research efforts, some of them described in this report, attempt to break documents into segments, often called "passages." Sometimes, the boundaries of these segments are determined orthographically, e.g., on the basis of paragraph or section or sentence boundaries. In other cases, documents are segmented arbitrarily, e.g., by overlapping windows *N* characters long. The former approach takes semantics into account, but only indirectly, by assuming that sentence, paragraph, or section boundaries specified by the author accurately reflect her intended semantic structure. The latter approach ignores semantics in favor of locality. Of course, it is likely that the words or sentences that occupy a local passage have some semantic relationship, but it is impossible to say a priori what that relationship will be.

Liddy et al. [Proc RIAO Conf., 1994] have taken a more principled approach by studying the discourse structure (based on "discourse linguistic theory") of various types of documents, e.g., newspaper articles [TREC-2, 1994], or abstracts of empirical technical documents [Liddy, ASIS '87] Liddy, 1988]. A coherent, well-written document has a semantic structure that represents the way the author has organized the ideas or story she wants to tell. Moreover, textual documents of a particular type will have a predictable, standard structure. The elements of this structure are called "discourse components." Liddy has extended an earlier model due to van Dijk [Hillsdale, 1988] for the text type "newspaper article." She has identified 38 discourse components in her

extended model. Each clause or sentence in a given article can be tagged as one of these components. These tags "instantiate" the model. Assigning a tag to a clause says that the given clause belongs in the corresponding component of the model. Each component will contain certain kinds of information relative to the story told by the entire article. Examples of component tags for the newspaper article model are: MAIN EVENT, VERBAL REACTION, EVALUATION, FUTURE CONSEQUENCE, and PREVIOUS EVENT. Components may be nested, corresponding to nesting in the sentence structure. Linguistic clues are used to identify the components. For example, Liddy [TREC-2] offers the following example of nested, tagged discourse components.

> <LEAD-FUT> South Korea's trade surplus, <LEAD-HIST> which more than doubled in 1987 to $6.55 billion, <LEAD-HIST> is expected to narrow this year to above $4 billion, </LEAD-HIST> is expected to narrow this year to above $4 billion. </LEAD-FUT>

Plainly, this is a LEAD-FUTURE component about the expected future trade surplus of South Korea, as indicated by linguistic clues such as the phrase "is expected to," containing a nested LEAD-HISTORY component about South Korea's past trade surplus, as indicated by linguistic clues such as the past tense "doubled" and the "1987" date.

Tagging the clauses and sentences of a document by discourse component allows Liddy to generate multiple SFC vectors, one for each component. This means that one can not only match the subjects found in a topic against the subjects found in a document; one can also determine whether they are in the correct discourse component. For example, if the topic required that a document discuss future trade surpluses in South Korea, it would be important not only that the subject appear in a given document, but that it appear in a FUTURE EVENT or LEAD-FUTURE discourse component. A document that has the right subject in the right discourse component should receive a higher relevance ranking score than a document that has the right subject in the wrong component. Liddy has identified 38 discourse components for the newspaper article text type. However, she has found that topic requests usually do not have so fine a discourse grain. Hence, she has improved the performance of DR-LINK by mapping the 38 components into seven meta-components for the purpose of topic-document matching and ranking: LEAD-MAIN, HISTORY, FUTURE, CONSEQUENCE, EVALUATION, ONGOING, and OTHERS. These seven meta-components yield eight SFC vectors, one for each component and one for the combination of all seven together. The resulting module that matches topics against documents using these eight SFC vectors is called the "V-8 SFC Matcher."

Mann et al. [Text, 1988] have developed an alternative method of discourse analysis called *Rhetorical Structure Theory* (RST) [Mann et al., Text]. RST can be used for the automated markup and parsing of natural language texts [Marcu, AAAI, 96] [Marcu, PC, 1999]. Both Marcu and Eklund et al. [proposal, 1999] are exploring possible applications of RST to automated textual information extraction. Marcu is studying the application of RST to document summarization and machine translation of natural languages. Eklund et al. have considered its application to text data mining, knowledge base construction, and knowledge fusion across documents.

Mann et al. claim that RST "provides a general way to describe the relations [called *rhetorical relations*] among clauses in a text, whether or not they are grammatically or lexically signalled."

Of course, automatic parsing as developed by Marcu, depends on recognizing just such grammatical or lexical signals, and using them to drive the actions of the parser. Marcu has taken two approaches to automated RST parsing. First, he has developed a set of manual rules. Second, he has applied a machine learning tool to a large text corpus to "learn" a set of parsing rules. Clearly, the success of such automated parsing depends on the text possessing a certain coherence and clarity typical of news article text, and well-written scientific and legal papers. The techniques might be much less successful if applied to informal text, e.g., e-mail; they have never been applied to such informal texts. [Marcu, PC, 1999]

A rhetorical relation links two clauses (non-overlapping spans), called the *nucleus* and the *satellite*. The significance of these terms is that most (though not all) of the relations are asymmetric. One clause, the nucleus, is usually more essential than the other. The less essential clause, the satellite, is sometimes incomprehensible without the nucleus to which it is related. Even where the satellite is comprehensible by itself, the nucleus is generally more essential for the writer's purposes. Marcu capitalizes on this fact to generate automatic document summaries. The nuclear clauses, taken by themselves, form a coherent summary of the document's essence. The satellites do not. Another indication that the satellites are less essential is that they are often substitutable, i.e., in a given relation one satellite can often be substituted for another while retaining the same nucleus. Mann and Thompson define 24 rhetorical relations, but stress that the set is open-ended. Marcu has discovered a considerably larger set of relations, but anticipates that other researchers may discover yet more relations, as they explore other classes of text. [Marcu, PC, 1999]

Marcu is also considering the possible application of RST parsing to machine translation (MT). One of the difficulties with existing MT is that even when words and phrases and even clauses are properly translated, the overall translation of the text may be awkward or incorrect. This is because the discourse structure in the RST sense may vary from one language to another, especially at the lower levels of the RST trees. Translating this structure may substantially improve the quality of the translation.

The parsing of a text according to RST identifies and marks up all the rhetorical relations, and the clauses participating in those relations. The rhetorical structure is defined recursively, i.e., one of the spans participating in a rhetorical relation can itself be composed of rhetorical relations. Hence, the rhetorical structure of a text is a tree. Only the nodes of the tree are necessarily "simple" clauses. A given text can be parsed in multiple ways. Hence, a given text may be represented by multiple rhetorical parse trees. However, Marcu has defined principled rules for "legal" parse trees. If one adheres to Marcu's rules, one can still generate multiple trees for a given text document, but it becomes possible to reject some candidate parses as ill-formed while accepting others as well-formed.

The following passage illustrates two asymmetric rhetorical relations: "concession" and "elaboration."

> Although discourse markers are ambiguous,[1]
> one can use them to build discourse trees for unrestricted texts;[2]
> this may lead to many new applications in text data mining.[3]

A *concession* relation exists between the nucleus, either 2 or 3, and the satellite, 1. The nucleus is asserted to be true despite the contradictory "concession" of the satellite. An *elaboration* relation exists between the satellite 3 and the nucleus, either 1 or 2. The satellite elaborates on the assertion made by the nucleus. Note that the same clause, e.g., 3, can be a nucleus in one instance of one relation, and a satellite in one instance of another relation.

The word "although" is a lexical marker for the *concession* relation in the example given above. Similarly, the semicolon is a marker for the *elaboration* relation in this example. However, Mann et al. stress that "the definitions [of the rhetorical relations] do not depend on morphological or syntactic signals … We have found no reliable, unambiguous signals for any of the relations." Marcu's relative success indicates that for well-structured text, reliable markers can be found for a fairly high proportion of instances of the relations. But Marcu's studies have been limited to well-structured text types, e.g., Scientific American articles. Mann et al., on the other hand, have studied a wide variety of types including "administrative memos, magazine articles, advertisements, personal letters, political essays, scientific abstracts, and more." They claim that an RST analysis is possible for all of these diverse types. However, not surprisingly, they have found no simple linguistic markers that work for recognizing or delimiting these relations across all of the diverse text types. They also find that certain text types do *not* have RST analyses, including "laws, contracts, reports 'for the record' and various kinds of language-as-art, including some poetry."A common characteristic of the text types studied with some success by Marcu is that they are types of *expository* writing. Hence, Marcu's approach might work well with legal (judicial) opinions which are typically expository, although according to Mann et al. as quoted above, it would probably not work for laws and contracts, which are typically *not* expository.

Marcu measured the success of his automated parsers in terms of the classical precision and recall measures. Recall measured the proportion of the rhetorical relations in a text that were identified, i.e., the "coverage." Precision measured the proportion of identified relations that were identified correctly. These values were computed by comparing the results of the automated parses against parses performed manually by human judges. It was found that human judges could achieve a high degree of agreement in their respective parses, thereby justifying the claims of RST for the text types studied. However, it was also found that the human judges required a considerable amount of training and practice before they could achieve this consistency. Parsing according to the rules of RST is far from trivial.

Eklund et al. have proposed combining RST with FCA by generating formal contexts whose objects are larger entities like clauses, sentences, and even documents. Correspondingly, the relations would be rhetorical relations. If a simple formal context based on RST is converted to a simple conceptual graph (CG), its nucleus (in the RST sense) would become the "head" of the CG. Other text spans having RST relations to the nucleus could then be used to build more complete CG's. In this way, a CG knowledge base could be constructed, extending over whole documents and fusing the knowledge of multiple documents.

9.3.3 Proper Nouns, Complex Nominals and Discourse Structure

It is not sufficient to match topics against documents at the level of subject categories, i.e., at the level of SFC's. Much of the essential content of a topic request and the corresponding content of

documents, is found in proper names (PN's), e.g., names of persons, countries, companies, government agencies, etc. Much of the remaining content is found in "complex nominals" (CN'S), e.g., noun phrases formed by adjacent nouns, adjective-noun combinations, etc. In the passage about South Korea quoted near the beginning of the preceding section, "South Korea" is an example of an essential proper name (PN); "trade surplus" is an example of an essential complex nominal (CN) noun phrase. Systems that recognize and extract CN's and PN's for use as document and topic descriptors include DR-LINK [Liddy et al.,TREC-2], and Strzalkowsk's system [TREC-3] developed at NYU. The latter system augments a traditional statistical backbone "with various natural language processing components."

Recognition and extraction of complex nominals (CN's) involves several problems. [Strzalkowski et al., TREC-3] First, it is necessary to recognize CN's in any of various syntactic structure, e.g., to recognize "information retrieval" in "information retrieval system," "retrieval of information from databases," and "information that can be retrieved by..." It is necessary to distinguish cases where two-word CN's are satisfactory, from cases where longer CN's are necessary, e.g., in the phrase "former Soviet president," "former president" and "Soviet president" have quite different meanings so the longer three-word phrase should be preserved. It is also important to resolve the ambiguity in parsing CN's, e.g., to recognize that in the phrase "insider trading case," the key head word is "trading" and its modifier "insider." Phrases like "insider case" and "trading case" would be much less significant. Here, statistics must supplement pure syntactic analysis to ensure that the extracted phrase is semantically significant as well as syntactically correct. A statistical analysis of a corpus of business and economics stories will reveal that "insider trading" occurs far more often than the other pairs.

DR-LINK extracts CN's with its "Complex Nominal Phraser." Complex nominals are recognized "as adjacent noun pairs or non-predicating adjective + noun pairs in the output of the part-of-speech tagger." DR-LINK's PN recognition and categorization capability is described below in a subsequent section.

The problems of recognizing, extracting, and categorizing proper nouns, and the approaches to these problems taken by several research teams, is discussed in a later section.

DR-LINK not only extracts proper nouns (PN's) and complex nominals (CN's) from the topic statement, but also identifies the discourse component in which each should preferably be found. A discourse component in a given document is weighted on the basis of how many complex nouns and complex nominals from the topic statement it contains. This weight is then multiplied by another weight factor that reduces the total weight if the discourse component is not the one required by the topic statement, e.g., if the topic statement requires a noun phrase to be in a FUTURE component, but the phrase only occurs in the LEAD-MAIN component of a given document, the weight of that component will be reduced. If the FUTURE component also contains the given noun phrase, the FUTURE component will receive a higher weight than the LEAD-MAIN component. A CONSEQUENCE component that doesn't contain the desired noun phrase at all will receive a lower weight than the LEAD-MAIN component. Hence, the PN/CN similarity score depends not only on the presence of specified proper nouns and complex nominals in a given document (conventional keyword matching), but also on the discourse components in which they occur (discourse text structure matching). In fact, it is properly called a PN/CN/TS score

where the third acronym stands for Text Structure. Hence, both the SFC similarity score and the PN/CN/TS similarity score for a given document reflect the discourse structure of the document, and the discourse requirements of the topic statement.

Observe too that proper nouns are assigned semantic categories in DR-LINK [Paik, ARPA Workshop]; this categorization is discussed below. Hence, the PN similarity score of a document relative to a given topic may depend on a category match as well as a match on the actual proper noun itself.

9.3.4 Integrated SFC/PN/CN Matching

Plainly, a document that matches a topic statement not only at the level of subject categories (SFC's) and the discourse components in which they occur, but also matches on proper nouns and noun phrases, should be ranked higher relative to the given topic statement than a document that matches only on the one or the other. DR-LINK's "integrated matcher" combines these two kinds of matching. It takes as input two similarity scores: one based on SFC vector similarity, and one based on Proper Noun (PN), Complex Nominal (CN) similarity. An SFC cutoff or threshold score is computed. Documents are then ranked by a combined similarity score, with documents having a non-zero PN/CN similarity being ranked above those with a zero PN/CN score. Generally, documents below the SFC threshold are not returned, but a fraction of documents (depending on the recall level) with high PN/CN scores and documents below the SFC threshold are inserted above the zero PN/CN documents.

9.3.5 Relations and Conceptual Graph Matching

If a topic and a document match on the presence of two entities, there is a chance that the document is about the specified topic. If the two entities occur in close proximity in the document, the chance is better. If the topic requires that the entities occur in a certain discourse component, and the entities occur in the required component within the document, the chance of document relevance to the given topic is better still. But the assurance of relevance will be even stronger if the entities are related in the same way in both topic and document. Hence, DR-LINK also has the capability to identify relationships in a number of syntactic cases, e.g., noun phrases (NP's), nominalized verbs (NV's), prepositional phrases (PP's), and the complex nominals (CN's) already mentioned. In all cases where DR-LINK can identify a relation, it generates a concept-relation-concept (CRC) triple. This process not only identifies relations. It also converts varied syntactic forms into a single canonical form to simplify topic-document matching.

Once CRC triples have been generated, they are linked together to form larger units, e.g., clauses. Linking can occur because the same proper noun, e.g., a company name, occurs in two CRC's. Or it can occur because of coreference resolution, e.g., a pronoun in one CRC is identified as referring to a given proper noun in another. The structures that DR-LINK forms by linking CRC's are *conceptual graphs* (CG's). CG's [Sowa, 1984] are a graphical notation for representing the syntax and semantics of natural language. CG's are also available in a linear textual form. The graphical nodes represent entities, and relationships among the entities. However, CG's are more powerful than simple entity-relationship diagrams. They can represent first-order predicate logic, e.g., they

can express quantification. CG's have been discussed earlier with respect to Wille's work on their equivalence to formal contexts, as defined in Formal Context Analysis

Liddy offers an example of converting an NV into canonical CRC's: The phrase "the company's investigation of the incident" is converted into:

[investigate] -> (AGENT) -> [company]
[investigate] -> (PATIENT) -> [incident]

Note that the "nominalized" form "investigation" has been converted into the standard verb form "investigate." This simplifies matching with another phrase in which the verb form occurs. The relations "AGENT" and "PATIENT" are standard CG relations. The AGENT is the entity (also called the "actor') who performs the action, in this case investigates. The PATIENT is the passive entity that is the object of the action, in this case the entity that is being investigated.

Once CG's have been generated, they are made more "conceptual," by replacing entity nodes by codes representing the concepts of which the entities are instances. In TREC-2, DR-LINK used the Roget International Thesaurus (RIT) codes. (Current versions of DR-LINK, use WordNet Synsets. [Liddy, PC]) Finally, CG's in topics can be matched against CG's in documents. [Myaeng et al., JITAE, 1992]

9.3.6 Recognition of Semantic Similarity in CN's

Strzalkowski [IP&M, 1994] [TREC-2] [TREC-3] uses statistics not only to choose semantically significant head-modifier pairs from ambiguous CN's. He also use it to identify clusters of words that are semantically related, and hence are candidates for use in query expansion. Two terms are candidates if they occur in the same contexts, e.g., as head nouns with a number of common modifiers, or as modifiers of a number of common head nouns. For example, "man" and "boy" are modified by a number of the same modifiers (appear in a number of the same "contexts") in the corpus studied, including "bad," "young," and "older." That is, the corpus contains a number of references to "young men" and "young boys," "older men" and "older boys," etc. The same two words, "man" and "boy," also serve as modifiers of a number of the same head nouns, including "age" (references to "man's age" and "boy's age), "mother" (references to "a man's mother" and "a boy's mother"), and so on. Hence, "man" and "boy" appear in the same contexts both as head nouns and as modifiers.

Several additional factors must be considered to identify two terms as semantically similar. First, the terms should appear in few contexts other than the ones they share. Second, the shared contexts must not be too common, e.g., "natural" is too common a term to justify predicting similarity of "logarithm" and "language" on the basis of the shared contexts, "natural language" and "natural logarithm." Third, the number of *distinct* shared contexts must exceed some threshold; this threshold depends on how narrow or broad the training corpus is, e.g., a Wall Street Journal corpus is broader than a Communications of the ACM (CACM) corpus. Hence, "banana" and "Baltic" may share the context "republic" a number of times, but their similarity is rejected because they share no other context. Similarly, "Dominican" and "banana" share two contexts, "republic" and "plant," but in a broad corpus, this is still not enough. A still more striking example is "phar-

maceutical" and ""aerospace" which are not semantically similar despite being found to share more than six common contexts: "firm," "industry," "sector," "concern," etc. Here the commonness, and hence relative unimportance, of the shared contexts must outweigh the mere number of shared contexts. Strzalkowski et al. also observe that modifiers are more reliable contexts than head terms; hence, in totalling the number of shared contexts for a term, they count a head context as only 0.6 of a context.

A final problem with Strzalkowski's statistical clustering of terms with shared contexts is that it does not distinguish similarities that indicate synonyms ("merge," "buyout," and "acquisition"), and specialization, which generate terms suitable for query expansion, from complements ("Australian" and "Canadian") and antonyms ("accept" and "reject") which are generally *not* suitable for query expansion. Strzalkowski addresses this by defining a "Global Term Specificity" (GTS) measure. The GTS is roughly analogous to the *idf* but is measured over syntactic contexts rather than documents. Moreover, it is only useful for comparing terms that are already known to be similar in terms of context co-occurrence. For such contextually similar terms w_1 and w_2, the assumption is that w_1 is more specific than w_2 if it occurs in fewer distinct contexts; if w_1 and w_2 occur in the same number of distinct contexts, but w_1 occurs in many more instances of those contexts than w_2, it may be more specific. GTS is given by:

$$GTS = \begin{cases} IC_L(w) \bullet IC_R(w) & \text{if both exist} \\ IC_R(w) & \text{if only it exists} \\ IC_L(w) & otherwise \end{cases}$$

where (with n_w, d_w>0):

$$IC_L(w) = \frac{n_w}{d_w(n_w + d_w + 1)} \qquad \text{head is context}$$

$$IC_L(w) = \frac{n_w}{d_w(n_w + d_w + 1)} \qquad \text{modifier is context}$$

Here, d_w is the number of distinct contexts in which *w* occurs (as a modifier for IC_L, as a head for IC_R), and n_w is the number of actual occurrences of *w* in these contexts. If $GTS(w_1)$ is greater than or equal by some appropriate factor than $GTS(w_2)$, then w_1 is assumed to be more specific than w_2. If $GTS(w_1)$ is less than or equal to $GTS(w_2)$ by some appropriate factor and vice versa, then the terms are assumed to be synonymous. Hence, this process leads to clusters of terms that are predicted to be either synonyms, or in the relation that one term is a specialization of the other. These clusters can be used either for automatic query expansion or interactively, to suggest candidate expansion terms to a human user.

DR-LINK also uses statistical techniques to identify terms that are likely to be interchangeable in certain CN contexts, specifically terms that are premodified by the same set of terms in a given corpus. If two terms *a* and *b* are premodified by the same set of terms, there is said to be a "second order association" between *a* and *b.* Hence, this process identifies phrases that are substitutable for each other for purposes of document-topic matching.

9.4 Proper Noun Recognition, Categorization, Normalization, and Matching

The presence of specified proper nouns is often a necessary, though not necessarily sufficient, condition for a document to be relevant to a specified topic. If the topic is the Japanese stock market, then some form of the proper noun “Japan” is clearly essential, although by itself hardly sufficient, since documents might deal with many other aspects of Japan. Names of persons, companies, government agencies, religions, chemicals, and many other entities may be essential to the specification of a topic, and the recognition of documents relevant to the given topic.

Recognition, extraction, and matching of proper nouns is considerably more complex than it might at first seem. A variety of factors complicate the process. Many proper nouns consist of more than one noun, e.g., “Wall Street Journal.” Many proper nouns include a preposition, e.g., “Department of Defense,” or a conjunction, e.g., “John Wiley and Sons.” Many proper nouns can be specified in multiple forms, e.g., “MCI Communications Corp.,” “MCI Communications,” and “MCI.” Many proper nouns are group nouns, which may result in references either to the group as a whole, or to the individual entities making up the group, e.g., “European Community,” “Latin America.” Common nouns and noun phrases may also group individual entities that have proper noun names, e.g., western nations, socialist countries, third world, agricultural chemicals.

Borgman et al. [JASIS] discuss at length the particularly difficult case of names of persons. Conventions for assigning names vary with the culture and historical period. In ancient times, single names were normal. The practice of assigning multiple names, e.g., first, middle, and last names, is more recent. Some cultures use compound surnames, but the conventions vary from one culture to another, e.g., “[h]ispanic children receive a combination of their parents’ surnames, and wives acquire a combination of their maiden surnames, and their husbands’ surnames.” Order of names also varies with culture, e.g., “[a]sians traditionally place the surname first, although asians living in Western nations often report their names with surnames last.” “Personal names may be translated from one language to another, retaining meaning,... or be transliterated from one alphabet or character set to another.” Multiple transliteration schemes exist. People change their names over their lifetime, as a result of marriage, divorce, adoption, or movement from one country to another. People adopt or receive nicknames and diminutives, e.g., “Dick” for “Richard,” “Bob” for “Robert.” A person may use one form of her name on a drivers license, but another form for publication as an author. On top of all this, errors are common, not only typographical errors, which affect any typed input, but phonetic errors, e.g., a person from one cultural or linguistic background transcribing a spoken name from another cultural or linguistic background is especially likely to err.

Paik et al. [ARPA Workshop] [Corpus Proc] have developed a sophisticated series of procedures for proper noun recognition and matching in their DR-LINK (Document Retrieval through LINguistic Knowledge) and KNOW-IT (KNOWledge base Information Tools) IR engines. The proper noun recognition system described here was developed through corpus analysis of newspaper texts. First they assign parts of speech to all the words in the document; then they execute a general purpose noun phrase bracketter, and a special-purpose proper noun phrase boundary identifier. Next the system categorizes all the proper nouns; this is consistent with the DR-LINK emphasis on capturing the conceptual level of a document, as well as the actual keywords and phrases. Topic requests may often be stated at the conceptual level. As Liddy et al. note, “queries

about government regulations of use of agrochemicals on produce from abroad, require presence of the following proper noun categories: government agency, chemical, and foreign country." Note that a document that contains proper nouns in those categories may not actually contain the words "government," "agrochemicals," "produce," or "abroad." DR-LINK attempts to recognize eight categories: Geographic Entity, Affiliation, Organization, Human, Document, Equipment, Scientific, and Temporal; within each of these categories, DR-LINK recognizes two or more sub-categories, for a total of 29 meaningful sub-categories. (A more recent version, embodied in both DR-LINK and another commercial tool, KNOW-IT, recognizes over 60 sub-categories.) "Affiliation" includes "religion" and "nationality." "Human" includes "person" and "title." "Scientific" includes "disease," "drug," and "chemical." And so on. DR-LINK performs this categorization using such clues as known prefixes, infixes, and suffixes for each category, e.g., Dr., Mr., Ms.,and Jr. for persons, Inc. and Ltd., for companies, etc. DR-LINK also uses a database of aliases for alternate names of some proper nouns, and knowledge bases such as gazetteers, the CIA World Factbase, etc. Contextual clues are also used, e.g., if the pattern proper noun, comma, proper noun is encountered, and the second noun has been identified as a state, the first noun (if not otherwise categorized) will be categorized as a city.

Since a given proper noun may take multiple forms, DR-LINK standardizes proper nouns as they are being categorized. That is, all forms of the same proper noun are mapped into a single standard form, to simplify subsequent matching. This is equivalent to stemming of ordinary words, reducing all variants to a common form. However, whereas stemming (at least in English!) largely involves processing of multiple suffixes, standardizing of proper nouns involves standardizing of prefixes, infixes, suffixes, and variant forms of proper nouns, e.g., "Dick" to "Richard." Note that two variant forms of the same proper noun, referring to the same entity, may occur not only in two different documents, or in a document and a topic request, but also within a single document. In particular, an entity may be named in full on its first reference, and mentioned in a more abbreviated form on subsequent references. It is an important instance of reference resolution for a Natural Language Processing (NLP) based IR system to recognize that these are references to the same entity.

DR-LINK also expands group proper and common nouns, so that a topic request can match a document on either the group name or its constituents. For example [Feldman, ONLINE], a request for documents about "African countries which have had civil wars, insurrections, coups, or rebellions" will return not only documents that contain some form of the proper noun "Africa," but also documents containing references to countries within Africa. DR-LINK uses proper noun and common noun expansion databases.

Note that, in a system like DR-LINK, proper nouns can provide several levels of evidence for topic-document similarity computations. First, there is the obvious matching on the names themselves. Second (as noted earlier), there is matching on categories assigned to the names. This category matching is similar to, but supplements, the matching on subject categories (SFC's) described in an earlier section. Third, expansion of group nouns can result in matching a document on proper nouns not actually mentioned in the topic statement. This can work in the other direction too, e.g., if a document mentions Montana or Atlanta, then these references may be used to match the document against a topic that only speaks about "American" companies. Fourth, proper nouns naming geographical entities can provide relationship information, e.g., they can

"reveal the location of a company or the nationality of an individual." Subject information can be combined with proper noun category information for more refined topic-document matching. A report of a merger should involve (at least) two proper nouns of category "company," while a report of an invasion is likely to specify two geographic entities, most likely at the level of country or province.

A significantly different approach to proper noun recognition is taken by Mani et al. [Corpus Proc, 1996] Their approach differs somewhat both in goals and methods. They focus on a much smaller set of subject categories: people, products, organizations, and locations. Within large text corpora, they seek (like Paik) to categorize previously unknown names automatically. However, they attempt to go further than Paik, extracting from the text appropriate semantic attributes for each named entity, e.g., the occupation and gender of a person. A given entity may be mentioned more than once in a given document, and each mention may employ a different variation of the entity's name, e.g., "President Clinton," "Bill Clinton," "Clinton," "the president," etc. They seek to "unify" these mentions, i.e., to recognize all mentions to the same entity, and to combine the attributes associated with these varied mentions into one common schema describing the given entity. This is called "coreference resolution" for proper nouns. When two mentions (and their associated attributes) are successfully unified as referring to the same entity, they are said to be "coanchored." Note that this goes considerably beyond (although it includes) the normalization of proper nouns performed by Paik.

Coreference resolution is closely tied to attribute extraction. On the one hand, attributes extracted from one mention of a given entity can be combined with attributes extracted from another mention, to fill out as many of the "slots" associated with the given type of entity as possible. For example, one mention may indicate that Clinton's occupation or title is "president." Another mention may indicate that his gender is "male." On the other hand, extracted attributes can serve as evidence to determine whether two mentions refer to the same entity, or to two distinct entities. For example, if "President Clinton" has been associated with the gender attribute value "male," and "Hilary Clinton" has been associated with the gender attribute value "female," this is evidence that these mentions do *not* refer to the same entity. But, "President Clinton" and "Mr. Clinton" will match on gender, and hence will be coreference candidates unless additional evidence indicates a contradiction. Moreover, attributes may also serve to indicate whether two mentions refer to distinct but related entities. For example, "Bill Clinton," "Hilary Clinton," and "the Clintons," are distinct, but related entities. As a further refinement, Mani distinguishes between "discourse pegs," i.e., entities that are distinct in a given discourse, and entities that are distinct in the real world. For example, President Clinton, and ex-Governor Clinton may be two distinct discourse pegs for purposes of analyzing a given document, although they refer to the same real-world object in the world model or belief system of an external knowledge base.

As Mani encounters new proper noun mentions in the text of a given document, he naturally wants to limit the number of earlier mentions that must be evaluated as possible candidates for coreference. He does this by indexing each mention by *normalized name* (a standardized form, analogous to Paik), by *name elements* in its name (individual words within the name), and by its *abbreviations*. Only mentions that match on at least one of these indexes are coreference candidates. Abbreviations are generated by rule, or retrieved from a lexicon; hence, a full name in one mention can be matched against an abbreviated name in another.

Another difference between the Mani and Paik approaches is that Mani makes greater use of the context surrounding a proper noun, and of the discourse structure of successive mentions. In particular, Mani makes use of both honorifics and "appositive phrases," phrases adjoining and identifying a proper noun. It is a widely used convention, especially in news stories, to attach an honorific or an appositive phrase to the first mention of a given name, e.g., "Anthony Lake, Clinton's national security advisor," or "Osamu Nagayama, 33, senior vice president and chief financial officer of Chugai," or "German Chancellor Gerhard Schroeder." Such appositives and honorifics are generally employed whenever the named entity is not a "household name," and is not sufficiently identified by title. It is applied to entities other than persons, especially organizations and locations, e.g., "X, a small Bay Area town." (Paik indicates that one of their intended research directions is the use of appositive phrases. However, in one knowledge base derived by KNOW-IT from New York Times articles, Anthony Lake was erroneously categorized as a body of water, presumably because the appositive phrase was ignored or misinterpreted.) Mani identifies candidate appositive phrases by pattern matching based on left and right delimiters such as commas and certain parts of speech. Syntactic analysis is then used to extract key elements, e.g., a head or premodifier, from the given phrase. In the "Nagayama" example above, "senior vice president" would be extracted, and looked up in a semantic lexicon ontology, which identifies the title as a "corporate officer." Plainly, the value of such appositive phrases for categorization depends on the availability of lexicons that enable one to interpret their semantic content.

Another distinctive feature of the Mani methodology, closely related to the gathering of evidence over multiple mentions of an entity, is the explicit handling of uncertainty. Evidence gathered in one mention can reinforce or contradict evidence gathered in another mention. Mani employs a variety of Knowledge Sources (KS's). KS's are little rule-based programs that attempt to categorize ("tag") entities. Many of the rules employed by Mani's KS's are similar to the rules employed by Paik's system, e.g., one KS attempts to identify organizations by using suffixes such as "Inc." and "Ltd." Another tries to identify persons by looking for titles and honorifics, e.g., "Mr.", "Lt. Col.", "Ms.", etc. Other KS's use lexicons, e.g., organization lexicons, gazetteers as geographic lexicons, etc. On the basis of the evidence it collects, a KS can generate multiple hypotheses with different confidences.Mani offers the example that "General Electric Co." may generate one hypothesis that the entity named is a person, with "General" as a title, while other hypotheses may be that it is an organization or a county, based on the abbreviated suffix "Co.". On the other hand, multiple KS's may generate the same hypothesis based on different evidence, e.g., one KS may hypothesize that the given mention is an organization based on the "Co." suffix; another KS may generate the same hypothesis based on the presence of the name in an organization lexicon. A "Combine-Confidence" function computes the confidence of a given hypothesis about a given mention as the weighted sum of the probabilities assigned to the hypothesis by each KS that contributed to it, each probability weighted by the reliability of the KS that generated it.

The confidence values associated with hypotheses play an important role in mention unification. If two person mentions have conflicting hypotheses about the occupation slot, but one hypothesis has a much lower confidence than the other, unification may succeed. On the other hand, if two mentions have conflicting gender hypotheses, and these hypotheses both have high confidence values (e.g., based on the honorifics "Mr." and "Mrs." respectively), the unification will fail.

9.5 Semantic Descriptions of Collections

In a later section, the fusion of IR results from multiple collections is discussed. However, all the cases discussed there assume that the set of collections to be accessed is known, preferably in advance. If the set of collections is very large, diverse, and dynamic, e.g., the Internet, this assumption no longer holds in general. In such cases, IR becomes a two-stage process, i.e., first find an appropriate set of collections, and then apply IR techniques such as those discussed in this paper. The process of finding the "right" collections becomes more manageable (though far from trivial) if each candidate collection includes, or is assigned, a formal machine-readable description of its contents. (The problem of searching and indexing the Internet - or a large Intranet - in the normal case where such standardized formal descriptions are not available, is discussed in a later section.) The first stage of the IR process then becomes the matching of a given query against a "collection" consisting of these collection descriptions. Chakravarthy and Haase [SIGIR '95] explore a case where structured collection descriptions (they call the collections "archives") with semantic content are created manually using WordNet, and then natural language queries are translated automatically (using syntactic/semantic techniques, an on-line Webster's dictionary, and WordNet again) into a structured form that can be matched against the archive descriptions. They report that their system, NetSerf, has a "database" that currently contains descriptions (they call them "representations") of "227 Internet archives. Most of these are from two sources, the Whole Internet Catalog [Krol, 1992] and the Internet Services List [Yanoff, 1993]."

An archive description ("representation") consists of <relation-type, relation-word> pairs. Relation types illustrated by the authors in their examples include: TOPIC, INFO-TYPE, OBJECT, AUTHOR, PERTAINS-TO, IN, and HAS-OBJECT. "For each relation-word, NetSerf uses WordNet to identify all its synsets." The relation word is disambiguated by the synsets containing the word that are *not* chosen. Chakravarthy and Haase offer the example of the World Factbook archive, described in natural language as, "World facts listed by country." The TOPIC is "country," and the INFO-TYPE is "facts." Three of WordNet's four synsets are assigned to the relation-word, "country:"

SYNSET: {nation, nationality, land, country, a_people}

SYNSET: {state, nation, country, land, commonwealth, res_publica, body_politic}

SYNSET: {country, state, land, nation}

A fourth synset of "country," [rural area, country] is omitted since it corresponds to a sense of the word "country" that is obviously inapplicable here.

The authors plan to explore in the future the automatic construction of such descriptions from sources such as home pages and README files.

Query processing in NetSerf starts with a natural language query. "The query processor makes the assumption that the query, after preprocessing, consists of one or more topic words followed by prepositional phrases and verb clauses that modify either the topic words or preceding modifiers." Manual rephrasing is sometimes necessary, e.g., where the original query takes the form of two

sentences. The query is then “tagged,” i.e., a part of speech is assigned to each word or other lexical “token.” Common query introductions such as “What is” are deleted. Words or phrases identifying the leading information type are extracted, e.g., given the query “satellite photographs of hurricane’s progress,” the information type “satellite photographs” is extracted. Topic words and modifiers are extracted and cast into <relation-type, relation-word> form. The relation type is determined by the syntactic type of the modifier. A word sense disambiguator based on neighboring relation words in the original textual query, WordNet hypernyms, etc., is executed. Finally, the main topic relation words are expanded “using semantic relations from the dictionary” that are “extracted using a pattern definition language.” For example, given the topic relation-word “pub” and its dictionary definition, the query processor generates the <relation-type, relation-word> pair: <PERTAINS-TO, “alcoholic beverage”>.

NetSerf queries are matched to archive representations. Query relation-words are matched against archive description relation-words. A “hit” occurs “if some valid synset of some relation word in R [the archive representation] is a hypernym of some valid synset of some relation word in Q. [the NetSerf query]” A positive weight is added for every hit where the relation types match. A negative weight is added for every hit where the relation types do not match.

Chakravarthy and Haase found that “structured representations [of archives, i.e., the archive descriptions], and semantic knowledge-based matching lead to significant improvements.” On the other hand, sense disambiguation led to a slight degradation of performance.

9.6 Information Extraction

One of the most important areas of IR where NLP plays a crucial role is *information extraction* (IE). IE is the extraction of information from a collection of documents in response to a query.

IE must be clearly distinguished from *document retrieval* (DR), and *document summarization* (DS). DR, the focus of much of the research described in this report, is the retrieval of *documents* relevant to a given query or topic. The document set retrieved may be relevance ranked or not. Either way, what the user receives is a set of documents believed to be relevant to the user’s need. DS is similar to DR except that the system generates a summary of each retrieved document. This summary may be a few sentences or paragraphs (perhaps modified syntactically achieve greater reading “smoothness” and “continuity”) believed to capture the essence of what the given document is about, or a set of key words believed to suggest the document’s essence. In either case, the retrieval is document-based. Indeed, the distinction between DR and DS systems is often not clear-cut. A DR system seldom returns a list of documents directly. Rather, it typically returns a list of document identifiers, perhaps accompanied by relevance scores. These identifiers may be titles, subject lines, summaries, etc. The user can then request the actual text of a given document by selecting its identifier. On the other hand, a DS system returns a list of summaries. The DS system may allow the user to expand on the summary by requesting the document from which the summary was extracted.

By contrast, an IE system returns to the user *information* (*not* a document list) responding to the user’s information request (query). The response may be generated from multiple documents, or from a combination of a document and a database entry. Moreover, note that the term is *informa-*

tion extraction, not *text* extraction. This implies that the "answer" generated by the IE system does not necessarily consists of text literally extracted from the document, with only minor syntactic tinkering if any. Instead, the answer may be (as with the systems participating in the Message Understanding Conference - MUC [Darpa, 1992] competition) a template in which slots have been filled in. The template is usually generated manually in advance of the IE competition, or execution of an IE application. The names of the template fields do not necessarily appear in any document from which relevant data is extracted. The IE system must "understand" (typically using simple linguistic clues) that a term appearing in the text of a document, e.g., a human or corporate name, is an appropriate value for a given field of the template, and "fill in" the value of the field with the extracted value. Hence, the answer provided to the user is a mixture of manually generated template names, and extracted text values. Moreover, if a person is identified from information in a text document, the template may be filled in with a combination of information about the given person extracted from the given document, and additional information extracted from an entry about the person in a structured database.

Alternatively, the answer returned to the user may be textual, e.g., a sentence or paragraph. However, the sentence may be a combination of extracts from two sentences in the original document, linked by co-reference resolution. For example, the first sentence may identify a person by name and title. The second sentence may refer back to the individual by pronoun, e.g., "he announced that …." Hence, the textual answer generated and returned to the user never actually appears explicitly in the text of the given document.

Note that, even in an IE system, e.g., KNOW-IT [Liddy, WP, 1999], the user may be given the option of going back to the original document(s) from which the answer was extracted.

Ideally, an IE system would "understand" the documents it reads (just as a human library researcher would), extract and condense all the information relevant to the user's request, and return a single, coherent, comprehensive answer. In actual fact, such a strategy is far beyond the state of the art. The time required to compile the knowledge and logic required for such a level of processing, even for a relatively narrow subject domain, would be prohibitive. Moreover, such a level of logical analysis would be far too elaborate and slow for processing the huge document collections available today in libraries and on the Internet.

Hence, many IE systems use statistical part-of-speech tagging followed by "shallow parsing," (also called "partial parsing"). [Cowie et al., 1996] The term "shallow parsing" refers to high-speed parsing techniques in which only key fragments of a sentence are parsed. The phrase "shallow knowledge" refers to relatively simple ad hoc rules, often tailored to the needs and characteristics of a given domain, and supplemented by large lexicons, machine readable dictionaries (MRD's) and other on-line reference sources.

Part-of-speech tagging is the process that assigns to each word in a sentence the grammatical role, e.g., noun, verb, adjective, determiner, etc., that the given word plays in the given sentence. This role is called its "part of speech." "[M]ost English words have only one possible part of speech [but] many words have multiple possible parts of speech and it is the responsibility of a tagger to choose the correct one for the sentence at hand." {Charniak, 1997] For example, the word "can" can be a modal verb, noun, or verb. Statistical parsers are typically trained on a tagged corpus. In

the simplest form, the parser will simply know which part of speech is most common for the given word in the training corpus. A more sophisticated statistical tagger uses context, e.g., knows which part of speech is most probable for a given word when the word occurs following some other part of speech. For example, taken by itself, the most probable part of speech for “can” is the modal-verb. However, if the word “can” follows the determiner “the,” the part of speech “noun” becomes far more probable.

Shallow parsing allows an IE system to “skim” over a sentence, only parsing the most critical fragments, rather than generating a complete parse tree for the entire sentence. Parsers that attempt to parse a sentence fully “typically operate in polynomial time and tend to get bogged down with sentences containing more than 20 to 30 words.” [Cowie et al. 1996] Moreover, complete parsers generate a great many legal parses for a single sentence of significant length. For many IE tasks, the output of a partial parser is quite adequate, identifying critical subjects, objects, proper noun categories, etc.

Shallow knowledge is domain-specific knowledge, typically consisting of ad hoc rules that work in, but perhaps only in, the given domain. As an example of how narrowly tailored these IE rules can be, consider these examples from the U.Mass/MUC-4 [MUC-4] [Cowie et al., 1996] system for extracting events in the domain of Latin American terrorism (the rule numbering is arbitrary):

Rule 1: The direct object of “robbed” (active voice) is the victim of a robbery.

Rule 2: The subject of “disappeared” (active voice) is the victim of a kidnapping.

Rule 3: The object of “in” after traveling (active voice) is the target of an attack.

Rule 4: The subject of “hurled” passive voice is the instrument of an attack.

Rule 5: The subject of “placed” is the instrument of a bombing.

The rules were evidently derived from the text corpus. Rule 1 is fairly general, and might apply outside of the target domain. On the other hand, the other rules would obviously break down badly outside the intended domain. Consider the use of Rule 4 in a “baseball” domain! Rules 3 and 5 are even more specialized. Obviously, in most domains, traveling in a vehicle carries no implication about an attack at all, let alone who the target of the attack is. Similarly, in most domains, objects can be “placed” without any implication that they are bombing instruments. It is clear that the use of such rules requires a two-step process. First, IR techniques must be employed to locate the documents (or passages within documents) that are likely to be about terrorism. Only when the corpus has been narrowed down in this way do rules such as those above stand any chance of working. Yet within the intended domain, such naive rules have been found to work fairly well.

Note that because ad hoc shallow knowledge is only useful in the specific domain for which it was developed, it is usually the case that it cannot be re-used in another domain. Therefore, the development of a shallow knowledge base makes sense only if it can be generated so rapidly and inexpensively that it can be treated as “a disposable artifact.” In other words, the assumption

underlying the shallow knowledge approach is that it is easier and cheaper to create a shallow knowledge base for each new domain that comes along, than to create a deep knowledge base that can be re-used for many domains.

The opposite extreme is represented by the Cyc project [Lenat et al., 1989], the goal of which is to create a huge KB of commonsense knowledge of the world, the kind of knowledge that is not explicitly represented in encyclopedia and other reference books because it is knowledge that "everybody knows," knowledge that is taken for granted, but knowledge that expert systems and IE systems tailored to a specific domain do not possess. Cyc research has demonstrated that building such a KB is a very long-term, expensive, difficult affair. It should be noted that the Cyc approach has been to enter knowledge manually, although the hope has always been that Cyc would reach a critical mass at which it could begin acquiring knowledge automatically from large textual sources.

The KNOW-IT system represents an intermediate approach to IE. It is not tailored to any particular application or knowledge domain. Like its technical and commercial relation, DR-LINK, it supports a broad hierarchy (60+) of proper noun categories, in a hierarchy eight levels deep. Similarly, it supports generic semantic relationships, such as "affiliation, agent, duration, location, or point in time," as opposed to relations specific to a particular domain such as terrorism, e.g., relations "such as 'weapons used' or 'victim'." The KNOW-IT approach takes advantage of the "common practice among writers of including predictable information-rich linguistic constructions in close proximity to related proper names." Hence, KNOW-IT identifies and categorizes proper nouns, then identifies generic relationships among the concepts embodied by those proper nouns, so-called Concept-Relation-Concept (CRC) triples. (In common with most other IE systems, KNOW-IT also performs part-of-speech tagging to all the words in the text, assigning one of 48 possible grammatical tags, such preposition, determiner, or singular noun.) The user can display the concept structure graphically, penetrate from higher to lower levels of the concept hierarchy, until actual proper nouns are reached. She can also display the relationships in which concepts or proper nouns participate, the documents in which those proper nouns and relationships occur, the sentences in which they occur, and finally, the full text of the documents in which the sentences occur. DR-LINK and KNOW-IT were originally developed for newswire text, but KNOW-IT has been extended to document types as diverse as technical manuals and WWW home pages.

The methods used by KNOW-IT for proper noun recognition and categorization are those developed by Paik [1993] for the DR-LINK project, as described earlier in the section on proper noun techniques. The CRC triples are the same as those widely used in Conceptual Graph studies, as developed by Sowa and others. CG's are discussed earlier in the section on formal concepts, and again in the section on relations and conceptual graph mapping. The former discusses Wille's work on the equivalence of CG's and formal contexts in Formal Context Analysis (FCA). The latter discusses the work on CRC triples and CG's in the DR-LINK system, closely related to KNOW-IT.

Although the basic KNOW-IT approach is domain-independent, it can and in some cases has, been extended with knowledge of some specialized domain, e.g., international politics.

10 Clustering

"Clustering" of documents is the grouping of documents into distinct classes according to their intrinsic (usually statistical) properties. Clustering is a kind of classification but it differs from the classification for routing purposes discussed in the section above on routing in one crucial respect: In a routing application, the documents are classified in terms of their similarity or relevance to external queries or topics or user profiles. In "clustering," we seek features that will separate the documents into natural groups based entirely on the internal properties of the collection. Ideally, the groups will be completely separate and as far apart as possible in feature space. But sometimes, overlap of clusters is unavoidable. [van Rijsbergen, 1979] Since clustering depends on the statistical properties of the collection being clustered rather than on matching the documents against some external set of queries, it is normally (but not always - see below!) applied to a pre-existing collection rather than an incoming stream of documents as in a routing application.

Why should documents be clustered? The basic reason is that clustering can reveal the intrinsic structure of a collection, e.g., by topic, subtopic, etc., (assuming of course, that there *is* a significant internal structure). If a language-independent statistical method such as "*n*-grams" is used, a collection may also be clustered by language or document type, by topic within language, etc. (See section 3.3.6.) Moreover, by the "cluster hypothesis," "closely associated documents tend to be relevant to the same requests." [van Rijsbergen, 1979] Document clustering of a large collection is particularly effective when it is *hierarchical*, i.e., when the collection is partitioned into (relatively) large, high-level clusters corresponding to broad categories, each high-level cluster in turn clustered into smaller clusters corresponding to tighter, more cohesive categories, which in turn are composed of still smaller, still more cohesive clusters, and so on. Ideally, the lowest level clusters in such a hierarchy will consist of documents that are very similar, e.g., that are all relevant to most of the same topics or queries. Hence, clustering, especially when combined with modern graphical display techniques, can be an effective tool for browsing a large collection and "zeroing in" on documents relevant to some given topic or other criterion. For similar reasons, it can increase the *effectiveness* of document retrieval, i.e., of querying large collections. [Willetts, IP&M, 1988]

Searching a hierarchically clustered collection can proceed either *top-down* or *bottom-up*. Top-down searching proceeds as follows:

> A top-down search of the cluster hierarchy is performed by comparing (using a similarity measure) the query to cluster representatives [e.g., centroids] of the top-level (largest) clusters, choosing the best clusters, comparing the query with representatives of lower-level clusters within these clusters, and so on until a ranked list of lowest-level clusters is produced. The documents in the top-ranked [of these lowest-level] clusters are then ranked individually for presentation to the user. [Belkin & Croft, ARIST, 1987]

The biggest problem with a top-down search is that the highest-level clusters may be so large and loosely coupled that a representative of such a cluster may bear little resemblance to most of the documents in the cluster. Hence, choosing a cluster at the highest levels becomes almost arbitrary

and the search procedure is likely to choose a search path that misses the relevant documents. Hence, top-down searches work best when the clustering method and associated threshold ensure that even the highest-level clusters are reasonably small and cohesive.

A bottom-up search is the inverse of a top-down search. If one starts at the bottom, the clusters should be smaller and more cohesive than at the top. The problem is which one of those bottom-level clusters to choose as a starting point. One approach is to do a conventional query search to find one relevant document. Then one can start the cluster search with the cluster containing that document. Or, one can do a conventional query search to match the query against the representatives of all the bottom-level clusters. The cluster representative that is most similar to the given query then determines the starting cluster. Cluster searching then proceeds upward until a cluster is reached containing the number of documents the user wants to retrieve. [Willett, IP&M, 1988]

Of course, there is no guarantee that the cluster hypothesis is widely satisfied. It can only be verified empirically in any given collection. (See section on cluster validation.) In general, it is possible that a given algorithm will not generate any clusters, or that the clusters will overlap too much to be useful, or that the clusters which are formed will not correspond to meaningful topics of interest to prospective users. In an earlier section on relevance feedback (see above), another possible complication was pointed out: the documents relating to a given topic may form not one, but two or more separate clusters. However, it should be noted that all statistical IR techniques assume that it is possible to separate a collection of documents into at least two classes with respect to any given query, i.e., relevant and non-relevant documents.

Hierarchical clustering offers the potential for very fast retrieval search time since most of the searching involves cluster representatives rather than all the documents in each cluster. However, if documents are fully indexed, vector space or boolean retrieval without clustering may provide retrieval time as fast or better since the only documents that need to be searched are those whose terms match the terms in the given query. If the query is well-formulated, these documents may be only a very small fraction of the collection.

The great virtue of most proposed clustering methods is that they are automatic. Of course, manual assignment of useful categories to each document in a collection is more certain to be useful, but it is very time-consuming and requires substantial manpower for large collections. Automatic clustering offers the hope of eliminating much of this effort. In the preceding section, an intermediate approach was described: Subject categories were assigned manually to a training set; thereafter, categories were assigned to new documents automatically.

Clustering requires some measure of the similarity between documents in document space. The widely used cosine similarity described earlier is an obvious choice. But other similarity measures are available. [van Rijsbergen, 1979] [Salton & McGill, 1983] [Korfhage, 1997] (See the discussion of document query similarity above.) Measures of dissimilarity can also be used, especially since the objective is to maximize the distance between clusters, e.g., between their centroids, in document space. A dissimilarity measure is, in essence, a distance measure, i.e., its value for any two documents D_1 and D_2 is greater the farther apart D_1 and D_2 are in document space. (Cosine similarity has the opposite property; it has its maximum value when two document vectors coin-

cide, and has a value of zero when the document vectors are orthogonal. But 1 - cosine is a distance measure, increasing with *angular* distance.)

Document clustering methods are generally distinguished from the descriptors used to characterize each document, and the similarity (or dissimilarity) function of those descriptors that is used by the clustering method. In general, the choice of inter-document similarity measure is independent of the choice of clustering method. And the choice of a similarity measure is (often) independent of the choice of the document descriptors that serve as independent variables of the similarity function. [Willett, IP&M, 1988] (But note that this is *not* true of the method of Zamir et al., described below.)

In general, research into clustering has focused on clustering methods, and algorithms for implementing these methods efficiently with respect to space and time. Willett concedes that the "similarity coefficient may affect the clustering that is obtained." The method of Zamir et al. [SIGIR 98], described below, provides at least preliminary evidence that a novel document descriptor, combined with novel inter-document and inter-cluster similarity functions, can produce dramatic improvement in cluster quality.

Two main strategies have been used for clustering. Both require a document-to-document similarity measure. The first strategy requires that the complete collection of documents to be clustered be available at the start of the clustering process. Hence, one may call this the "complete" strategy. (One might also call it the "static" strategy, since it assumes that the collection of documents is not going to change, i.e., documents are not going to be added or deleted, during the clustering process.) Most methods based on the complete/static strategy start by generating a list containing the similarity of every document pair in the collection. Hence, if the collection contains N documents, the list will contain $N(N\text{-}1)/2$ similarities. Methods using this "complete" strategy are expensive because of the large number of similarities involved, and the large number of comparisons that must be performed as documents are combined into clusters, and clusters are combined into larger clusters. A straightforward implementation requires that the interdocument similarity matrix, containing O(N2) elements, must be searched O(N) times, once for each "fusion" of two clusters. Hence, the time requirement is O(N3) and the space requirement is O(N2). More sophisticated implementations have reduced the time requirement to O(N2), but even this is prohibitive for document collections of realistic size such as the larger TREC collections. On the other hand, because these "complete" cluster methods take advantage of the full set of inter-document similarities, they meet van Rijsbergen's three criteria for theoretical soundness: [van Rijsbergen, 1979] [Salton, 1989] (1) The clustering is unlikely to be altered drastically as new documents are added, (2) Small errors in the description of documents lead to small errors in clustering, and (3) the method is independent of the initial order of document reference, e.g., the order of document pair similarities. Essentially, these methods "attempt to seek an underlying structure in the data [rather than] impose a suitable structure on it." [van Rijsbergen, 1979] Moreover, the expense is incurred mainly (apart from updates for new documents) when the collection is indexed, not at document retrieval time. However for realistically large values of N, the execution time and storage requirements even for pre-processing is prohibitively large. [Willett, IP&M, 1988]

The second major approach is the "incremental" strategy. Incremental methods [Zamir et al., SIGIR 98] [Salton, 1989] assume that the document collection to be clustered is arriving in a

stream as the clustering proceeds. Hence, as each document arrives it is added to some cluster, or becomes the seed of a new cluster. When document *i* arrives, the *i-1* documents that preceded it are already clustered. The *i*-th document may be added to one of those existing clusters.It may become the seed of a new cluster. Or the existing clusters may be re-clustered in the process of adding the *i*-th document. If reclustering is not allowed by the method, i.e., if the *i*-th document must be added to one of the existing clusters, the method may be termed single-pass, not just incremental.

10.1 Hierarchical Cluster Generation ("Complete/Static" Methods)

A *complete* algorithm may start by considering all the documents as a single cluster and then breaking it down into smaller clusters ("divisive" clustering). Or, the algorithm can start with the individual documents and group them together into progressively larger clusters ("agglomerative" clustering). (Since agglomerative clustering produces a hierarchy of clusters grouped into larger clusters, it is often called Agglomerative Hierarchical Clustering, or AHC for short.) In the latter case, the similarities are sorted in descending order. Initially, each document is considered a separate cluster. The general rule is that at each stage the two most similar clusters are combined. Initially, the most similar documents are combined into a cluster. At that stage, "most similar" means having the highest similarity of any document pair. Thereafter, we need a criterion for deciding what "most similar" means when some of the clusters are still single documents and some are multi-document clusters that we have previously formed by agglomeration (or when all of the clusters have become multi-document). The various agglomerative cluster methods are distinguished by the rule for determining inter-cluster similarity when one or both clusters being compared are multi-document. [Willett, IP&M, 1988]

In "single-link" clustering (the most famous clustering method), the similarity between two clusters is defined to be "the similarity between the *most similar* pair of items, one of which appears in each cluster; thus each cluster member will be more similar to at least one member in that same cluster than to any member of another cluster." The algorithm is called "single-link" because two clusters can be combined on the basis of one high similarity between a document in the one cluster and a document in the other. It is also called "nearest neighbor" clustering because two clusters are combined on the basis of the two documents, one from each cluster, that are nearest to each other. Hence, each cluster is formed by a chain of nearest neighbor document-to-document single links.

In "complete-link" clustering by contrast, the similarity between two clusters is defined to be "the similarity between the least similar pair of items" one of which appears in each cluster. "[T]hus each cluster member is more similar to the most dissimilar member of that cluster than to the most dissimilar member of any other cluster." [Salton, 1989] Hence, in the "complete-link" algorithm, the similarity between two clusters (which determines whether they should be combined or not) depends on *all* the similarities between documents in the one cluster and documents in the other.

For any clustering method a similarity threshold can be applied, i.e., two clusters will be combined only if their inter-cluster similarity is greater than some threshold, *T*. Or the clustering process can be terminated when some halting criterion is reached, e.g., a pre-specified number of clusters. In the case of agglomerative clustering, the number of clusters is successively reduced

until all clusters have been combined into a single cluster, the "root" of the hierarchy. If a halting criterion is specified, the agglomeration may stop when the criterion is satisfied, e.g., when successive levels of clustering have reduced the number of clusters to a specified value NC_H. Zamir et al. note that these AHC algorithms "are very sensitive to the halting criterion - when the algorithm mistakenly merges multiple 'good' clusters, the resulting cluster could be meaningless to the user."

Note that by either criterion, it is quite possible for a document in cluster C_1 to be more similar to *some* documents in cluster C_2 than to *some* other documents in C_1. Ideally, one would want to impose the criterion that every document in C_1 is closer to every other document in C_1 than to any document in any other cluster C_2. However, such strict criteria for "cohesion and isolation" of clusters appear to be too strict; in experiments, "very few sets could be found to satisfy" such criteria. [van Rijsbergen, 1979]

"Complete-link" clustering is considerably more computationally expensive (in either space or time but not both — there is a trade-off) than "single-link" clustering, but it has the advantage that one can generate clusters such that every pair of documents in a given cluster is above a specified similarity threshold. A document D_i whose similarity to any other document D_j is lower than the specified threshold will not be in *any* cluster. By contrast, in "single-link" clustering, the members of a cluster are chained together by "single links" such that the similarity between any two documents connected by a link, i.e., any two documents that were nearest neighbors at some stage of cluster combination, is guaranteed to be above the specified similarity threshold, but a pair of documents that are both in the same cluster but which were not directly chained together by the clustering process are *not* guaranteed to be above the threshold. Hence, single-link clustering allows document pairs with very low similarity to be in the same cluster. For that reason, complete-link clustering is probably better suited to IR applications. [Salton, 1989] In particular, complete-link clustering tends to produce small, tightly bound, cohesive clusters, whereas single-link clustering tends to produce large, loosely-bound, "straggly" clusters. [Willets, IP&M, 1988]

A third agglomerative clustering method, "group-average" clustering, is intermediate between single-link and complete-link in that each member of a cluster has a greater *average* similarity to the remaining members of the that cluster than it does to all members of any other cluster.

A fourth agglomerative clustering method, Ward's method, "joins together those two clusters whose fusion results in the least increase in the sum of the [Euclidean] distances from each document [in the fused cluster] to the centroid" of the cluster. Evidently, this method is only defined when Euclidean distance is used for computing interdocument similarity. The centroid of a cluster is the average of the document vectors comprising the given cluster.

The defining characteristic of a cluster *method* is the rule that defines the clusters. For example, if the method is *single link clustering*, then the defining cluster rule is the one stated above: Two clusters are combined into a single cluster, if their closest members (according to the given interdocument similarity measure) are closer (more similar) than either is to the closest member of any other cluster.

In general, each of these cluster *methods* should be distinguished from the *algorithms* that have been developed to implement it since a given method often can be implemented by many different algorithms, each algorithm having its own distinct performance characteristics regarding space and time. [Willets, IP&M, 1988] For example, many algorithms are known that produce single-link clusters. They are all the "same" from the outside, i.e., given the same *N* documents, they will produce the same hierarchy of clusters. However, they may vary considerably in their space and time requirements. The SLINK algorithm [Sibson, 1973] has been shown to achieve optimal performance for single link clustering: $O(N^2)$ time complexity and $O(N)$ space. Trade-offs are possible. The complete-link method has the same time complexity, $O(N^2)$, as single-link, if it also has access to $O(N^2)$ space; however, if it only has $O(N)$ space, it requires $O(N^3)$ time complexity. [Voorhees, PC]

10.2 Heuristic Cluster Methods

The term "heuristic" has been used by authors such as van Rijsbergen [1979] (he also uses the term "ad hoc") to characterize methods that take shortcuts to achieve greater efficiency in terms of space and time requirements. In particular, such terms refer to cluster methods that do *not* generate or do not access the full $O(N^2)$ set of interdocument similarities in a collection of *N* documents. Such methods effectively make fewer (sometimes far fewer) effective "passes" through the interdocument similarity matrix or its equivalent. Heuristic methods tend to violate van Rijsbergen's three criteria for theoretical soundness. In particular, it is characteristic of many such methods that the clusters for a given set of *N* documents vary depending on the order in which documents are initially referenced. In the case of the "Buckshot" method discussed below, the method starts with a random sample of the *N* documents; hence, the clusters produced will vary from one execution of the clustering method to another as the random sample varies.

In return for sacrificing theoretical soundness (and correspondingly, reducing one's confidence that the true underlying structure of the collection has been captured), these methods tend to execute in $O(N)$ time. Actually, some of them execute in $O(kN)$ time (sometimes called *rectangular* time bounds, see below), where *k* is equal to the number of clusters desired, or proportional to the number of clusters. If *k* is a constant, especially a small constant, these are linear time methods. However, if the *k* is proportional to *N* (as for some methods and purposes it may be), then the method becomes $O(N^2)$ after all.

The distinction between *heuristic* and *non-heuristic* methods cuts across another distinction: that between *incremental* and *non-incremental* methods. Briefly, incremental methods operate entirely in "update" mode, generating and modifying clusters "on the fly" as each document is accessed. Below, we discuss some non-incremental heuristic methods. In the next section, we discuss incremental methods in some detail. Purely incremental methods tend also to be heuristic, e.g., single-pass methods and the like. However, we discuss one novel incremental method, STC, that succeeds in being linear time *and* non-heuristic.

In one respect, the very term "heuristic" is misleading with respect to clustering methods. In conventional usage, the concept of a "heuristic" algorithm or method of solving a problem implies that there is a perfect solution to the given problem. Heuristic methods are, by definition, not guaranteed to find that perfect solution. One employs a heuristic method only if methods that are guar-

anteed to find the perfect answer are either (a) unknown, or (b) computationally impractical, e.g., far too costly with respect to space or time, or not guaranteed to terminate at all. However, in the realm of document clustering, a "perfect" result is not defined. In general, the "best" set of clusters depends on the objective for which clustering is being performed, e.g., classification, information retrieval, browsing, thesaurus generation, etc. Moreover, the best cluster method even for a given objective may depend on the statistical characteristics of the collection. Hence, even the so-called "complete" or $O(N^2)$ methods are not guaranteed to produce the "best" result.

A heuristic algorithm, associated with Rocchio, [Rocchio, 1996] was developed on the SMART project. It begins with applying a density test to each document that has not yet been clustered, thereby identifying "cluster seeds," documents "that lie in dense regions of the document space, that is items surrounded by many other items in close proximity." [Salton, 1989] For example, density may be defined by requiring that n_1 documents have a similarity (Rocchio typically used cosine similarity) of at least p_1, and n_2 documents have a correlation of p_2. All documents sufficiently similar to a seed, i.e., having a similarity to the seed that exceeds a pre-specified threshold, form a cluster. Clusters may overlap, i.e., a document may be assigned to more than one cluster. [van Rijsbergen, 1979] In a second, iterative stage, the clusters formed in the first stage are adjusted to conform to certain pre-specified parameters, e.g., minimum and maximum documents per cluster, number of clusters desired, degree of overlap permitted, etc. Documents too far removed from cluster seeds, or occupying regions insufficiently dense, remain unclustered, or undergo a separate clustering process in a third stage.

Much more recently, Cutting et al. [SIGIR 1992] have developed two linear-time heuristic clustering methods, called *Buckshot* and *Fractionation*, respectively. More precisely, they are rectangular bound, i.e., $O(kN)$, methods. These clustering methods have been developed for use in an interactive browsing technique called "Scatter/Gather." (See section on User Interaction below.) For this application, k is small and constant, e.g., eight in the examples given by the authors. Hence, $O(kN)$ is practical and effectively linear. Moreover, Cutting et al. [SIGIR 1993] have developed an enhancement to Scatter/Gather that works even for very large corpora, e.g., gigabytes. Hence, Buckshot and Fractionation continue to be useful even for TREC-sized corpora.

The essential "trick" of Buckshot and Fractionation is to use one of the "complete" $O(N^2)$ clustering methods, but to apply it very incompletely to the original large corpus. The result is to generate rough clusters very rapidly. The centroids of these clusters then become "seeds" around which the entire corpus can be clustered on the basis of simple document-centroid similarity.

For example, in Buckshot, clustering is applied initially to a random sample of $\sqrt{kN}$ documents. Hence, an $O(N^2)$ method applied to this sample clearly runs in $O(kN)$ time. (Cutting et al. use Group Average Agglomerative Hierarchical Clustering (AHC) as their $O(N^2)$ method.) This AHC method is used to obtain k clusters from the random sample. The centers of these clusters are then used as seeds around which the entire corpus is clustered. Hence, in Buckshot, a complete, hierarchical clustering method is used, but it is applied to a small sample of the corpus.

In Fractionation, the entire corpus is initially partitioned into N/m buckets, each of fixed size m documents, where $m > k$. Each of the buckets is then clustered using an AHC method. The objective of the clustering is to reduce each bucket from m individual documents to mr clusters, where

the *r* is a pre-specified reduction factor ($r < 1$). Hence, after the first clustering stage, there are *mr* clusters in each of the *m* original buckets. Each of these *mr* clusters is then treated as an individual "virtual document" of size 1/*r* original documents. Since there are *N/m* buckets, there are now $mr(N/m) = Nr$ virtual documents. These virtual documents are now processed exactly like the original documents, i.e., partitioned into *Nr/m* buckets, each of which is then clustered into *mr* virtual documents at the second stage. Hence, after stage two, there are $(Nr/m)*mr = Nr^2$ virtual documents. This process continues until after *j* stages, there are $Nr^j < k$ virtual documents or clusters. One final agglomerative stage produces *k* clusters, whose centers then become the seeds for clustering the entire corpus on an individual document basis. Hence, in Fractionation (in contrast to Buckshot), the complete hierarchical clustering method is applied to the entire corpus, but the clustering performed is coarse, because the corpus is partitioned arbitrarily into buckets, and the clustering is applied separately to each bucket rather than to the corpus as a whole.

Given the *k* seeds or cluster centers (produced by either Buckshot or Fractionation), how are the *N* documents clustered? As a simple approach (similar to Rocchio), each document can be assigned to the center to which it is most similar. This process can be refined by iteration, i.e., after each document in the corpus have been clustered around one of the seeds, the centers can be recomputed, and once again, each document can be assigned to the most similar center. Several other refinements are employed by Cutting et al. First, they summarize each cluster as a *profile*. The profile of a cluster is a term vector equal to the sum of the term vectors representing the documents in the cluster. Hence, in successive iterations, instead of adding a document to the cluster with the most similar center, one can add the document to the cluster with the most similar profile vector. They also define *split* and *join* algorithms. The former 'separates poorly defined clusters into two well separated parts," and the latter "merges clusters which are too similar." In general, the choice whether to use these refinements, and how many refinements and iterations to apply, involves a trade-off between speed and accuracy. However, the entire method including refinements, runs in $O(kN)$ time.

Buckshot and Fractionation partition the corpus, i.e., they do not support overlap (although they can be modified to support overlap). [Zamir et al., SIGIR '98] Neither algorithm is incremental. Moreover, Buckshot generates its initial cluster samples from a random sample of the initial corpus. Hence, it is not deterministic, i.e., in successive runs, it may generate different clusters "although [in the author's experience] repeated trials generally produce qualitatively similar partitions." [Cutting et al., 1992] A further risk is that "when one is possibly interested in small clusters,... they may not be represented in the [random] sample." [Zamir et al., SIGIR 1998]

Buckshot and Fractionation (and the STC clustering method described in the next section) are all motivated by the desire to allow the user to browse a large document collection interactively. Such an application requires that the clustering be performed very rapidly, e.g., in seconds, even for very large collections. If the collection is available in advance, e.g., hours before the user begins browsing, then hybrid approaches to speed up the interactive clustering are possible. The essential idea is that a fixed cluster hierarchy is generated off-line. The nodes of this hierarchy then become "virtual documents" to be clustered on-line (as in Fractionation above).

Clearly, the "complete" $O(N^2)$ methods described in the preceding section can be used to generate this cluster hierarchy. However, for large collections, e.g., thousands of documents or more, $O(N^2)$

methods are far too slow even for off-line clustering. Hence, Cutting et al. [SIGIR '93] propose using an *O(kN)* clustering method such as their own Buckshot or Fractionation, even in the off-line stage. These algorithms only produce one level of cluster, i.e., they produce a "flat" partition of the collection into *k* clusters. However, the flat clustering method can be applied iteratively to partition each of the original *k* clusters into *k* sub-clusters, then to partition each of these sub-clusters yet again, and so on. The partitioning stops only when individual documents are reached. This iterative procedure generates the required cluster hierarchy. Since the clustering at each level runs *O(kN)* time, the entire hierarchy can be generated in *O(kN log N)* time. (In their reported experiment, Cutting et al. generated a cluster hierarchy for a DARPA Tipster collection of 700,000 documents, occupying 2.1 gigabytes of text, and containing over a million unique words, in forty hours on a Sun SPARCStation 10. While faster hardware would obviously do the job more quickly, it is evident that the production of such cluster hierarchies remains an off-line task.)

Given this pre-computed cluster hierarchy, Cutting performs constant-time clustering interactively, based on the assumption that a fixed number of virtual documents $M >> k$ is to be clustered. *M* is chosen such that the clustering can be performed in the desired constant time bound, regardless of the true number of actual documents in the collection. Cutting obtains the *M* virtual documents (also called *meta-documents*) by starting at the root of the cluster hierarchy, or at any node in the hierarchy designated by the user. He replaces the initial node by its children. Each child node is itself a virtual document. At each subsequent stage, he finds the child that has the most leaves and replace that child by *it*s children. He continues this process until *M* children, i.e., *M* virtual documents, have been accumulated. Then, he clusters these *M* virtual documents into *k* clusters. In his published experiment, he found that interactive clustering took approximately 20 seconds.

Note that most clustering methods (including those used by Cutting) involve computing inter-document or document-centroid similarities. These similarities typically involve a similarity function such as cosine similarity applied to documents represented as term vectors, where each term in a vector is a word or phrase. When the documents being clustered are *virtual* documents, the term vectors will be very long; the term vector for virtual document V_i will contain a non-zero value for every term in any of its descendent leaves, i.e., for any of the actual documents of which it is composed. In other words, the set of terms in a virtual document is the union of the terms in its component actual documents. To keep the similarity computations from being very slow, Cutting *truncates* the term vectors of every virtual document, retaining only the fifty highest weighted terms. (This also reduces substantially the space required to store the cluster hierarchy.) Schutze et al. [SIGIR '97] find that with such truncation, "the speed increase is significant while - surprisingly - the quality of clustering is not adversely affected."

The Cutting constant-time clustering method clusters virtual documents, where each virtual document is essentially a node from some level in the pre-computed cluster hierarchy. The clusters produced are always unions of these virtual documents. Of course, as the browsing user focuses more narrowly, on lower levels of the cluster hierarchy, the virtual documents that get clustered may be correspondingly narrow, i.e., may contain small numbers of actual documents. But to some degree, the user is limited by the original pre-computed hierarchical structure. Silverstein et al. [SIGIR '97] address this issue with their "almost-constant-time" cluster method.

Like Cutting, Silverstein assumes a pre-computed cluster hierarchy, *H*, covering a document corpus *C* of size *N*. However, whereas Cutting only allows the user to designate a node (or perhaps several nodes at some level) in *H* as the starting point for clustering, Silverstein assumes that the user has obtained (and wishes to cluster for browsing) some subset *S* of actual documents from *C*. For example, the subset *S* might be obtained by executing a query *Q* against *C* using some IR engine. Silverstein wishes to map *S* into *H* in such a way that the clusters reflect *S* but use the pre-computed *H* to speed up the clustering. He acquires *M* virtual document nodes from *H* for clustering as Cutting does. However, he departs from Cutting in two significant ways. First, in expanding the set of virtual documents to reach his goal of *M*, he uses a "goodness" test to weed out the nodes that contain the smallest proportion of documents from *S*. Specifically, when a node is replaced by its children, this goodness test is used to identify the *worst* child, the one with the smallest proportion of documents from *S*. This "bad seed" is replaced by *some* of its children. The replacement process is also a weeding out process: Specifically, child nodes that contain no documents from *S* are discarded. Each child node that contains only a single document from *S* (actually less than *c* documents from *S*, where *c* is a small constant) is replaced by a "singleton" node that contains only the document(s) from *S*. After *M* nodes have been accumulated, Silverstein clusters these *M* nodes (actually the union of the *M* nodes and the "singleton" nodes) into *k* clusters, as Cutting does. Finally, he goes through the all the clustered nodes, removing actual documents that are not in *S*.

The Cutting method is constant time (at user interaction time, not cluster hierarchy pre-computation time, of course) because it clusters a fixed number *M* of nodes, where *M* is chosen so that the clustering time is acceptable to an interactive user. The Silverstein method is *almost*-constant-time because it requires the generation of a function (implemented as a table) that identifies the documents in *S* that are contained in any given node n of H. The computation of this table takes $O(|S| \log N)$ time. Hence, computation of this table is not constant, and cannot be pre-computed because it depends on *S*, which is specified by the interactive user. However, the time to compute the table is "dwarfed, in practice by the contribution of the [constant] clustering step."

It should be stressed that both the Cutting and Silverstein methods depend on the availability in advance, i.e., off-line, of the collection to be clustered (or in the case of Silverstein, the collection from which a subset is to be clustered). Moreover, this collection must be available well in advance of user interaction, because the required pre-computation of a cluster hierarchy is a time-consuming process. By contrast, the STC method, described in the next section, achieves comparable or better run-time speed without any pre-computation. It is genuinely *incremental*.

Heuristic algorithms fail van Rijsbergen's three criteria for theoretical soundness, but have been found "about as effective as those based on hierarchical agglomerative methodologies [e.g., single-link and complete-link]" in "many retrieval settings." [Salton, 1989]

10.3 Incremental Cluster Generation

Incremental methods also make use of a similarity measure but they don't require that similarities be pre-computed for all document pairs. Indeed, all document pairs are not available initially, since by definition, incremental methods cluster a stream of incoming documents. The similarities are computed "on the fly" as the documents stream past the incremental cluster system (or what

comes to the same thing, as the incremental system makes a "pass" through the document collection). All incremental cluster methods may be said to make a single pass through the documents, in the sense that as the *i*-th document is accessed and processed, the result is the best set of clusters the method can produce with *i* documents. By the time the *N*-th document, i.e., the "final" document, is processed, the entire collection has been clustered and the method is "done." Note that it doesn't make any difference to an incremental method whether the collection of *N* documents is available initially, or whether it is a stream of *N* documents arriving in sequence, e.g., *N* documents that have been retrieved from the Web by an IR engine. In either case, an incremental method processes the *i*-th document as if it only has knowledge about the first *i* documents, and hasn't yet encountered documents *i*+1 to *N*.

One can distinguish between methods that are purely incremental, and methods that can run in either an incremental or non-incremental mode. For example, the single-link method discussed in the preceding section is classically executed in a static mode, i.e., it starts with *N* documents. Algorithms exist for clustering the *N* documents according to the single-link method. Algorithms also exist for adding an additional (*N*+1)th document, without starting over from scratch. Such *update* procedures produce the same effect *as if* one had started from scratch. One can see that such algorithms must exist for the single-link method by considering the effect they must produce in terms of the single-link clustering rule. The *N* documents have been grouped into a hierarchy of clusters according to the single-link rule. Document D_{N+1} must be linked to the document D_i ($i \leq N$) with which it is most similar. The effect is to link it into every cluster in the hierarchy that contains D_i. If D_{N+1} is equally similar to several "closest" documents D_i, Dj, and D_k (which need not be very similar to each other, and hence may be in different clusters up to a very high level in the hierarchy), then the clusters to which they belong will be linked together into a single cluster (if they were not already so linked). It is plain that the cluster hierarchy formed by adding document D_{N+1} continues to obey the single-link clustering rule.

However, if an update procedure exists, then why not use it starting with document D_2 (the 2nd document encountered), and throw away any non-incremental methods that work on the first *N* documents, where $N >> 2$? If the non-incremental procedure is retained, then it must be because it is more efficient (in documents clustered per unit of time) than the update procedure, or because it produces a data structure better-suited to efficient document or cluster retrieval.

On the other hand, if the update procedure is the *only* procedure used in the clustering, then the implementation is purely incremental. If the update procedure is the cluster-defining rule, then the cluster *method* is incremental.

All incremental methods make a single pass through the documents. However, they are not all single-pass methods. The essential distinction is whether the method, on encountering the *i*-th document, revisits earlier documents $j < i$, or existing clusters, with a view to possible re-clustering. If it does, it is not a "pure" single pass method, since it may visit documents more than once, in the process of making its "single" pass. Actually, virtually all incremental methods do *some* revisiting, but a method that does no reclustering, and minimizes document revisiting is "more" single-pass than a method that revisits and re-clusters extensively. In general, single-pass methods are order-dependent, i.e., the clusters formed depend on the order in which documents are processed. The reason is evident; there is no opportunity to revise clusters formed from early documents on

the basis of information in documents that are processed later. This violates one of van Rijsbergen's three criteria for theoretical soundness.

A nearly single-pass method works as follows: If a new document is similar enough (according to some similarity measure and threshold) to one of the preceding documents, they are combined into a cluster. Similarly, if a new document is similar enough to a cluster formed from two or more of the preceding documents, it is added to that cluster. A document is added to all clusters to which it is similar enough. (To simplify the process of computing the similarity of a document and a cluster, a centroid is computed for each cluster. Each new document is then compared to the centroid of each existing cluster. If a document is added to a cluster, its centroid is recomputed. Of course, recomputation of the centroid of a cluster involves revisiting the documents that are members of the given cluster, so even this simple algorithm involves *some* revisiting.) When the single pass is completed, a number of clusters have been formed. The results may depend on the order in which the documents were examined. This process may result in very large clusters, or clusters with a large amount of overlap. Hence, one usually specifies such parameters as minimum and maximum cluster size, and maximum overlap, etc. The clusters formed in the original pass through the data are then adjusted, e.g., by cluster splitting and merging, to conform to these parameters.

Zamir et al. [SIGIR 98] have developed a novel incremental clustering method, called Suffix Tree Clustering (STC). The STC method is motivated by a problem that arises frequently when querying the Web with an IR engine: A huge ranked list of documents is retrieved, of which only a very small number are relevant to the user's query. To make matters worse, the relevant documents are often far down the list of documents returned. In IR terminology, precision is often very low. Zamir et al. propose to alleviate this problem by clustering the documents returned to the user, and labeling each cluster with phrases that characterize (one hopes) its common topic(s). The user then browses through the clusters, picking out and ranking the clusters that appear most relevant to her query on the basis of their labels. She can then begin examining documents in the most relevant cluster. If she needs or desires more information than she finds in the first cluster, she can go on to the next most relevant cluster (as determined by its labels and the user's judgment), and so on. In this way, the number of documents that she must sift through to find relevant data can be reduced by at least an order of magnitude. (See the section on User Interaction for further discussion of browsing.)

Note that STC, although motivated by the difficulties of Web retrieval, is applicable in any case where the number of documents *N* retrieved, is large, and the precision is low. It is particularly applicable if the user is doing her retrieval interactively, and wants to see her output very quickly.

STC has a number of very noteworthy features. Some of these features are found in other clustering methods, but no other method (to the author's knowledge) combines them all. (1) STC is a *linear time* method, i.e., the time required to cluster *N* documents rises only linearly with *N*. This is an essential requirement whenever *N* is very large, as it is in many practical applications. (There are constant-time and almost-constant-time clustering methods (see previous section), but they depend on off-line pre-clustering.) (2) STC is *incremental*, which means that it can begin clustering documents as soon as the first document arrives; it doesn't have to wait until all the retrieved documents have arrived before it begins clustering, as non-incremental methods do. (By contrast,

the *complete* clustering methods discussed in an earlier section are non-incremental and non-linear time. The *heuristic* clustering methods discussed in the preceding section are either linear time, but non-incremental, or near constant time, but require substantial pre-processing, and hence are also non-incremental.) (3) The STC method is *non-heuristic* in the sense that the clusters it produces are independent of the order in which the documents it clusters are initially accessed. (4) The STC method does *not r*equire pre-specification of either the number of clusters to be generated, or the maximum or minimum size of the resulting clusters. Heuristic methods commonly have such halting or clean-up criteria; even AHC algorithms are often used in conjunction with such criteria. STC only requires pre-specification of two parameters, a cluster overlap measure and the number of "best clusters" to be reclustered (see below). But Zamir et al. find that STC is not very sensitive to the value of the overlap parameter. (5) The STC method permits a document to be placed in more than one cluster (a characteristic that is termed *cluster overlap*). This is a very important consideration when the objective is to cluster documents by topic, since a given document may be about multiple topics.

(6) A feature that sets STC apart from practically all other clustering methods is that it uses ordered strings of words, which it calls "phrases," as its document descriptors. (Most other methods of text document clustering use unordered sets of words.) Moreover, STC uses the presence of a phrase in two documents as its inter-document similarity measure for clustering purposes. The presence of a shared phrase is also the STC primary (first stage) cluster method rule. (In sharp contrast, all of the other clustering methods described above are totally indifferent to how similarity of text documents is defined, or indeed even to the fact that the objects being clustered *are* text documents.) In other words, if D_1 and D_2 share at least one common phrase (as defined below), they are combined into a "base" cluster. Hence, instead of an N^2 matrix of inter-document similarities, STC employs a structure called a "suffix tree," that indexes a document collection by the phrases of which its documents are composed. The index itself grows linearly with the size of the text, and can be updated and accessed in linear time. Experiments by Zamir et al. indicate that the use of ordered strings as document descriptors is critical to the success of the STC method.

STC employs a second stage of clustering, using a different clustering rule. The second stage rule combines the clusters produced by the first stage according to the proportion of documents that they share (see below). In other words, cluster overlap is permitted, but if the percentage of overlap is very high (Zamir et al. use a parameter of 50%), the clusters are combined into a larger cluster. Note that STC is not hierarchical; only two stages of clustering are performed. Hence, no root cluster is generated, and no halting criterion is required. STC is also not iterative (like Rocchio's method). Rather, it is incremental. The two clustering stages are performed every time a new document arrives or is accessed.

STC indexes the collection of documents as a "suffix tree," hence the name of the method. In most past applications, a body of text has been viewed as an ordered string of characters. The suffix tree has been employed as an efficient representation of the string for such purposes as finding (in linear time): a given word *w* in the text, the first occurrence of *w* in the text, the number of occurrences of *w* in the text, etc. [Crochemore et al., 1994] Each path in the tree from root to a leaf node represents a "suffix" of the text. Each internal node of the tree represents a "prefix" shared by two or more suffixes. In STC, the text is viewed as an ordered string of *words*. The application is to cluster documents that share one or more word strings. (Zamir et al. call these strings "phrases,"

but no syntactic structure is implied.) A suffix tree is constructed to represent and index all the sentences in a collection of documents. Each sentence SE_i is treated as an ordered string of n words, w_{i1}, w_{i2},..., w_{in}. Every sub-string w_{ij},...w_{in} for $1 <= j <= n$, is a suffix of SE_i. In other words, there are n suffixes for every sentence of length n words, the string beginning at word 1, the string beginning at word 2, etc. (Technically, there are n+1 suffixes, because the set includes the "empty" suffix, located at the root of the tree.) A suffix tree is constructed for the first sentence of document 1, updated with the suffixes of the second sentence, and so on. As each new document arrives and is added to the growing collection, the suffix tree is updated to reflect all the new suffixes in its sentences that were not encountered in any previous document, and all the re-occurrences of suffixes that were previously encountered in one or more earlier documents. As each new sentence SE_i of the current document D_j is processed, the suffix tree is updated to reflect all the new suffixes in SE_i that were not encountered in any previous document or any previous sentence of D_j, and all the re-occurrences of suffixes that were previously encountered in one or more earlier documents. Each distinct suffix becomes the path, the series of edges, leading to a leaf node of the suffix tree. Each leaf node is labeled with the suffix whose path leads to that node, and is indexed by all the documents in which the given suffix appears. Similarly, the path to each internal (non-leaf) node represents a *prefix*, a string of words that begins two or more suffixes. The internal node is labeled with its prefix, and indexed to identify all the documents in which its suffixes occur.

For example, document D_1 may contain the suffix "The quick brown fox jumped over the lazy dog." Call this suffix S_1. (Actually, there will be n suffixes for this sentence, corresponding to: "dog," "lazy dog," "the lazy dog," etc., up to and including the suffix representing the complete sentence.) Each non-leaf (internal) node represents the first n_1 words, the prefix, of two or more suffixes. There will be n-1 i*mplicit* prefixes for this sentence: "The," "The quick," "The quick brown," "The quick brown fox," etc. However, these prefixes remain implicit, until one of them is encountered in a subsequent sentence of D_1 or a subsequent document D_2, with a different continuation. Each internal node must have at least two children, each child representing a different continuation of the parent. Document D_2 may contain a sentence with the suffix, S_2, "The quick brown fox leaped over the sleeping collie." When D_2 is encountered, the implicit prefix node "The quick brown fox" becomes an explicit node P_a. A third document D_3 may arrive containing a sentence with the suffix S_3, "The quick are livelier than the dead." In that case, an internal explicit node is created representing the prefix, P_b, "The quick." This node will have two children, one the leaf node for S_3, the other the internal node for prefix P_a (a continuation of P_b), branching in turn to the two children representing the suffixes S_1 and S_2. In this way, all of the word sequences representing (and characterizing) a collection of documents can be efficiently represented, and easily updated as new documents arrive, containing new suffixes, and new instances of existing suffixes. Each internal node of the suffix tree represents a prefix, and the set of one or more documents containing that prefix, e.g., in the above example there is an internal node representing the prefix, P_b, and the documents, D_1, D_2, and D_3, containing it. Each leaf node represents a suffix, and the set of documents containing it, e.g., there is a leaf node representing S_3, and the document D_3 containing it. The set of documents associated with a node of the suffix tree is called a "base cluster," and the phrase that is common to those documents is its label. D_1, D_2, and D_3 comprise a base cluster whose label is P_b. Note that a phrase can be either a prefix or a suffix, or even both, i.e., it can be a suffix of a sentence in one document, and a prefix of a sentence in another document.

Note that, for purposes of clustering, we are only interested in phrases (prefixes or suffixes) that occur in two or more documents. However, we create a node for every suffix of a sentence in a given document D_1, and for every prefix that occurs in two or more sentences of D_1. That is because we don't know in advance whether a given prefix or suffix in D_1 will be encountered again in a later document. However, we only form base clusters for phrases that occur in at least two documents.

Any sequence of words in a sentence of the collection is a phrase in STC. The sequence need not be a phrase in any syntactic sense, though syntactic phrases will be included (provided they are composed of contiguous words). Also observe that a given document may (and normally will) contain multiple phrases, e.g., D_1 contains P_b, P_a, and S_1. Some of these phrases will be unique to the given document, others will be shared with other documents. I*t* should also be noted that two documents may share more than one phrase, e.g., D_1 and D_2 share both P_a and P_b. Hence, the base cluster with label P_a and the base cluster with label P_b overlap, i.e., share D_1 and D_2. On the other hand, the two base clusters do *not* coincide; the base cluster for P_b also contains D_3, a document that is not in the base cluster for P_a.

The construction of the suffix tree not only indexes the document collection, linking each phrase (including suffixes) to the documents that contain it; it also performs the first stage of clustering, the identification of the base clusters. The second stage of clustering combines base clusters that are "similar." Base cluster similarity is defined as having a high degree of document overlap. In other words, base cluster C_1 (defined by containing the phrase P_1) and base cluster C_2 (defined by containing the phrase P_2) are said to be similar if a high proportion of the documents in these two clusters contain *both* P_1 *and* P_2. Specifically,

$$|C_1 \cap C_2| / |C_1| > 0.5$$
$$|C_1 \cap C_2| / |C_2| > 0.5$$

where 0.5 is the overlap threshold chosen by Zamir et al.

The cluster C_{12} formed by combining C_1 and C_2 contains the union of the documents in C_1 and C_2 (and is labeled with both P_1 and P_2). Similarly, C_2 may be combined with C_3 into C_{23} on the basis that a high proportion of the documents in these clusters contain both P_2 and P_3. C_{12} and C_{23} are then combined into C_{123} on the basis of the common linking cluster C_2. The result is a form of single-link clustering with C_1 linked to C_2 which is linked to C_3. However, the undesirable chaining effect in which the end points of the chain are quite dissimilar is much less pronounced in the STC case because the chaining is taking place at the level of base clusters rather than individual documents. The composite clusters that emerge from stage two are labeled with the set of phrases that label their component base clusters.

Each new document that arrives causes the suffix tree to be updated. If document D_i contains a suffix S_j that did not occur in any of the preceding documents D_1 to D_{i-1}, then a new leaf node is added to the suffix tree with value S_j (and maybe a new internal node as well, if the new suffix is a new continuation of an existing prefix P_{old} that was previously only the beginning of one suffix or suffixes in only one document, D_h, $h< i$, but is now the prefix of suffixes in two documents and hence rates its own internal node.) Corresponding to the new leaf node, a new base cluster is created, containing initially just the one document D_i, and labeled with the new suffix, S_j. (If a new

internal node is created too, a new base cluster will be defined for *that* node, containing D_i and D_h, and labeled with the shared prefix, P_{old}.) If D_i contains a phrase P_k (prefix or suffix) that is already in the suffix tree, i.e., that occurs as the value of a node in one or more of the first *i*-1 documents, then D_i must be added to the base cluster with label P_k. Note that this may have the effect of either increasing or decreasing the similarity of base cluster C_k to other base clusters. If D_i also contains the phrase P_1, it increases the overlap of C_k with C_1 (or creates such an overlap if it did not exist previously). If D_i does not contain phrase P_1, it decreases the overlap (if any). Hence, D_i may push the overlap between C_1 and C_k over the user-defined threshold, causing clusters C_1 and C_k to be combined if they were not combined previously. Similarly, D_i may push the overlap between C_1 and C_k below the threshold, causing combined cluster C_{1k} to be split apart into its component base clusters C_1 and C_k. Hence, each new document may cause re-clustering of existing clusters. (STC is an incremental clustering method, but definitely not a pure one-pass method, as that term has been defined here.)

The potential re-clustering when a new document D_i arrives requires that inter-cluster similarity be recomputed for all affected base clusters, and for all clusters composed of affected base clusters. The number of such clusters goes up as each new document arrives. Hence, the number of required similarity computations can rise dramatically as the population of documents grows. Since the objective is to support real-time clustering (remember that the original motivation was to support browsing of documents retrieved by an interactive user from the Web!), the number of similarity computations is held constant by only comparing updated base clusters with the *q* "best" base clusters, where *q* is a parameter; *q*=500 in the research reported here. The term "best" does *not* refer to relevance to the user's query; that is determined interactively by the user after all the retrieved documents have been clustered. Instead, it refers to a measure of a cluster's merit *as* a cluster.

Base clusters are ranked by score *s(B)*, computed as:

$$s(B) = |B| \cdot f(|P|)$$

where $|B|$ is the number of documents in base cluster *B*, $|P|$ is the effective length of phrase *P*, and *f(P)* is a non-linear function of phrase length. The function *f(P)* "penalizes single word phrases, is linear for phrases that are two to six words long, and becomes constant for longer phrases." (Note that, strictly speaking, the "break points," two and six, of *f(P)* are also parameters of the STC method. Clearly, other values could be chosen, e.g., the phrase length score could be allowed to increase linearly up to a length of eight. Zamir et al. do not discuss how they arrived at these values, or any study of the effect of altering these values.) The length of a phrase is the total number of words, excluding stop words; for Web applications, the usual stoplist is expanded to include common Internet words like "java" and "mail." Hence, a phrase consisting of a single stoplist word has a length of zero. The value of *f(0)* is zero, so base clusters defined by sharing a single stopword are discarded. Note that although stop words do not contribute to the length of phrases in which they occur, and hence do not contribute to the ranking of the corresponding base cluster scores, they *can* serve to distinguish phrases. In the above example, if document D_4 arrives ending with the suffix, "The slow brown fox can't jump over anything," the phrase "The quick" becomes two distinct phrases, "The quick" and "quick," because the phrase, "The," now becomes a separate node with two distinct children, "quick" and one beginning with "slow." The base cluster corresponding to "The" is discarded because its score is zero.

After the *N*-th (final) retrieved document has been processed, all *N* documents have been clustered; the STC clustering process is completed. The user can then browse these clusters, judging on the basis of their labels which are most likely to contain relevant documents. (Zamir hypothesizes that the phrases that label a cluster will prove effective descriptors of that cluster's content for effective human browsing, but this belief had not yet been tested in the reported research.) When she finds the most promising cluster, the user can "drill down" and look at titles or whatever other "snippets" the Web engine has returned. When she finds an interesting "snippet," she can drill further to look at the full text of the corresponding page. Zamir assumes that even at the cluster level, the number of entities generated by STC will be greater than the user can comfortably browse. So, he ranks the final set of clusters, assigning each cluster a score "based on the scores of its base clusters, and their overlap." Hence, the user only has to (is allowed to?) browse the *p* best clusters. Again, "best" is a measure of cluster quality, e.g., number, size and overlap of its component base clusters (and hence of its coherence), length of the phrases that label it (longer phrases are likely to be more descriptive), etc. Cluster relevance is determined interactively by the browsing human user. The human browser sees the number of documents in each cluster, and the phrases of its base clusters.

The STC method appears to achieve the quality of a "complete," i.e., $O(N^2)$ method (see discussion of Cluster Validation below), while running in linear time, i.e., $O(N)$. The "secret" is the nature of the "similarity" measure STC uses, and the efficient data structure and algorithm STC uses to index the documents and compute the similarity. Practically all other cluster methods use a measure such that if document D_1 is similar to document D_2, and document D_2 is similar to D_3, one cannot assume that D_1 is similar to D_3. In a word, these measures, e.g., cosine similarity, are non-transitive. As a result, every pair of interdocument similarities needs to be computed and accessed for "completeness." By contrast, STC forms its base clusters on the basis of shared phrases. If D_1 and D_2 share a phrase, and D_2 and D_3 share the same phrase, then D_1 and D_3 certainly share that phrase too! Hence, STC can perform complete clustering at the base cluster level without incurring the $O(N^2)$ penalty. STC achieves $O(N)$ time and space by employing a suffix tree to index the document collection, and an efficient algorithm due to Ukkonen [Algorith, 1995] [Nelson, 1996] to build and update the suffix tree. The second-stage clustering of base clusters is not transitive, but involves clustering of base clusters, not documents. Moreover (and this is the most "heuristic" element of the method), during the incremental re-clustering of base clusters, only the *q* "best" existing clusters are revisited, as noted above. This keeps the time (actually the maximum time) required for stage two constant as the number of documents grows.

Finally, it should be noted that because it is incremental, the performance of STC in the application domain for which it was developed, e.g., clustering of documents retrieved from the Web, may be much better than linear. STC can cluster the documents as they are arriving. Hence, by the time the last (*N*th) document arrives, STC's clustering may be nearly completed. In a reported experiment, cluster results are returned "to the user a mere 0.01 seconds after the last document is received by" the Web retrieval engine. However, it should be noted that clustering in this case was performed on the "snippets" extracted from Web pages by the IR engine, not the full text of the actual pages. This is compared by Zamir to the truncation of document vectors by Cutting et al. {SIGIR '93] and Schutze et al. [SIGIR '97] described earlier

10.4 Cluster Validation

Since even randomly generated data can be clustered, it is important to determine whether the clusters produced when a given clustering method is applied to a given collection, are meaningful. It is even more important to determine whether the clusters produced contribute to *effective* information retrieval. In other words, are the clusters produced likely to satisfy the cluster hypothesis?. If a query or browsing method locates and retrieves a cluster of appropriate size, is it likely that many or most of the documents in that cluster will be relevant to the query, or of interest to the browsing user? If the user relaxes the cluster threshold, retrieving documents that were close to the boundary of the original cluster, are these new documents likely to be at least partially relevant to the user's need?

Several approaches to cluster evaluation with specific applicability to document retrieval have been tried. These approaches try to determine whether a given collection is a good candidate for clustering, i.e., whether clustering will promote retrieval effectiveness. One approach, due to van Rijsbergen and his associates [van Rijsbergen et al., 1973] is to compare the average interdocument similarity among relevant documents to the average similarity among relevant-nonrelevant document pairs. This average can be computed for a given query or over a set of queries. If the cluster hypothesis holds, the average similarity among relevant documents should be substantially larger than the average over relevant-nonrelevant pairs. A second approach, due to Voorhees, is to determine for each document relevant to a given query how many of its nearest neighbors are also relevant to the query. In her experiments, Voorhees [TR 85-658] considered the five nearest neighbors to each relevant document. These two methods both require that a query or set of queries be applied to the collection and that relevance judgments be applied to the documents retrieved by these queries. The assumption is made that the results for the given queries characterize the given collection in the sense that other queries applied to the collection will give similar results. A third approach, due to El-Hamdouchi and Willett [JIS, 1987] depends entirely on properties of the collection itself, or more precisely on the terms that index the documents in the collection. They calculate a *term density*, defined as the number of occurrences of all index terms in the collection (the number of *postings*) divided by the product of the number of documents in the collection and the number of unique index terms. This density is a measure of how densely populated the term-document matrix is. The theory is that the greater the term density, the more frequently documents will share terms, and hence the better a clustering can represent degrees of similarity between documents. In a reported comparison of these methods, the term density measure correlated best with effectiveness of cluster searching. [Willetts, IP&M, 1988]

As the size of distributed collections and the corresponding size of retrieval sets grow, the application of clustering to user interactive browsing also grows in importance. A number of the clustering methods described above, are specifically aimed at this application domain. Hence, some evaluations of effective clustering have also been aimed at this application. Browsing experiments have been conducted to evaluate the effectiveness of clustering for this purpose. These experiments are discussed further in the section below on User Interaction. However, one test of retrieval effectiveness [Zamir et al., SIGIR '98] that simulates browsing will be discussed here. This test compared the STC clustering method (described in the preceding section) against several heuristic clustering methods (described in the section on Heuristic Methods) and $O(N^2)$ methods (discussed in the section on Complete Methods). Specifically, it compared STC against four lin-

ear-time heuristic methods: Single-Pass, K-means (this is the Rocchio method), Buckshot, and Fractionation), and one $O(N^2)$ method: Group-Average Hierarchical Clustering (GAHC).

The strategy adopted by Zamir et al. is based on results reported by researchers who conducted actual browsing experiments. These experiments indicate that a user is usually (about 80% of the time) able to select the cluster containing the highest proportion documents relevant to her need, on the basis of the cluster labels or summaries provided to her. Hence, Zamir generated 10 queries, retrieved documents from the Web for each of those queries, and then manually generated human relevance judgments for each of the 10 retrieval sets, relative to the query for which it was retrieved. Then, they clustered each of the retrieval sets using each of the cluster methods, setting parameters as appropriate so that 10 clusters were generated for each retrieval set/cluster method pair. Then, for each retrieval set and cluster method, they automatically selected the "best" cluster, i.e., the cluster containing the highest proportion of relevant documents, then the next best, and so on, until they had selected clusters containing 10% of the documents in the "collection," i.e., in the given retrieval set. This was based on the assumption, noted above and borne out to some extent in practice, that users can select the best clusters on the basis of their labels or summaries. In all cases, the cutoff was 10% of the documents in the given set; this meant that the cutoff might occur in the middle of a cluster, even in the middle of the first cluster, if that cluster was large for a given cluster method. The resulting 10% documents were then ranked, and the average precision computed, averaged over all 10 collections. (Since STC supports document overlap, a given document might appear in two or more selected clusters. For purposes of ranking, such duplicates were discarded.) Equalizing the number of clusters generated, and the number of documents ranked, across methods and collections, allowed for a fair comparison of cluster methods. Note that Zamir ranked documents by cluster, i.e., the documents in the best cluster were ranked higher than the documents in the next best cluster, and so on. Zamir et al. do not specify how they ranked documents within a given cluster. However, Hearst et al. rank documents within a cluster using two different criteria: closeness to the cluster centroid, and similarity to the original query.

The results reported by Zamir et al. show that STC out-performed the other methods, even the GAHC method, by a significant margin. GAHC out-performed the other methods, which is not surprising considering that it is an $O(N^2)$ method (and consequently far too slow for interactive use, and even for off-line use of large collections). It is striking that STC, an $O(N)$ and incremental method, out-performed GAHC, which is neither! Zamir et al. concede that their results are preliminary; indeed, the title of their paper refers to the reported study as a "feasibility demonstration." The results are preliminary for (at least) four reasons. First, the results were obtained from non-standard, e.g., non-TREC, collections, retrieved from the Web by 10 arbitrary queries. (This was deliberate, since Web retrieval is the intended application of STC.) Second, the resulting collections were (relatively) small, e.g., 200 documents each. (On average, there were about forty relevant documents for each query.) Third, the relevance judgments were generated by the researchers rather than by independent judges, as in TREC. Fourth, the study did not use actual human users, performing actual interactive browsing. However, their system, MetaCrawler STC, has been fielded on the Web, so that statistics can be gathered from actual users. The data set employed is also being published on the Web, so that other researchers can replicate and validate these experiments.

11 Query Expansion and Refinement

A query or information need submitted by an end-user is ordinarily a short statement or an even shorter list of terms. This is only to be expected. The normal user is not necessarily an expert on all the term usages in a large collection of documents he wishes to query. Nor does he want to spend his time consulting thesauri and other reference works in an attempt to generate an ideal query. A sophisticated user may in fact do some of these things. But the approach taken in both some commercial IR engines and much IR research is to refine and expand the original query automatically based on the documents retrieved by the original query. {Salton et al., JASIS, 1990] Query refinement and expansion may involve adding additional terms, removing "poor" terms, and refining the weights assigned to the query terms. It is possible to recompute the weights without expanding the query, or to expand the query without recomputing the weights, but experiment indicates that both expansion and re-weighting improve retrieval performance. [Harman, SIGIR '92] The process of query expansion and re-weighting can be applied to either vector space queries or extended boolean queries. [Salton, ATP, 1989] [Salton, IP&M, 1988] The process may be wholly automatic or may involve a combination of automatic processes and user interaction.

11.1 Query Expansion (Addition of Terms)

A number of approaches to automatic query expansion have been tried. A common approach is to expand the query with terms drawn from the most relevant documents, i.e., the documents that the user judges relevant among those that were retrieved when her original query was applied to the collection. This process is called "relevance feedback." [Salton et al., JASIS, 1990], [Harman, SIGIR '92] The process is interactive to the extent that the user must look at the documents most highly ranked by the retrieval system and tell the system which ones are relevant. Note that since modern IR systems typically rank all the documents in the collection, the user must decide how far down the ranking she wants to go, e.g., she must decide to read the highest ranking *X* documents where *X* is a parameter of the retrieval procedure. In effect, she makes the working assumption that the first *X* documents are likely to be relevant and hence are worth examining initially to judge their relevance. (But see below for further refinements.) However, the system can "take it from there," extracting terms from the relevant documents and adding them to the original query. The system can also reweight terms in the original query, e.g., increasing the weights of terms that appear in the documents judged relevant, and reducing the weights of terms that do not appear in the relevant documents. If there is a training set of documents such that human judges have already identified relevant and non-relevant documents for this set (relative to the given query or topic), then terms occurring in the relevant documents of this training set can be added to the initial query. (Training sets are normal for routing or classification applications.) The process of relevance feedback is iterative. Each time the query is expanded and re-weighted, this "improved" query, also called the "feedback query" [Buckley et al., SIGIR '94], is executed. The user then makes relevance judgments of the top *N* documents retrieved by the feedback query. "The process can continue to iterate until the user's information need is satisfied." Harman [SIGIR '92] recommends repeating the process as long as each iteration retrieves relevant documents not retrieved in the previous iteration. (Note: In the "residual collection" method of evaluation of relevance feedback [Salton and Buckley, JASIS, 1990] [Haines and Croft, SIGIR '93] [Chang et al., SMART, 1971], all documents previously seen and judged by the user are factored out of evaluation of the

next iteration so that each iteration is only "credited" with additional relevant documents not previously retrieved.)

A refinement of the above procedure is to let the IR system *assume* that the top *X* documents as ranked by the IR system are relevant. The system automatically expands the query using these *X* documents, runs the expanded query and returns the ranking produced by this second-stage query as the first result actually seen by the user. Note that for the purposes of automatic query expansion (the first stage of this two-stage process), precision is more important than recall, i.e., it is essential that as many of the high-ranking *X* documents as possible are relevant even if some relevant documents are overlooked. Hence, similarity measures may be employed in this automatic first stage that emphasize precision and sacrifice some recall. [Cornell, TREC 5]

When terms from relevant documents are to be added to the query, two questions are important: (1) How many documents should the user judge for relevance, e.g., the top 10 (a screenful), the top 30, the top 200? Or should the threshold be determined by similarity or probability value, e.g., all documents above a given similarity value? In any case, once a threshold is set, all documents above that threshold become the "retrieval set." The user judges each document in the retrieval set as relevant or non-relevant. (2) What should be the measure of whether a term that occurs in documents judged relevant is "important enough" to be added to the query? A common answer to the latter question is to add the term vectors of all documents judged relevant to the query. In other words, all terms in the relevant documents (after stop-word removal and stemming) are added to the query. However, if a relevant document is very large, adding all the terms in its term vector to the original query can produce a very large expanded query. This can degrade response time because "efficient large-scale retrieval systems have response times that are heavily dependent on the number of query terms rather than the size of the collection." [Harman, SIGIR '92]

A refinement to the term selection procedure is to take all the terms from the term vectors of the relevant retrieved documents, sort them according to some criterion of importance, and then add the top *N* terms from this sorted list to the query. An effective key for sorting in one experiment [Harman, SIGIR '92] was found to be "*noise*frequency*" where *noise* is a global distribution term similar to *idf*, and *frequency* is the log of the total number of occurrences of the given term within the set of relevant retrieved documents. In other words, preference is given to terms that occur frequently in the documents judged relevant but that do not occur frequently in the collection as a whole. The same experiment found that adding the top 20 terms produced better retrieval performance (measured as average precision) than adding a much larger number of terms, e.g., all of the non-stop words in the retrieved relevant documents. Adding all the terms dilutes "the effect of the 'important' terms … causing many non-relevant documents to move up in rank." (Of course, if a given query goes through repeated iterations, 20 terms will be added at each iteration.) This experiment simulated an interactive, ad hoc query environment in which the user issues a query, judges the top *N* (*N* = 10) documents for relevance, and the system then automatically expands the query using the "best" terms taken from the relevant retrieved documents. The user repeats this relevance feedback process with each iteration using the query expansion produced by the previous iteration.

Buckley et al.[TREC 2] [TREC 3] achieved substantial improvement in a routing environment by massive query expansion, e.g., each query was expanded by 200-300 terms before retrieval

improvement reached a point of diminishing returns. Subsequently [SIGIR ‘94], they reported improved performance for expansion up to 500 terms. However, Singhal et al. [TREC 4] found that with their improved term weighting scheme (*Lnu* — see section 3.3.2), the maximum improvement occurs at 80-100 added terms. (Traditional term weighting favors shorter documents. Massive query expansion compensates for this bias by favoring longer documents. The new weighting scheme, by eliminating the document length bias, reduces the need for the compensating bias of massive expansion.)

The above approaches distinguish “important” terms, i.e., terms that are effective at discriminating documents about a given topic, from terms that are poor at such discrimination. However, as noted earlier, a large document may deal with a number of topics. Hence, it is quite possible that inappropriate terms will be added to the query, drawn from non-relevant sections of relevant documents. These terms may be good topic discriminators but they may discriminate the wrong topic. One approach to this problem, discussed earlier (see section on query-document similarity), is to break each large document into sections, commonly called “passages,” and treat each passage as a “document.” {Allan, SIGIR ‘95] In other words, the system computes the similarity between each passage and the user’s query. This enables the system to determine the “best” passage, i.e., the passage in a given document most similar (and hence one hopes most relevant) to the query. The query is then expanded only with terms taken from the “best” passage of each relevant document.

An alternative method of query expansion (mentioned earlier) is to expand each term in the original query with synonyms or related terms drawn from a generic on-line thesaurus, e.g., Princeton’s public domain WordNet [Miller, IJL, 1990] or a thesaurus developed for a particular application domain. [van Rijsbergen, 1979] A thesaurus may be an independently generated reference work, e.g., WordNet, or it may be generated from the target collection based on term co-occurrence or adjacency, e.g., INQUERY’s PhraseFinder. [Callan et al, IP&M, 1995] A term can be expanded with synonyms or replaced by a more general word representing the class to which the original term belongs. [van Rijsbergen, 1979] The use of a thesaurus to expand a set of terms automatically into a boolean expression was discussed briefly in the section above on boolean queries. Query expansion by thesaurus may be truly automatic since unlike expansion based on relevance feedback, no feedback from the user is required. On the other hand, a user may expand a query manually using a thesaurus. Of course, expansion by thesaurus, and expansion with terms drawn from the relevant document set of the original query, are not mutually exclusive.

In a third approach to query expansion, interactive query expansion, the user is supplied with a list of candidate expansion terms derived from the relevant documents and ordered by some criterion of importance such as the “noise*frequency” discussed above. The user then chooses terms from this list to add to the query. [Efthimiadis, IP&M, 1995].

Whether terms for query expansion are selected interactively or automatically, a term ordering (also called “ranking” or “selection”) method is required. Efthimiadis, Harman [SIGIR ‘92], and Haines and Croft [SIGIR ‘93], have all studied term ordering methods. In general, the experiments conducted by these authors found that multiple ordering methods had comparable performance. They also found that the results depended on the collection to which it was applied, e.g., Haines and Croft found that relevance feedback worked much better on a collection of abstracts than on a collection of full-text documents. They found that a good term ordering method was

*rdf*idf* where *rdf* is the number of documents judged relevant by the user that contain the given term. Harman and Efthimiadis both found that a probabilistic ordering method comparable to (in Harman, slightly better than) "noise*frequency" was the BI weighting formula (see section above on probability):

$$w_k = \log \frac{p_k \cdot (1 - u_k)}{u_k \cdot (1 - u_k)}$$

where p_k is the probability that the term t_k appears in a document given that the document is relevant, and u_k is the probability that t_k appears in a document given that the document is non-relevant. Harman and Efthimiadis both got good results with $w_k(p_k - u_k)$. Efthimiadis got comparable results with his "r-lohi" method which consists of ranking terms by *rtf* (the number of occurrences of the given term in relevant documents) and breaking ties according to their term frequency (over all documents) from low frequency to high frequency. Buckley et al. [SIGIR '94] use *rdf*, breaking ties by choosing the term with the highest average weight in the set of relevant documents.

Local Context Analysis (LCA) [Xu et al., SIGIR '96] employs a more elaborate scheme for automatic choosing, ranking, and weighting of query expansion terms. LCA "combines global analysis and local feedback." It is based on fixed length passages (300 words in the reported experiments). It is also based on "concepts" where a concept is a noun group (phrase), defined as "either a single noun, two adjacent nouns, or three adjacent nouns." Given a query *Q*, a standard IR system (INQUERY in the reported experiments) retrieves the top n passages in the collection being queried, i.e., the *n* passages in the entire collection that match *Q* most closely. Concepts in the top n passages are ranked according to the formula:

$$bel(Q, c) = \prod_{t_i \varepsilon Q} (\delta + \log(af(c, t_i)) idf_c / \log(n))^{idf_i}$$

where *c* is the concept being ranked relative to query *Q*, and *bel* (which stands for "belief") is the ranking function. The heart of this ranking function is:

$$af(c, t_i) = \sum_{j=1}^{n} ft_{ij} fc_j$$

which multiplies the frequency of occurrence of query term t_i in passage *j* (ft_{ij}) by the frequency of occurrence of concept $_c$ in passage *j* (fc_j), and then sums this product over all the *n* top passages. Hence, this factor measures the co-occurrence of a given query term t_i with the concept c being ranked, over all of the top *n* passages. The greater the amount of co-occurrence (co-occurrence in more passages, greater frequency of co-occurring query term and concept within a given passage), the greater the ranking of *c*. This ranking factor is modified (multiplied) by idf_c, a variation on the familiar global *idf* statistic which penalizes concepts occurring frequently in the collection. This product is normalized by the log of the number of top ranked passages. A small term, δ, is added to prevent any concept from getting a score of zero. The resulting sum is raised to the

power idf_i to emphasize infrequent query terms. The result is a score for *c* relative to query term t_i. Finally, the scores for *c* relative to each of the query terms t_i are multiplied together to obtain the final ranking score for concept *c*. Note that "[m]ultiplication is used to emphasize co-occurrence with all query terms." In other words, if even one of the query terms has a very low co-occurrence score with a concept *c*, that concept will receive a low ranking.

Once the concepts in the top *n* passages have been ranked, the *m* highest ranking concepts are used to form an auxiliary query, which combined with the original query *Q* using a weighted sum operator. In the reported experiments, m was set to 70. These *m* expansion concepts were weighted (within the auxiliary query) according to a linear weighting function such that the highest ranking concept, c_1 received a weight close to 1, and the lowest ranking (*m*-th) concept received a weight of 0.1. The expanded query was then applied to TREC3 and TREC4 data. All runs produced improvement (measured in average precision), but the amount of improvement depended on the number of passages. The best run on TREC3 (200 passages) produced an improvement of 24.4%. The best run on TREC4 (100 passages) produced an improvement of 23.5%. Currently, no method of choosing the optimum number of passages is known, but fortunately, performance is relatively flat for a wide range (30 to 300 passages), at least for the TREC data. If the number of passages is below or above this optimum range, the level of improvement declines.

Note the mixture of global and local analysis in LCA. The *n* passages that best match *Q* are selected globally from the entire collection. Likewise, the two *idf* statistics, idf_c and idf_i, are calculated globally over all the passages in the collection. On the other hand, query term/concept co-occurrence is computed "locally," i.e., it is computed only for those *n* best passages retrieved for *Q*. Note that term occurrence statistics and idf statistics can be computed ahead of time for all the passages in a given collection. At run time, i.e., when a query *Q* is being executed, the only global activity required is retrieval of the *n* best passages. Co-occurrence statistics and the concept ranking can then be computed locally. Hence, retrieval time is very fast.

LSI has been used as a form of query "expansion" in conjunction with (or in place of) relevance feedback. In "[m]ost of the tests … the initial query is replaced with the vector sum of the documents the users have selected as relevant." [Berry et al., SIAM, 1995] The effect is equivalent to incorporating terms from the relevant documents into the query, yet the resulting query vector using LSI factors is much lower-dimensional than if the terms themselves were actually added (which means that the query will execute much faster). [Harman, SIGIR '92] [Hull, SIGIR '94] Dramatic improvements were achieved when even the first relevant document or the average of the first three relevant documents were used.

11.2 Query Refinement (Term Re-Weighting)

Once the query has been expanded, the weights of query terms must be recalculated. A widely used method of re-calculating the term weights given relevance feedback is the Rocchio formula. [Buckley et al., SIGIR '94] The Rocchio formula calculates new term weights from the old term weights and the relevance judgments as follows. Let Q_i^{old} be the existing weight of the *i*-th term, e.g., computed using a scheme like the popular "*ltc*" (see section above on term weights) for a term in the original (non-expanded) query. Of course, if term t_i is an expansion term, then there is

no Q_i^{old}. i.e., $Q_i^{old} = 0$ in the equation below. Let Q_i^{new} be the new weight of query term *i* after re-evaluation. Let |*rel docs*| be the number of retrieved documents judged relevant. Let |*nonrel-docs*| be the number of retrieved documents judged non-relevant. Let wt_i the weight of term t_i in any given relevant or non-relevant document. Let *A, B,* and *C* be three constants to be adjusted experimentally. Then the Rocchio formula [Buckley et al., SIGIR '94] [Salton and Buckley, JASIS, 1990] is:

$$Q_i^{new} = A \cdot Q_i^{old} + B \cdot \frac{1}{|rel\ docs|} \cdot \sum_{rel\ docs} wt_i - C \cdot \frac{1}{|nonrel\ docs|} \cdot \sum_{nonrel\ docs} wt_i$$

Note that the *B* term in the Rocchio formula averages the weights of query term t_i over the relevant documents. (Remember that a given term can have a different weight in each relevant document in which it occurs.) The *C* term averages the weights of query term t_i over the non-relevant documents. Hence, the ratios of *A, B,* and *C* determine the relative importance of the old query (i.e., the previous version of the query), the relevant documents, and the non-relevant documents in modifying term weights in the query. It should be stressed that the query expansion terms are drawn entirely from the documents judged relevant. However, some of these expansion terms may also occur in non-relevant documents. Hence, the *C* portion of the Rocchio formula does not add any terms to the query; its only effect is to reduce the weights of some expansion query terms (or on a first iteration, original query terms) because of their occurrence in non-relevant documents. The effect of the *C* portion of Rocchio may be to make the weight of some terms negative. Terms with negative weights may be dropped. [Buckley et al., SIGIR '94]

This is the "original" Rocchio formula. A modified version of the formula [Buckley, SIGIR '94] re-interprets "*nonrel docs*" to include not just the retrieved documents that the user has explicitly judged non-relevant but all documents in the collection that have not been explicitly judged relevant by the user. The assumption of the "modified" formula is that most of the documents the user never sees, the non-retrieved or lower-ranking documents, will in fact be non-relevant.

Buckley et al. point out a significant virtue of the modified Rocchio formula. By pure chance, a low frequency term may easily happen to occur more frequently in relevant than in non-relevant documents within a given collection. Rocchio averages such a term over all the relevant documents, and all the non-relevant documents, in the collection, not merely the relevant and non-relevant documents in which the term happens to occur. Hence, both relevant and non-relevant documents will make a small contribution to the weight of a low-frequency term. Rocchio weights an expansion term by the difference between these contributions, which will therefore also be small. This is contrasted with the probabilistic BI term weighting formula discussed in section 7.3.1. This formula computes the term weight as the log of a ratio involving the probabilities of occurrence of the given term in relevant and non-relevant documents. For example, if 1/10 of the relevant documents contain the given term t_k ($p_k = 0.1$), and 1/100 of the non-relevant documents contain t_k ($u_k = 0.01$), then t_k's weight, w_k (by the BI formula) is log (11). If t_k is much lower frequency, e.g., $p_k = 0.01$ and $u_k = 0.001$), then w_k is approximately equal to log (10.1). The weight of a low frequency term can be approximately the same as the weight of a high frequency term, as

long as the ratio of their occurrence in relevant and non-relevant terms, is the same. Harman [SIGIR ‘92] notes the same problem with probabilistic term weighting.

In general, the SMART system weighting scheme described above allows one weighting scheme to be applied to query terms and another weighting scheme to be applied to document terms. However, as Buckley et al. [SIGIR ‘94] point out, the Rocchio formula involves adding query weights to document weights so the same weighting scheme, e.g., *ltc*, must be applied to both query and document terms if Rocchio is to be used for relevance feedback. (This is in contrast to the usual scheme, e.g., *lnc-ltc* as described in section 6.3, in which the *idf* is a factor in the query term weight, but not in the document term weights.)

The effect of Rocchio re-weighting is to increase the weights of terms that occur in relevant documents and to reduce the weights of terms that occur in non-relevant documents. This works well if all the relevant documents are clustered near each other in document space.[Salton, ATP 1979] In that case, relevance re-weighting using the Rocchio formula will move the query vector toward the centroid of the cluster of relevant documents and away from the centroid of the non-relevant documents. However, if the relevant documents are not tightly clustered, the “optimum” query may not be effective at retrieval. Even worse, repeated re-weighting may cause the query to wander around the document space, never settling down, because one set of relevant documents may pull the query this way, another set may pull it that way. One possible solution is to use a technique like LSI that captures term dependencies; relevant documents may be far more tightly clustered in a document space whose dimensions are LSI factors than in a conventional document space whose dimensions are actual terms occurring in the documents. Another possibility is to modify the document vectors by adding terms drawn from the user’s query or application domain to the indexes of documents judged relevant. The effect is to move relevant documents closer together in document space, and move non-relevant documents farther away. [Salton, ATP, 1979] Of course, this approach won’t improve performance for the original query, but it will help if the user submits similar queries in the future. A third approach is to cluster the retrieved relevant documents (see section on clustering). If two or more well-defined clusters are detected, one can split the original query into multiple queries, one for each identified cluster, and then proceed with normal relevance feedback.[Salton, ATP, 1989]

Other re-weighting formulas can be effective. In general, term ordering formulas are potentially term re-weighting formulas. The BI probabilistic formula can be applied to relevance feedback data as described above in the section on probabilistic approaches. This formula can be used as both a term ordering formula to select good terms for query expansion, and as a formula for re-weighting query terms after each query iteration. [Sparck Jones, JDoc, 1979] According to Harman [SIGIR ‘92], the effectiveness of “query expansion using the probabilistic model seems to be heavily dependent on the test collection being used … Collections with short documents … generally perform poorly, probably because there are not enough terms to make expansion effective.” Salton and Buckley [JASIS, 1990] note that probabilistic re-weighting does not make use of such useful information as the weights of terms assigned to retrieved documents. “Furthermore, the set of relevant retrieved items [documents] is not used directly for query adjustment … Instead the term distribution in the relevant retrieved items [documents] is used indirectly to determine a probabilistic term weight.” They claim that probabilistic relevance feedback is less effective than vector, e.g., Rocchio, relevance feedback, perhaps for these reasons.

Haines and Croft [SIGIR ‘93] used *rtf*idf* to weight query expansion terms, where *rtf* is the frequency, i.e., total number of occurrences, of the given term in relevant documents. The *rtf*idf* is similar to Harman’s “*noise*frequency*”. Note that they could not have used either BI or Rocchio for re-weighting because they were running their queries against an inference network retrieval engine. For an inference network, “the weight associated with a query term is used to estimate the probability that an information need is satisfied given that a document is represented by that term.” Hence, relevance feedback involves re-estimation of *that* probability rather than estimation of the probability that a document described by a given term is relevant to the given query (as in the BI formula). Moreover, non-relevant documents are not considered at all in their inference network. [Haines and Croft, SIGIR ‘93] Instead, “relevance feedback using the inference network model adds new terms as parents of the query node … and re-estimates the relative weights of the parents’ contributions to [a] weighted sum” representing the belief that the information need expressed by that query is satisfied by a given document.

Buckley et al. [TREC 5} have tried to improve re-weighting by applying relevance feedback not to the entire collection, but to documents in a vector sub-space around the query, called a *query zone*. This is a loose region around the query, such that documents in the region are not necessarily relevant to the query, but are (it is hoped) related to the query, e.g., they may be about the application domain to which the query belongs. For example, a document about computer monitors is not relevant to a query such as, “Which disk drive should I get for my Mac?” but query and document both belong to the domain of computer hardware, and hence may be in the same query zone. Use of query zone may improve re-weighting in two ways: First, a term like “computer” that is very common in the query zone, but much less common in the larger collection will be substantially downweighted, reflecting the fact that the term is good for distinguishing the domain, but not good for distinguishing relevant from non-relevant documents within the domain. Second, a term may be common in the collection as a whole (hence, not good for distinguishing the domain), but very good for distinguishing the relevant from non-relevant documents within the domain. They offer the example of *tire* for the query *recycling of tires*. Such a term will have its weight substantially increased by the query zone approach. Clearly, the choice of a good similarity measure for inclusion in the zone is the key to success with this approach.

Buckley and Salton [SIGIR ‘94] point out that:

> [t]he same relevance feedback techniques can be used in the routing environment, in which a user may have a long-lived information need and is interested in any new documents that match the need. In this case, the user’s query [initially weighted by the training set] can be constantly updated by the system as it receives relevance information about new documents seen by the user. Over the life of the query, thousands of documents could conceivably be returned to the user for relevance judgments.

In a document routing or classification environment, relevance feedback is often provided by a (relatively large) “training set” or “learning set” of documents whose relevance or non-relevance has been judged before the actual routing begins. In a test and evaluation environment such as the TREC conferences, it is common to break a document collection in half, with one half serving as

the “training set” and the other half serving as the “test set” on which the routing abilities of a trained system (or an enhanced, refined query) are to be tested. The crucial point is that the system is trained (at least initially) on a different set of documents than the ones on which it will be required to perform. Hence, there is a danger that the system will be so perfectly “overfitted” to the training set that it will perform very poorly against the “real” documents. Buckley and Salton [SIGIR ‘95] point to an extreme example of overfitting: an information need expressed as a boolean expression in which each known relevant document in the training set is represented by the AND of all its index terms, and these ANDs are then ORed together. Such a query is guaranteed to retrieve all of the relevant documents in the training set, but it has been so specialized that it will do very poorly against new, incoming documents.

Buckley and Salton [SIGIR ‘95] try to avoid the danger of overfitting by what they call Dynamic Feedback Optimization (DFO). They generate an initial feedback query by expanding the initial query using the *N* best terms from the known relevant documents in the training set. (Their ordering criterion for “best” terms is number of occurrences of the given term in the relevant documents, i.e., *rtf*.) They weight the terms using the Rocchio approach. Then (this is where DFO comes in) they refine the query weights by a process that does *not* involve adding any additional terms. Instead, they run a series of passes on the existing set of query terms and weights. On each pass they try the effect of increasing each term weight by a factor which is fixed for each pass but may change (be reduced) from one pass to the next. They test each term weight increase by running the query with just that change against the training set. At the end of a pass, they preserve the term weight increases that improved performance against the training set as measured by average recall-precision on the top *X* documents (200 in their experiments). Number of passes, percentage weight increase per pass, Rocchio coefficients (*A, B, C*) etc., are parameters that can be varied. Interestingly, Buckley and Salton note that the run in which the weights were increased most on each pass produced the best “retrospective” performance on the training set, but the worst performance on the test set, a classic example of overfitting! They conclude that limiting the weight increase on each pass is an essential element of DFO.

11.3 Expansion/Refinement of Boolean and Other Structured Queries

The previous section discussed the expansion and re-weighting of vector space, e.g., term-based queries. Similar techniques, e.g., relevance feedback, can be used to expand and re-weight structured queries, e.g. boolean [Salton, ATP, 1989] [Salton, IP&M, 1988], and phrase-structured queries. [Haines and Croft, SIGIR ‘93]

Salton expands a boolean query by extracting terms from retrieved documents judged relevant as in the vector case. However, instead of merely ranking the terms as candidates for expansion as in the vector case, he uses the term “postings”, i.e., the number of relevant documents indexed by the given term (*rdf*), to estimate the number of relevant documents likely to be retrieved by various boolean combinations, e.g., for three terms t_i, t_j, and t_k, the combinations (t_i AND t_j), (t_i AND t_k), (t_j AND t_k), and (t_i AND t_j AND t_k) are applicable. The combinations that are estimated to retrieve the most additional relevant documents are then ORed to the original query. Obviously, the number of additional terms must be limited to prevent a combinatorial explosion of boolean AND terms. Also, it is clear that boolean expansions generated in this way do not reflect the full seman-

tics that might be derived from a natural language statement of the information need using the same terms.

Strict boolean expressions do not have weights. However, extended booleans do, e.g., the *p*-norm model discussed earlier. The components of a *p*-norm extended boolean query can be weighted by assigning different values of the parameter *p* to different clauses. A higher value (Salton suggests $p = 2.5$) can be assigned to more important clauses, "implying a stricter interpretation of the boolean operators;" less important clauses can be assigned a lower *p* value, e.g., $p = 1.5$, meaning that their boolean operators will be interpreted more loosely (closer to a vector space interpretation).

Haines and Croft [SIGIR '93] ran some relevance feedback tests on structured queries in which the structure consisted of phrase or proximity operators rather than the standard boolean operators. In other words, the query was a term vector in which some of the terms were phrases or single terms in proximity. The tests were run on an inference network search engine (see section on inference networks). They used the classic query expansion and re-weighting technique except that the "terms" added were not single words but phrases or groups of words in proximity. A phrase is a statistical or syntactic co-occurrence of words. The words in a phrase may but need not satisfy a proximity relationship. [Croft et al., SIGIR '91] Croft et al. found that relevance feedback improved the performance of such structured queries but not as much as for queries composed of single words.

11.4 Re-Use of Queries

A lot of effort goes into refining, expanding, and optimizing a query using the methods described above, particularly the methods that require user relevance judgments. Yet once the optimized query is executed to the user's satisfaction it is typically thrown away. (The one notable exception to this is the routing or classification case where a query or topic specification is used on an ongoing basis to identify relevant incoming documents and direct them to the appropriate user.)

Raghavan and Sever [SIGIR '95] suggest that these "past optimal queries" should be saved for Information Retrieval (IR) applications too. Their reasonable assumption is that user information needs will recur. They call queries that satisfy such recurrent needs "persistent" queries. They propose storing optimal persistent queries in a query data base which they call a "query base;" each persistent query in the query base would be associated with the documents that it had previously retrieved. (Obviously, it would not be necessary to store the actual retrieved documents for a given persistent query in the query base, only logical pointers to these documents.)

Raghavan and Sever identify two ways in which this query base could be used in conjunction with a new user query: (1) "If there exists a persistent query with which [the] user query has sufficient similarity, then the retrieval output that is associated with the persistent query is presented to the user." (2) If there is no persistent query in the query base close enough to the user query to provide candidate output as in case 1, the system can find the persistent query that is closest to the user query, and use this persistent query as the starting point of a search through query space for an optimal query via a "steepest descent" search method. Note: The search through query *space* is *not* a search through the query *base*; if the optimal query for which we were searching was already in the query base, this would be case 1. The search method is described briefly below.

What does it mean for two queries to exhibit "sufficient similarity?" Raghavan and Sever don't specify a similarity threshold; indeed, the threshold might depend on the application domain. However, they do make a valuable point: Most of the IR literature, especially the literature devoted to vector space methods, assumes that a query can be viewed as "just another document;" in other words, queries and documents are viewed as occupying the same space, and the same similarity measures, e.g., cosine similarity, are applied whether two documents are being compared or whether a document is being compared to a query. However, the IR literature seldom addresses the issue of computing similarity between two queries. Raghavan and Sever argue that a different kind of similarity measure is required for comparing queries, and they propose such a measure. Then, the threshold for "sufficient similarity" in a given application domain can be expressed in terms of their proposed similarity measure.

Their argument is based on another valuable idea: the concept of a "solution region". Briefly, the search for an optimal query to satisfy the user's need will yield not a single optimal query but a region of query space containing optimal queries. This region is called the "solution region". An important property of a solution region *S* is that if two query vectors *Q* and *Q'* are both members of *S*, then *kQ* ($k > 0$) and *Q*+*Q'* are members of *S* too. On the other hand, cosine similarity is not preserved by such a transformations. Hence, cosine similarity can lead to various anomalies when queries are compared with reference to a solution region. For example, two different "optimal" queries will not be computed as exactly similar (identical) according to cosine similarity even though they are both in the solution region. Moreover, given an "optimal" query Q_{opt0} in the solution region, and two queries Q_1 and Q_2 outside the region, cosine similarity may say that Q_1 is closer (more similar) to Q_{opt0} than Q_2 even though Q_2 is closer to the solution region than Q_1. Their proposed query similarity measure overcomes these difficulties.

The essential idea is to compute a normalized "distance" between two queries Q_1 and Q_2 based on comparing *not* the queries themselves but their respective retrieval sets. The retrieval sets can be compared either on the basis of their IR engine rankings, on the basis of the user's relevance judgments, or on the basis of some combination of the two measures. For example, they propose a normalized distance measure based on ranking such that the distance between Q_1 and Q_2 will be minimum (zero) if they produce identical rankings of common retrieved documents, and maximum (one) if they are in complete disagreement in their rankings. They propose a normalized distance measure based on relevance judgments such that distance will be minimum (zero) if the set of documents judged relevant in the retrieval set for Q_1 is identical to the set of documents judged relevant in the retrieval set for Q_2; the distance for the relevance measure will be maximum (one) if there is no document judged relevant that is common to the retrieval sets of Q_1 and Q_2. In a combined measure, IR engine rankings for documents retrieved by Q_1 and Q_2 are compared as before, but only for document pairs such that the first is drawn from the set of relevant retrieved documents, and the 2nd from the set of non-relevant retrieved documents. In all three cases, the similarity of Q_1 and Q_2 is simply 1—*distance*.

How do they propose to search through query space for an optimal query? Their method assumes that the user supplies more relevance information than is usual in relevance feedback methods. As in other relevance feedback methods, the initial user query goes through a series of refinements. As in other feedback methods, each refinement of the query is applied to the document collection

to obtain a set of retrieved documents. And as in other feedback methods, the user provides relevance judgments for each successive retrieval set. However, the user supplies not just a judgment as to whether each document in the retrieval output is relevant or non-relevant, but pairwise preferences: given any pair of documents D_1 and D_2 in the retrieval set, the user specifies either $D_1 \geq D_2$ (meaning D_1 is either preferable to D_2 or equally good) or $D_2 \geq D_1$. If $D_1 \geq D_2$ and D_2 is not $\geq D_1$, then $D_1 > D_2$ (D_1 is preferable to D_2). Transitivity is assumed to hold, i.e., if $D_1 \geq D_2$ and $D_2 \geq D_3$, then $D_1 \geq D_3$, so the number of preferences the user needs to specify does not explode combinatorially. (Also, note that this reduces to the conventional case if the user divides the retrieval set into a relevant set and a non-relevant set such that each document in the relevant set is preferable to every document in the non-relevant set.) The query optimization process starts by choosing as the initial query, not the query specified by the user, Q_{user}, but the query from the query base that has the maximum similarity to Q_{user}. Call this Q_0. Then the kth iteration, $k = 0, 1, 2$, etc., consists of looking for document pairs D_1, D_2 in the retrieval set for Q_k such that $D_1 > D_2$ but the IR retrieval engine ranks D_2 above (more relevant to Q_k than) D_1. For each such pair, the difference vector D_1—D_2 is added to Q_k. (So, if a given term t_a has a higher weight w_{a1} in D_1 than its weight w_{a2} in D_2, then w_{a1}—w_{a2} will be added to the weight of t_a in Q_k. Similarly, if for some other term t_b, $w_{b1} < w_{b2}$, then the weight of t_b in Q_k will be reduced by w_{b2}—w_{b1}. In other words, the query weights for the next iteration k+1 are moved in the direction of each "preferable" document D_1.) This iterative process continues until it reaches a Q_n whose retrieval set is such that whenever $D_1 > D_2$, D_1 is ranked by the IR engine above D_2. In other words, we have reached a query for which the user's subjective relevance judgments agree with the IR engine's rankings. This Q_n is considered the optimal query Q_{opt}.

12 Fusion of Results

Considerable research has been devoted in IR to the problem of "fusion" of results. The term "fusion" has been applied in IR to two different problems: the fusion of results retrieved from multiple collections, and the fusion of results retrieved from the same collection by multiple methods.

12.1 Fusion of Results from Multiple Collections

"The need to search multiple collections in distributed environments is becoming important as the sizes of individual collections grow and network information services proliferate." [Callan, SIGIR '95] A given query may be submitted to multiple collections. A list of documents, typically ranked, will be retrieved from each collection. The fusion problem is then how to merge these ranked collections for presentation to the user. "[T]he goal … is to combine the retrieval results from multiple, independent collections into a single result such that the effectiveness of the combination approximates the effectiveness of searching the entire set of documents as a single collection." [Voorhees, SIGIR '95] Given that the user wants to see a total of N documents, the problem is to determine how many of those documents to obtain from each of C collections, C_1, C_2, …, C_C.

Note that collections may have different specialties. Hence, the number of documents relevant to a given query Q_1 may vary widely from one collection to another. For this reason, it is generally

unsatisfactory to merge documents by taking equal numbers (N/C) from each collection. Also note that, in general, each collection may be indexed differently, and may be served by a different information server, using a different IR algorithm or combination of algorithms. Hence, even the same set of documents may be relevance-ranked differently with respect to the same query Q_1 by two different information servers. We cannot, in general, assume that the similarities computed by the various information servers are comparable. Hence, in general, we cannot simply take the N documents having the highest similarity scores relative to Q_1.

Voorhees et al. propose two schemes for determining how many documents DR_i to retrieve from the i-th collection such that $DR_1 + DR_2 + \ldots + DR_C = N$. The schemes differ in how the DR_i cutoffs are to be computed. However, the merging of documents given that the cutoffs have been computed is the same in both cases. The merging scheme is to keep track of how many documents remain to be merged from each collection C_i to satisfy its cutoff requirement DR_i. Let DN_i be the number of documents not yet selected from collection C_i to reach its cutoff. (Initially, $DN_i = DR_i$.) At each step, a C-faced die (C = the number of collections) is (conceptually) thrown. The die is biased in proportion to the DN_i so that the probability of selecting the next document from C_i is DN_i/N. Whichever face of the die "comes up", the next document in rank order from the corresponding collection is selected and added to the growing merge stream. The biases of the faces of the die are recomputed, the die is thrown again, and another selection made.

For example, suppose that $N = 10$, $C = 3$, and the cutoffs are $DR_1 = 5$, $DR_2 = 3$, $DR_3 = 2$. Then, the merging would proceed as follows (where D_{ij} is the document of rank j retrieved from collection i): The first document selected would be either D_{11} (with a probability of 1/2), or D_{21} (with a probability of 3/10), or D_{31} (with a probability of 1/5). Assuming for concreteness that D_{11} was selected, the biases would be recomputed so that the next document selected would be either D_{12} (with a probability of 4/9), D_{21} (with a probability of 1/3), or D_{31} (with a probability of 2/9). This process would continue until ten documents had been selected. Note that the order of documents from a given collection C_i in the merged retrieval set reflects the document rankings returned from C_i, and the number of documents from each collection in the merged retrieval set equals its cutoff value. Since C_2 has a cutoff of 3, the merged retrieval set will contain the three top ranking documents from C_2, i.e., D_{21}, D_{22}, and D_{23} in that order, but their actual ranks in the merged list will be determined probabalistically by the throws of the die. It should also be noted that this merge method is completely independent of how the cutoff values are computed or estimated; it merely assumes that the user has obtained cutoff values for each collection in the set whose retrieval outputs are to be fused. The cutoff values might even be obtained after retrieval from each of the individual collections has occurred. That might be necessary, for example, in a dynamic, distributed environment where the collections to be accessed is not known in advance.

Now, how can the cutoff values be computed? Both methods proposed by Voorhees et al. are based on the existence of a static set of collections known in advance, a training set of queries, and the corresponding training set of retrieved documents from each collection for each query in the training set. Each document in the training retrieval set for a given training query is assumed to have been judged relevant or non-relevant to the given query. (That, of course, is what makes it a training set!) As with all methods dependent on a training set, the assumption is that the results obtained with the training queries are predictive of the results that will be obtained with new queries, i.e., that there will be training queries "similar enough" to the new queries with respect to the

given collections. The first method is called the "relevant document distribution" (RDD) method. Given a proposed query Q_1, the k most similar training queries (the "nearest neighbors" to Q_1 in query space) are computed using some similarity measure. (Voorhees uses cosine similarity.) The value of k is a parameter of the method. The retrieval sets for these k most similar queries are then used to compute a "relevant document distribution" as follows: For each collection C_i, determine the average number of relevant documents at rank one for the k nearest neighbor training queries. Then, determine the average number of relevant documents at ranks one plus two for the same k queries, etc. For example, suppose that there are three collections, C_1, C_2, and C_3. Further assume that k = 3, i.e., there are three nearest neighbor training queries to Q_1: QT_1, QT_2, and QT_3. Then our training data includes nine retrieval sets, e.g., the retrieval set for QT_1 applied to collection C_2 is R_{12} The retrieval sets R_{12}, R_{22}, and R_{32} tell us how documents in C_2 were ranked by QT_1, QT_2, and QT_3, respectively. If the rank one document in R_{12} is relevant, but the rank one documents in R_{22} and R_{32} respectively are non-relevant, then an "average" of 0.333 relevant documents (one relevant document divided by three training queries) is retrieved from C_2 at rank one by the set of nearest neighbor training queries. Similarly, if the rank two documents in R_{12} and R_{32} are relevant but the rank two document in R_{22} is non-relevant, then after two documents have been retrieved from C_2 by each of the three training queries, an average of one relevant document has been retrieved (three relevant documents divided by three training queries). We repeat this process for ranks three to N, generating a cumulative average distribution of relevant documents by rank (the RDD) for C_2. Similar distributions are calculated for C_1 and C_3. By examining these distributions, one can determine the optimum cutoff value for each of the collections C_1, C_2, and C_3. The essential point is that the optimum set of cutoffs for a set of collections depends not only on how many relevant documents are retrieved in the top N documents for each collection. It also depends on how these relevant documents are ranked in each collection. For example, suppose that on the average (across a set of k training queries for the given query Q_1), three relevant documents are retrieved in the top ten for each of two collections C_1 and C_2. But suppose that the relevant documents have higher rankings on average in C_1 than in C_2, e.g., ranks 1, 2, and 4 in C_1 vs. ranks 3, 7 and 8 in C_2. The RDD for C_1 is {1, 2, 2, 3, 4, 4, 4, 4, 4, 4}; the RDD for C_2 is {0, 0, 1, 1, 1, 1, 2, 3, 3, 3}. Then for N = 10 (the merged stream returned to the user will contain ten documents), the optimum cutoffs are plainly DR_1 = 2, DR_2 = 8 for which the RDD's predict that the user will retrieve five (2 + 3) relevant documents.

The 2nd proposed method of computing cutoffs from a set of training queries is computationally cheaper but does not take rankings into account.

> For each collection, the set of training queries is clustered using the number of documents retrieved in common between two queries as a similarity measure. The assumption is that if two queries retrieve many documents in common, they are about the same topic … The centroid of a query cluster is created by averaging the vectors of the queries contained within the cluster. This centroid is the system's representation of the topic covered by that query cluster.

For each query cluster in the training set and each collection, a weight is assigned equal to the average number of relevant documents retrieved by queries in the cluster from the given collection among the first L documents (where L is a method parameter). Hence given a query cluster, we have a set of weights, one for each collection.

Given a query Q_1, the query cluster method associates Q_1 with the cluster whose centroid vector is most similar to Q_1 (presumably using cosine similarity or the like again). A cutoff is assigned to each collection as in the RDD method. As in RDD, the cutoffs must sum to N, but this time they are proportional to the collection weights for the given cluster. Hence, the number of documents DR_i taken from a given collection C_i in response to Q_1 is proportional to the average number of relevant documents retrieved from C_i by training queries in the cluster that is most similar to Q_1. It is not at all dependent on how the documents retrieved from C_i by the training queries were ranked (except that they must have been ranked L or better, of course).

Initial experiments using a subset of the TREC topics showed that the relevant document distribution method performed better than the query clustering method, but at a cost in greater processing time and larger data structures. Both methods performed less well than the ideal, i.e., the results that would have been obtained if all the documents were in a single collection. But the decrease in quality was small enough that it appears that these fusion methods are feasible given the assumptions on which they are based: static collections, training queries, etc.

Note that regardless of the method used to compute the cutoffs, DR_i, it is quite possible that some of the DR_i will be equal to zero, reflecting the fact that some collections may have few if any documents relevant to the given query. This is especially likely if some of the collections are specialized to particular topics.

Also note, as pointed out by Voorhees et al., that these fusion methods make no allowance for the possibility that the same relevant document may be retrieved from two or more collections, a possibility that is particularly likely if two or more collections deal with the same specialized topic.

What if the collections are widely distributed and dynamic? Callan et al. [SIGIR ‘95] have proposed an alternative fusion approach, an extension to the inference network method, a probabilistic approach to IR (see preceding section) that has been applied in the INQUERY system. The inference network approach has previously been applied to retrieval of documents in a collection. In the paper discussed here, it has been extended to apply to fusion of outputs from collections by re-interpreting some of the basic statistical measures at the collection level. Hence, two levels of index are created: First, there is the traditional document level where documents are indexed by the terms they contain, their frequencies within each document, and the distributions of these terms over the entire collection. Then, there is a new collection level of index where collections are in effect treated as very large “virtual documents” indexed by terms in ways analogous to the document level indexes. It would appear that the same kind of extension could be applied to any method based on term indexing of documents. The inference network defined by this collection level index is called a Collection Retrieval Inference Network, or CORI net for short.

For example, term frequency (*tf*) meaning “number of occurrences of a given term within a given document” is replaced by document frequency (*df*) meaning “the number of documents within a given collection containing occurrences of a given term.” Similarly, the number of documents in a collection, D, is replaced by the number of collections to be accessed, C. The document frequency, in turn, is replaced by collection frequency (*cf*) meaning “the number of collections (out of the total C to be accessed) containing a given term.” And, inverse document frequency (*idf*),

defined for a given term in a given collection as log(*D*/*df*), is replaced by inverse collection frequency (*icf*), defined for a given term in a given set of collections as log (*C*/*cf*). The *idf* is a measure of how few documents in a given collection use a given term, i.e., it is zero if the term is used by every document in the given collection and is maximized if only one document in the given collection uses the term (which may indicate that it is a good descriptor of the topic with which the given document deals). Similarly, the *icf* is a measure of how few collections (out of the *C* to be accessed) use a given term. Note: Actually, Callan et al. use a slightly more complex formula for *icf*:

$$icf = \frac{\log \frac{C + 0.5}{cf}}{\log(C + 1.0)}$$

In the usual application of inference networks to document retrieval from a single collection, the presence of a given term in a given document may provide evidence to increase the probability that the given document satisfies the user's information need. In the extension to multiple collections, a given term may also provide evidence about the probability that a given collection contains documents satisfying the user's information need.

Hence, retrieval of documents for a given query *Q* from a distributed collection of documents becomes a two-stage process: First the collection level index is used to rank the collections relative to *Q*. Then, if the number of collections is large, document retrieval is applied only to the higher ranking collections, the ones most likely to contain documents relevant to Q. If the number of collections is small, all the collections (or all with score above some threshold) may be accessed, but greater weight may be given to documents retrieved from higher ranking collections.

One *very* important limitation of a collection level index should be noted here. In an ordinary document level index, each term (and associated term frequency) is associated with the documents in which it occurs. Hence, if two terms *t1* and *t2* appear in the index, one can determine whether they co-occur in the same document(s). (However, if they co-occur in document *D1*, one can't tell whether they occur in close proximity, unless term position within the document is retained in the index, which would greatly increase the index size, or the actual document itself is retrieved and examined.) If two terms *t1* and *t2* appear in a collection level index, one can tell whether they co-occur in the same collection, i.e., the same "virtual document," but we can't tell whether they co-occur in the same document within that collection, let alone whether they occur in close proximity within a given document. This means that the problem of determining whether two co-occurring terms are semantically related, difficult enough when the terms are known to co-occur in a given document, becomes much more severe when the terms are only known to co-occur in a given collection, perhaps in entirely different documents. Methods of compensating for these limitations are discussed later.

The extension of the method from the document level to the collection level introduces a problem relating to large collections analogous to the problem identified by Lee (discussed earlier) for large documents. Lee pointed out that a large document is apt to deal with multiple topics. Obvi-

ously, this is even more likely with respect to a large collection! In the case of a large document, Lee points out that "maximum normalization," i.e., normalization of term weights for a given document by maximum term frequency in the given document, is liable to drag down the weights of terms relating to a relevant topic; specifically, if the highest frequency term in large document D_i deals with sub-topic *A*, it will drag down the weights of terms in D_i dealing with relevant sub-topic *B*. Callan et al. point out the same difficulty for large collections. At the collection level, maximum term frequency (tf_{max}) is replaced by maximum document frequency (df_{max}), the number of documents containing the most frequent term in the given collection. (Here, "most frequent term" means the term that occurs in the most documents of any term in the collection.) Normalizing document frequency (*df*) by df_{max} for a large collection can produce a similar effect to that noted by Lee for large documents. If the large collection contains a significant subset of documents relevant to an information need Q_1, maximum normalization may obscure the presence of this subset if other larger subsets deal with other topics, and particularly, if the most frequently occurring term is associated with one of these other topics.

Callan et al. propose to deal with this maximum normalization problem for collections by using a different kind of normalization for df:, scaling by $df + K$ where *K* is a large constant, rather than scaling by df_{max}. They suggest *K* should be a function of the number of documents in the collection. (They have tried a similar approach at the document level; in that case, the scaling factor is $tf + K$ and *K* is a smaller constant, a function of document length.)

The CORI net approach allows the system to rank collections by the probability that they will satisfy the user's information need, just as the document level inference network allows the system to rank documents within a given collection relative to the user's information need. One virtue of this approach is that the same system can perform the ranking at both levels; indeed, the same algorithm can be applied to both the document level inference network and the collection level CORI net, since the indexes have the same structure and analogous semantics. Another virtue is that the collections, like the documents within a collection, receive scores, not just rankings; the score for a given collection reflects the probability that it will contain documents that satisfy the user's information need.

Once each collection has been searched, we must address the same problem as with the "relevant document distribution" and "query clustering" methods discussed above: how to merge the ranked (and scored) outputs from each of the searched collections into a single merged, ranked output to present to the user. Because the CORI net approach generates a score for each collection, it is possible to compute a weight for each collection without using or requiring a query training set as in the query clustering method. An example of a formula for calculating collection weights from their CORI net is:

$$w_i = 1 + C \cdot \frac{s_i - \bar{s}}{\bar{s}}$$

where w_i is the weight assigned to collection C_i, *C* is the number of collections searched, s_i, the CORI net score of collection C_i, and $\bar{s}$ is the mean of all the collection scores. Bear in mind that

these collection weights (and the scores from which they are derived) are relative to a given user query or information need. Given a different query, different weights would be computed for the collections searched.

Once a weight has been computed for each collection, how should one use them to merge the retrieved (and ranked and scored) document outputs of these collections into a single stream? One could clearly use the same approach as in the query clustering method: convert the weights into proportional cutoffs summing to some desired total *N*, and then merge the retrieval sets using the merge algorithm described above. However, both the query clustering and relevant document distribution methods assume that the retrieval sets are ranked, but make no assumption about the documents in each set being scored. The inference net approach produces a score for each document as well as for each collection. If one can assume that the scores assigned to retrieved documents from one collection are strictly comparable to the scores of documents retrieved from another collection, then one can of course merge the documents into a linear order by score alone. However, experience shows that even when the same method, e.g., the inference net method, and the same indexing, is applied to different collections, the scores for a given query may not be comparable due to large variations in the statistical properties of the collections. Hence, the CORI net approach to merging is to compute a global score for each document as the product of the weight of the collection from which it was retrieved, and its "local" score within that collection. Documents are then merged by ranking them according to these global scores. The effect is to favor documents from highly weighted collections (equivalent to favoring documents from collections with high cutoffs in the cutoff-merge method); however, documents with very high scores from lower weighted collections are also favored (equivalent to choosing a very highly ranked document from a collection with a low cutoff in the cutoff-merge method).

Why should it be the case that scores from documents retrieved from two different collections are not necessarily comparable even when the documents are represented and described in the same way in both collections, and the same retrieval algorithm is used to compute document scores (relative to a given query) for both collections? Consider two collections, one containing legal opinions, the other containing papers relating to computers and computer science. Now consider a query Q_1 containing the word "tort." Many documents in the legal collection will probably contain the word "tort." The computer science collection may contain a few documents relating to liability of software engineers in which the word "tort" appears. If *idf* is used in the term weighting scheme, a document in the computer science collection dealing with liability will receive a much higher score relative to Q_1 than a comparable document (even the same document!) in the law collection. Viewing each collection separately, this is quite appropriate; "tort" is a much more significant descriptor in the computer science collection because of its rarity, than it is in the law collection. But the scores cannot be compared directly when Q_1 is used to search the two collections, and the resulting retrieval sets are to be merged. This problem can be remedied by computing normalized global statistics, e.g., a normalized *idf* that reflects the number of documents containing the term "tort" in the law and computer collections combined; this normalized idf has to be computed from the statistics of all the collections to be searched, it has to be computed before any of the searches are executed, and it has to be used by all the information servers in place of the local idf's available to those servers. The effect in the computer-law example is that the scores of documents in the law collection for Q_1 that contain "tort" become higher (because of all the documents in the computer collection that don't contain "tort") and the scores of the few

documents in the computer collection that contain “tort” become lower (because of all the documents in the law collection that *do* contain “tort”). Now the scores are comparable; in effect, the two collections are being treated as if they were one. However, this is an expensive procedure, in computation and communication, especially if the collections are widely distributed. Experiments indicate that ranking based on the product of collection weight and document score is about as effective, and considerably cheaper.

The CORI net method of fusion (or any similar approach using term-based indexing on the collection level) is obviously better suited than the relevant document distribution or query clustering methods to a dynamic environment where the collections are widely distributed, the number and identity of the collections to be searched changes rapidly from one query to another, and the contents of the individual collections themselves change rapidly as new documents are added. This is because it does not require the existence of a query training set of documents which must necessarily have been prepared by applying some fixed set of training queries to a fixed set of collections. Adding additional collections to be searched does not require the major effort of updating a training set to include these new collections. It only requires that each collection be comparably indexed *and* that global statistics for the set of collections to be searched, e.g., *icf,* be updated. The latter, of course may involve some significant run-time expense, but a lot less expense than updating a training set. Rapid updating of individual collections requires (at least) periodic updating of their indexes, and correspondingly, updating of global statistics like *icf* when a set of collections is searched. But again, this is less expensive than regular updating of a training set.

Xu and Callan [SIGIR ‘98] carry the CORI net research further, increasing the number of collections substantially (from 17 to 107), and improving the retrieval process. (They also, it appears, abandon the acronym CORI net, which is a good idea since the essence of the reported research does not depend on the use of an INQUERY inference network, and could be carried out with some other effective IR engine.)

The first important (but not surprising) result they report is that the effectiveness of straightforward word-based retrieval is considerably poorer for 107 collections than for 17 collections. In particular, average precision for a distributed set of collections becomes considerably worse as the number of collections goes up, relative to the precision achieved when the same data is effectively treated as a single integrated, centralized database, e.g., with population wide statistics such as *idf* computed and maintained for the entire set of collections rather than for each collection separately. Note that going from 17 to 107 collections did not involve increasing the amount of data. The same TREC data was used in both cases. The number of collections was increased in the latter case by subdividing the same total set of documents more finely, into a much larger number of subsets. Rather, the important difference was that statistics were computed and maintained in a separate index for each collection, as would be the case in a realistic situation, e.g., a large set of collections on the Internet. Yet with 17 collections (17 indexes) and the CORI net approach described above, average precision was almost as good as for a single, centralized collection with a single centralized index. With 107 collections (107 indexes), the average precision declines by 23% to 32.7%, depending on the cutoff (number of documents retrieved).

To compensate for the loss of precision resulting from distributing the index information over 107 separate indexes, they adopt two strategies: use of syntactically determined phrases as well as

individual words as index terms, and query expansion using the Local Context Analysis (LCA) method described above in the section on query expansion. The value of the former is obvious: If the common words “high,” “blood,” and “pressure” co-occur in a collection, they may occur in entirely different documents. However, if the phrase “blood pressure” occurs as an index term to a given collection, the likelihood that one or more documents in the collection actually deal with the subject of “high blood pressure” is obviously much greater.

The value of query expansion by LCA arises from the fact that terms are added to the original query that co-occur in actual documents. Such co-occurrence may be a significant indicator that the collection contains documents relevant to the topic of the original query. Especially important is the addition of “topic” words, i.e., words that by themselves are strong indicators of the topic under discussion in any document in which they appear. For example, given the topic “high blood pressure,” the expansion may generate words like “hypertension” and “cholesterol” whose presence greatly increases the likelihood that a given collection contains documents about the desired topic, and greatly increases the likelihood that documents retrieved from the given collection will actually be relevant to the given topic.

Note that, as explained earlier, LCA works by retrieving the best passages relative to the given query using a conventional IR engine, and then ranking candidate concepts for expansion on the basis of co-occurrence in the retrieved passages with all the query terms. Hence, LCA requires document indexes at the passage level, i.e., the “documents” are passages (fixed length text windows) within the documents. However, these passage-level indexes are maintained separately for each collection. The only global index is the collection-level index, which only contains term statistics by “virtual document,” i.e., by collection. Hence, the global index can be quite small relative to the large set of collections being indexed. Xu and Callan point out that if document boundary information, i.e., which documents a given term is in, were maintained in the collection level index, the index would be about as large as the set of collections being indexed!

In the reported experiments using TREC3 and TREC4 data, LCA proved quite effective. The average precision using the expanded queries was only slightly less (an average drop of only 2.6% for TREC4) than querying the same data as a single centralized database. Use of phrase descriptors also improved performance substantially for distributed collections, but not as much as LCA. The combination of LCA and phrase descriptors was best of all, but LCA alone accounted for most of the improvement.

Viles and French [SIGIR ‘95] deal with a closely related problem: How often must global statistics be updated as individual collections receive new documents? The situation they consider is not quite the one we have been considering above: a set of independent collections at (perhaps) widely distributed sites. Instead, they consider a single collection (they call it an “archive”) distributed over multiple sites. Each site has a different subset of the total set of documents in the collection, and its own server which maintains indexes for its site and cooperates with servers at other sites. How does this differ from the case of multiple “independent” collections which we have been considering up to now? The primary difference is that each server maintains copies of global statistics such as *idf*. In other words, all sites possess the same global value of *idf* for a given term, a value that applies to the total collection, i.e., the total of all the subsets from all the sites. This global *idf* is essentially the normalized idf discussed above for the computer collection/

law collection example. A given site must maintain such a global *idf* for every term used as an index at the given site. Given that a new document is added to the subset at one of the sites, ideally the update should be disseminated immediately to all the other sites so that they can update all their *idf*'s. This is quite expensive if there are many sites, they are widely distributed, and updates are frequent. Moreover, the addition of a single document at one site is not likely to have a large effect on all the global *idf*'s (or on any of them). Nor is it essential that sites receive update information in exactly the same order that the updates occurred. As Viles and French note, "The goals of an IR system generally do not include serializability of updates on the *idf*." The question Viles and French pose and attempt to answer is this: How often must each site notify other sites about updates it has received so that retrieval performance is not significantly degraded? (They call less-than-immediate dissemination of updates "lazy dissemination.")

To discuss this problem, Viles and French define a dissemination parameter, *d*, and an affinity probability, *a*. The *i*-th site, s_i, knows about all its own documents, i.e., the documents physically stored at s_i. The site also "knows about" the first *d*-th fraction of the documents stored at any other site, s_j; here, "knows about" means that global statistics such as *idf* have been updated at s_i to reflect that fraction of the documents at any other site, s_j. Hence, *d* varies continuously between 0 and 1. When $d = 0$, no dissemination occurs. When $d = 1$, each site has "complete" (statistical) knowledge about the documents at every other site. When $0 > d > 1$, s_i's global knowledge is derived partly from its own physical holdings, and partly from disseminated knowledge of documents held elsewhere. Hence, *d* is a parameter for experimenting with the percentage of a site's holdings for which knowledge is disseminated to other sites. The affinity probability, *a*, is a tool for experimenting with different strategies for allocating new documents among the sites. When $a = 0$, documents are randomly allocated among the sites. "When $a = 1$, documents relevant to the same query are co-located, mapping to the case where content has a large influence on document location." (The assumption that documents relevant to the same query are relevant to each other is a simple scheme employed by Viles and French to experiment with content-based allocation of documents; it is not intended "as a recommendation for document clustering.")

Viles and French experimented using SMART v11.0 software and four well-known document collections. Perhaps the most important result was that for all values of *a* the greatest increase in IR effectiveness occurred as *d* was increased from 0.0 (no dissemination) to 0.2. For $a = 0.0$, (random rather than content-based allocation of documents to sites), boosting *d* from 0.0 to 0.2 was sufficient to achieve precision levels for all levels of recall that were "essentially indistinguishable from [a] central archive". At high levels of affinity ($a = 1.0$) corresponding to content-based allocation of documents, higher levels of dissemination were required to achieve precision equivalent to that of a centralized archive. The required dissemination varied from 0.4 to 0.8 depending on the document collection used for the experiment. However, even in these cases, the largest part of the improvement occurred in going from $d = 0.0$ to $d = 0.2$. "Successive jumps in dissemination past the $d = 0.2$ mark yield relatively lower effectiveness gains." Hence, it appears that effective IR can occur in a distributed archive, even when each site has considerably less than complete knowledge of the other sites. However, "[t]here appears to be some minimal sample of documents that a site needs to know about to achieve search effectiveness comparable to a central archive. It remains to be seen whether this sample is a fraction of the whole, or if some minimal number of documents is needed."

12.2 Fusion of Results Obtained by Multiple Methods

A number of researchers have observed that different retrieval methods applied to the same collection to satisfy the same information need can result in retrieving quite different document sets, i.e., there was surprisingly little document overlap across sets, either in relevant documents retrieved, or in non-relevant documents retrieved. [Belkin et al, SIGIR '93] [McGill et al., Syr U, 1979] Moreover, the performance of these different methods tended to be comparable, i.e., the proportion of relevant documents retrieved did not vary as much as one would have expected from one method to another given the small amount of overlap in their respective retrieval sets. [Katzer et al., 1982] These plausible findings would lead one to expect that combining results of multiple methods would lead to improved retrieval, because more relevant documents would be retrieved (or would receive high rankings) from the combination of methods than from any one method alone. Such a result would be plausible, because one would expect that different methods would have different strong points and weak points. This has been called the "skimming effect" [Vogt et al., SIGIR '98] because the user is "skimming" the best documents retrieved by each method.

In contrast, Lee [SIGIR '97] and Vogt et al. [SIGIR '98] find (in more recent research) that different retrieval methods tend to retrieve the *same* relevant documents, but *different* non-relevant documents. This has been called the "chorus effect," [Vogt et al., SIGIR '98] because it means that the more methods retrieve a document (in other words, the louder the "chorus" acclaiming the document), the more likely it is to be relevant.

The phrase "different retrieval methods" can mean quite different things:

1. Different users using the same query formalism, e.g., all the users formulate boolean queries in response to the same information requirement, but each user formulates her query independently of the others. [Belkin et al, SIGIR '93] [Saracevic et al., JASIS, 1988] [McGill et al., Syr U, 1979]

2. The same or different users employing different query formalisms, e.g., natural language vs. Boolean vs. Probabilistic, to satisfy the same information need. [Turtle et al., ACM Trans IS, 1991] [Belkin et al, SIGIR '93]

3. The same or different users employing different vocabularies, e.g., controlled vs. free-text, to satisfy the same information need.[McGill et al., Syr U, 1979]

4. Different document representations, e.g., title vs. abstract, or automatically generated index terms vs. manually assigned terms, or LSI vs. keywords. [Katzer et al., IT, R&D, 1982] [Turtle et al., ACM Trans IS, 1991] [Foltz et al., CACM, 1992]

5. Different weights applied to the query terms and document terms within a single query representation and a single document representation. [Lee, SIGIR '95]

6. Different retrieval (document classification, filtering, query-document similarity) strategies, e.g., vector space cosine similarity with query expansion vs. logistic regression, vs. neural networks vs. linear discriminant analysis (LDA), or linear and logistic regression vs. neural networks vs. pattern recognition techniques. [Schutze et al., SIGIR '95] [Chen, CIKM '98]

As noted above, some of this research has also indicated that combining retrieval or classification results from two or more methods (fusion of results) can produce better retrieval performance than any one of the methods by itself. Typically, results are combined by applying each method separately to a given document, and taking the sum (or equivalently, the mean) of the scores. [Fox, et al., TREC-2] [Lee, SIGIR '95] [Hull et al., SIGIR '96] Note that there are two cases here: In the ad hoc query case, a query is executed against a given collection by each of several methods, e.g., several IR engines each employing a different method of computing document-query similarity. In that case, each individual run returns a ranked list of documents, with each retrieved document in the ranked list assigned a similarity or probability score. Then, for each document retrieved by at least one run, the scores assigned to the given document by each run are summed (or averaged); of course, if a given document is not retrieved at all by a given run, its score for that run is usually considered zero. On the other hand, in the routing (or filtering) case, a single stream of documents is classified. Each document is classified by each of the classification methods. Each classifier assigns a score to the given document with respect to each query. The scores assigned to the given document for a given query are then summed (or averaged) as in the ad hoc case.

Variations on this simple combination scheme are possible. A weighted sum may be employed if there is reason to believe that one method is more reliable than another. If the scores are probabilities, it "may make more sense" to average logodds ratios. [Hull et al., SIGIR '96]. If it is desired to give particular favor to documents retrieved by multiple methods, even more favor than does a simple sum of the scores, then one can employ Fox and Shaw's [TREC-2] CombMNZ function, defined as the sum of the similarities *times* the number of non-zero similarities. Lee's [SIGIR '95] findings with this function are discussed below.

In some of the cases listed above, the "multiple methods" are multiple query formulations, rather than distinct IR engines or classifiers. In case one, multiple users employ the same formalism. Hence, multiple individual runs correspond to multiple users, each formulating a query using the same formalism, in response to the same information need, and executing their queries against the same collection via the same IR engine. Hence, the individual runs can be combined exactly as above, but instead of trying to balance the strengths and weaknesses of different retrieval or classification methods, we are trying to balance the strengths and weaknesses and backgrounds of different users. The second "multiple method" case also involves multiple queries applied to the same information need, but in this case each query (whether generated by the same or a different user) employs a different formalism. So, once again, individual runs can be combined using the above methods, but now it is the strengths and weaknesses of different formalisms (assuming the users have comparable experience and ability, of course) that are being balanced. It should also be noted that, using a system like INQUERY, it is possible to combine separately formulated queries into one "super" query which is then executed normally by the INQUERY retrieval engine; in that situation (not discussed here), it is the queries (not the results) that are fused together, and only one output retrieval set is produced. [Belkin et al, SIGIR '93]

To complicate matters further, two distinct query formalisms *may* indicate two distinct retrieval methods, e.g., a term query may be treated as a term vector and evaluated by cosine similarity, while a boolean query is presumably being executed by boolean logic evaluation. However, a term query and a natural language query may both be evaluated as term vectors, i.e., in some cases, key

terms are extracted from a natural language query exactly as from a document to form a term vector.

Why should fusion of results produce better performance, e.g., better precision for a given level of recall? Belkin et al. [SIGIR '93] suggest two general reasons: First (and most obvious), if there is relatively little overlap between the document sets retrieved by two methods, and the methods (taken separately) exhibit comparable performance, the implication is that each method only retrieves a different fraction of the relevant documents. Hence, merging the best documents retrieved by each method should result in a set containing a higher percentage of relevant documents than any single method alone. Presumably, each query formulation or term weighting method or retrieval strategy has its own strengths and weaknesses.

Second, if there is some non-zero probability p_i that a method m_i will retrieve a relevant document D_{rel}, then the probability that at least one of several such distinct methods, m_j, m_k, and m_l, will retrieve D_{rel} is surely greater. Hence, a document retrieved by several different methods is more likely to be relevant than a document retrieved by one method alone. If a document is retrieved (or classified) as relevant to a given query by more than one method, i.e., more than one method assigns it a score above a specified threshold, the probability that it will "make the cut" when the outputs are combined is greater than if only one method retrieves the document. This probability is greater still if the rank of a document is raised in proportion to the number of methods that retrieve it. Saracevic and Kantor [JASIS, 1988] didn't actually return the intersection, but they found that "the odds of a document being judged relevant increased monotonically according to the number of retrieved sets that it appeared in." [Belkin et al., SIGIR '93]

It should be noted that all of the fusion results discussed in this section represent averages over a set of queries. If one looks closely at the results for individual queries, one finds that some methods work very well on some queries, other methods work very well on other queries. Hence, if one knew in advance which method was best for each query, applying that method would produce better results than fusion. But since, in the current state of the art, one generally does not know which method is best for a given query, fusion of the results of multiple methods represents the best compromise.

Let's consider several of the above examples briefly. Lee [SIGIR '95] combines the results of pairs of retrieval methods. All of the retrieval methods use the term-based vector space method (with the SMART system as a testbed). Each method differs from the others in the weighting scheme used. A weighting scheme (see section on "Classification of Term Vector Weighting Schemes") is characterized by two three-character codes, one three character code specifying the weighting scheme applied to the documents in the target collection, the other the weighting scheme applied to the query. Lee argues theoretically and demonstrated experimentally that "different classes of weighting schemes may retrieve different types of documents - different sets of document (both relevant and nonrelevant)." Specifically, weighting schemes that employ cosine normalization of documents (he calls this class *C*) are better at retrieving single topic documents of widely varying length. On the other hand, weighting schemes (called class *M)* that employ *maximum normalization* of documents, i.e., normalization of term frequency by maximum term frequency within a given document, but that do not employ cosine normalization, are better at retrieving those multi-topic documents in which only one of the topics is relevant to the given

query. Hence (as one would expect), combining the results of a class *C* run with the results of a class *M* run produced significant improvement over the results of either run alone. However, it is evident that the extent of improvement (if any) is dependent on the characteristics of the collection and the query.

However, Lee defines a third class of weighting schemes, *N*, consisting of schemes that use neither cosine nor maximum normalization. Such schemes tend to favor long documents over short documents. Surprisingly, combining a *C* run with an *N* run also produced improvement. As Lee summarizes, "we can get significant improvements by combining two runs in which one performs cosine normalization and the other does not if the two runs provide similar levels of retrieval effectiveness." Lee notes that "the combinations between class *C* and the other classes have less common documents than those between classes *M* and *N*, which means that cosine normalization is a more important factor than maximum normalization in retrieving different sets of documents."

It should be noted that, as discussed above in the section on normalization of term vectors, the pivoted unique normalization (*Lnu*) scheme developed by Singhal et al. appears to achieve the same kind of improvement with a single retrieval run that Lee achieves by fusion of output from multiple runs, each run using a different, older weighting scheme. In other words, *Lnu* appears to achieve (and perhaps improve on) the combined benefits of several older weighting schemes.

Finally, Lee combined an extended boolean (*p*-norm) run with a vector run, and achieved improvement both by combining *p*-norm with a *C* class and by combining *p*-norm with an *M* class vector weighting scheme.

In all of the above cases, each individual run produces a list of documents ranked by similarity. In each combined run, the results of the participating individual runs are combined so that each retrieved document receives a combined similarity score, and the documents are ranked by these combined scores. In all cases, Lee chooses the top-ranking *N* documents ($N = 200$), i.e., the documents with the top *N* similarity scores. The problem is how to compute the combined similarity for a given document. The vector space runs generate document rankings based on cosine similarity values (similarity of each document to the given query). The extended boolean runs generate a similarity score based on the *p*-norm model. The range of *possible* similarity values for cosine similarity or *p*-norm is always zero to one. However, the range of actual similarity values for the same query applied to the same document collection will be different for each model and weighting scheme. Hence, the similarity values must be normalized to make them comparable. The formula:

$$Normalized\_Similarity = \frac{old\_sim - minimum\_sim}{maximum\_sim - minimum\_sim}$$

converts each similarity, *old_sim*, calculated for a given query in a given individual retrieval run, to a value in the common range zero to one, i.e., the largest similarity value, *maximum_sim*, will be mapped into one, the smallest similarity value, *minimum_sim*, will be mapped into zero, and all

intermediate similarity values will be mapped into values between zero and one. (Lee also notes that if one knew in advance which retrieval runs were likely to perform better, it would make sense to weight the similarity values of those runs more heavily; however, in general, for ad-hoc queries and arbitrary collections, one doesn't have that kind of information.) Once all the similarity values have been calculated for the runs to be combined, the retrieval sets *could* be merged in straightforward numeric order by (normalized) similarity value, and the *N* documents with the highest similarity returned.

However, straightforward numeric merging by normalized similarity value has the drawback that it does not take into account the number of retrieval sets in which a given document occurs. As noted above, the more retrieval sets in which document D_i occurs, the more likely it is to be relevant to the given query. Hence, Lee, following Fox & Shaw [TREC-2, 1994], computes a combined similarity value for each document equal to the sum of its similarity values in each retrieval set in which it occurs. (Naturally, the similarity of a given document is zero in a retrieval set in which it does not occur.) Documents are then ranked by these combined similarity values, and the top *N* selected as before. Note that ranking each document by the sum of its similarity values is equivalent to ranking the given document by the mean of its similarity values.

Fox and Shaw [TREC-2, 1994] combined the results of extended boolean (*p*-norm) query runs with the results of "natural language vector query" runs, i.e., vector queries obtained by extracting and stemming terms in the usual way from natural language topic statements. As noted above, they merged the results of multiple retrieval runs for a given query by computing a combined similarity value for each document retrieved in at least one run. In addition to the sum (or equivalently mean) of the similarity values (which they call "CombSUM"), they also tried two other methods of combining similarities: In their "CombANZ" method, they divide the CombSUM value by the number of retrieval runs in which the document received a non-zero similarity; the effect is to compute a mean that ignores retrieval runs in which the given document is not retrieved. In their "CombMNZ" method, they multiply the CombSUM value by the number of non-zero similarities the given document received; the effect is to enhance the importance of retrieval of a given document by multiple runs.

Fox and Shaw ran five individual retrieval runs. Three of these runs used *p*-norm extended boolean queries, each with a different value of *p* (1.0, 1.5, and 2.0). The other two runs used vector queries. The vectors were generated from TREC-2 natural language topic descriptions. The "short vector" run took its query terms from the Title, Description, Concepts, and Definitions sections of the standard TREC-2 topic format. The "long vector" run took its query terms from all of those sections plus the Narrative section as well. In contrast to Lee, who only combined pairs of retrieval runs, Fox and Shaw combined all five of their individual retrieval runs as well as combining two or three individual runs. This led them to the following interesting observation: "While combining all five runs produced an overall improvement in retrieval effectiveness over each of the [individual] runs, the same does not always hold true when combining only two or three runs." Thus, the effectiveness of combining runs can depend not only on the query and the collection, but also on how many runs are combined and which ones.

In later research, Lee [SIGIR '97] builds on the work of Fox and Shaw. He studies combinations of up to six individual retrieval runs, using results derived from TREC-3. In his own previous

work [SIGIR ‘95], described above, he showed theoretically, and demonstrated experimentally, that two different term weighting schemes, appropriately chosen, could result in retrieving different relevant documents from the same collections, even when the weights are applied to term vectors in both cases, and cosine similarity is the method used to compute the individual query-document similarity in all cases. Improvement resulted in his experiments provided the two methods contrasted appropriately, and were equally effective. Even in that work, he also allowed for the effect of multiple runs retrieving the same document by using one of Fox and Shaw’s combination functions, CombSUM, to compute the combined similarity of a document that was retrieved by both runs. In the SIGIR’97 work, he finds that the improvement that results from combining multiple TREC runs derives primarily from the fact that the runs tend to retrieve the *same* relevant documents (which pushes up the combined similarity of each document retrieved by more than one run), but *different* non-relevant documents (which pushes down their combined similarity scores). He went beyond Fox and shaw by (1) normalizing the similarities, using his SIGIR ‘95 formula given above, so that the similarity scores combined would be more comparable, (2) by showing that CombMNZ (defined above), which emphasizes the importance of being retrieved by multiple runs, gives even better results than CombSUM, and (3) by computing the actual amount of overlap across individual runs, for relevant and non-relevant documents. Lee computes the overlap on a scale running from zero (no overlap) to one (total overlap), using the functions: $R_{overlap} = (R_{common} \times 2)/(R_1 + R_2)$ and $N_{overlap} = (N_{common} \times 2)/(N_1 + N_2)$. He finds values of $R_{overlap}$ in the range 0.75 to 0.82, values of $N_{overlap}$ in the range of 0.30 to 0.40. Plainly, the proportion of overlap among retrieved relevant documents is much higher than the proportion of retrieved non-relevant documents.

Turtle and Croft [ACM Trans IS, 1991] used the inference network approach (see section 7.3.2) to combine Boolean and term-based (they call the latter “probabilistic”) queries for the same information need. Both query formulations were based on an initial natural language statement of the problem. The queries were combined using a weighted sum. They found that the combination produced better results (better precision at most recall levels) than either type of query formulation by itself. However, they found that the improvement was due to the fact that the boolean query formulators used the boolean structure to capture information present in the natural language statement of the information need that was lacking in the term-based query. Hence, the boolean query retrieved a subset of the documents retrieved by the term-based query. Adding the boolean query to the term-based query produced not additional documents but a better ranking of the documents retrieved by the term-based query, resulting in better precision at a given recall level. They conjecture that if the boolean formulators had been asked to produce high-recall boolean queries, they would have added additional terms, and retrieved additional documents not retrieved by the term-based queries. It should be stressed that Turtle and Croft were combining two query formulations to produce a single query, and then running this single combined query against their retrieval engine. They were *not* combining the retrieval sets returned by the two queries run separately.

Belkin et al. [SIGIR ‘93] generated extended boolean queries which were executed using the INQUERY system’s extended boolean operators which are similar to, but not identical to, the operators of the P-norm model. They “recruited experienced on-line searchers to generate search statements for the same search topics.” The recruits were told to generate boolean queries using AND, OR, NOT, any degree of nesting desired, and operators (at the word level only) for adja-

cency, i.e., two terms next to each other, proximity, i.e., terms within a given distance of each other, and order, i.e., terms in proximity to each other and occurring in a specified order. They were not told the system (INQUERY) to which the queries were to be submitted or that the queries were to be executed as extended booleans. The queries were executed separately, and then in combination. However, in contrast to Lee, and Fox and Shaw, as described above, Belkin et al. did not combine the results (retrieval sets) of the individual query runs; instead, they used INQUERY's ability to combine the queries themselves. The queries were combined using the INQUERY "unweighted sum" operator. (Actually, the experiment was more complex. They started with five query "groups", each group consisting of a query for each of ten TREC-2 topics.) The queries were combined cumulatively, e.g., first a single query group, then two of the query groups (for the same set of search topics) combined, then three, then four, and then all five query groups for the given search topic combined. Results were reported for group 1, then group 1 plus group 2, etc. Unweighted sums were used to combine queries across groups. Combination within a group is not discussed. Then the combination of all five boolean queries was further combined with a natural language query based on the corresponding TREC-2 natural language topic description. This natural language query provides a powerful baseline because it takes advantage of a "version of INQUERY [that] performs a sophisticated analysis of the TREC topics … including recognition of country names and automatic syntactic phrase generation …" Combination of Boolean queries (translated into INQUERY) and the corresponding natural language baseline INQUERY queries (designated "INQC" by Belkin et al.) was done with various weighted sums.

The results obtained by Belkin et al. indicated that combining Boolean queries (actually query groups - see above) improved performance. An interesting point they note is that in some cases adding a group that performed poorly on its own to a group that performed well on its own resulted in better performance than the good group by itself. Hence, one cannot always judge the performance of a combination of methods solely by evaluating the methods separately. However, combining the Boolean queries with the INQC queries reduced performance when the boolean and the INQC queries were given equal weight. Significantly improved performance was obtained only when the INQC query was given a weight four times that of the combined boolean query. By contrast, Fox and Shaw obtained significant improvement (see above) when natural language queries were combined with extended boolean queries. However, as Fox and Shaw note, the methods of combination are not strictly comparable because Belkin et al. combine queries while Fox and Shaw combine output retrieval sets.

Foltz and Dumais [CACM, 1992] combine two vector space methods: key-word, i.e., terms are words, and LSI, i.e., terms are LSI factors. Their application is a routing/filtering rather than an IR application. The task they address is to assign abstracts of incoming technical reports to users based on *user profiles*, and *document profiles*. A user profile is a list of words and phrases provided by the user to characterize her technical interests. The document profile for a given user is the set of abstracts that the given user has previously rated as highly relevant to her interests. Hence, the query or information need for a given user is the user's profile or her document profile. The document "collection" is the stream of incoming technical reports. Hence, there are four IR methods:

1. Vector space retrieval by calculating the similarity of an incoming abstract and a given user profile,

2. Vector space retrieval by calculating the similarity of an incoming abstract to abstracts previously rated highly relevant by a given user,

3. Same as (1) but with similarity calculated in the reduced dimension LSI space.

4. Same as (2) but with similarity calculated in the reduced dimension LSI space.

Methods were combined by sending each user monthly the top seven abstracts selected by each of the four methods. This meant that each user could receive up to 28 abstracts per month. But since a given abstract could be selected by more than one method, the users actually received an average of 17 abstracts per month. Foltz and Dumais found that as the number of methods that retrieved a given abstract increased, the "mean relevance rating" increased too. (Each user was asked to rate the abstracts she received each month for relevance on a scale from one for non-relevant to seven for very relevant.) The rating went up from about three for an abstract selected by one or more methods to five for an abstract selected by all four methods. But of course, the number of abstracts selected by all four methods was much less (about 5%) than the number of abstracts returned to all users. However, ratings also improved if one was more selective for a given method, e.g., only selecting abstracts above a given similarity threshold. So the mean rating for abstracts selected by all four methods was compared to the mean rating for the top 5% of abstracts selected by each method separately. The mean rating for documents selected by all four methods together still came out on top, though not by as much, e.g., a mean of 4.54 for abstracts selected by a single method vs. 5.04 for abstracts selected by all four methods.

All of the above cases involve combination of IR algorithms or query formalisms, e.g., boolean and vector space, vector space with two different weightings, etc. However, if a training set of documents with relevance judgments is available (as is often the case in routing and filtering applications), one can make use of general methods of machine learning, methods not specific to IR. Each method can be trained to classify documents using the training set. The result is a set of predictive models, one for each learning technique. These predictive models can be combined just as traditional IR methods are.

For example, Hull et al. {SIGIR '96] study the combination of four methods: Only the first, Rocchio query expansion based on relevance feedback, derives from the IR field. The other three are general purpose "learning" methods, employed to generate a predictive model for document classification. They are: "Nearest Neighbors," Linear Discriminant Analysis (LDA)," and a "Neural Network" fitting a logistic model. Hull et al. study a filtering application. Hence, each predictive model must classify each incoming document as either relevant (accept the document), or non-relevant (discard the document. Each of the four resulting models, given a document to classify, generates a probability-of-relevance score. Hull et al. try several approaches to combining the scores for a given document. (1) Most simply, they compute a straight average of the scores. (2) Next, "given that they are working in a probabilistic domain," they average the logodds ratios, and then reconvert this average back to a probability. (Given a score interpreted as a probability, p, the logodds ratio is defined as $\log(p/(1-p))$. See the section on the Probabilistic Approach.) They point out that probabilities derived by straight averaging will tend to have much less variability than probabilities derived from averaging of logodds ratios, In particular, if one of the classifiers is

very certain of the relevance (or non-relevance) of a given document, the probability derived from logodds ratio averaging will be very close to one (or zero). In general, this is *not* the case with straight probability averaging. Hence, logodds ratio averaging will reflect the certainty of an individual classifier more clearly and directly than straight probability averaging. Both straight probability averaging and logodds ratio averaging were found to outperform the individual classifiers for ranking documents, but for a filtering application, where the important criterion is accurate calculation of relevance probability (or other similarity score) relative to a filtering threshold, the neural network classifier outperformed both classifiers. (3) Hence, to improve calculation of average probabilities, Hull et al. "renormalized the probability estimates via logistic regression using the relevance judgments from the training set." They found that "after normalization, the probability estimates [were] much more accurate, scoring significantly better than the neural network" except at very low thresholds.

All of the above examples of fusing different IR methods involve fusion of a small number of manually selected methods. Bartell et al. [SIGIR '94] have developed a method for automatically combining "experts," i.e., modules executing different IR methods. The method involves a heuristic gradient-based search over the space of possible combinations and can be applied to a large number of experts (although the two tests of the method discussed in their paper involve two experts and three experts respectively). The method is independent of how each expert performs its IR task; the only requirement, satisfied by an increasing number of IR systems, is that the experts must all return *ranked* output, i.e., each system must return a numerical estimate of the degree of relevance of each retrieved document to the given query. A notable feature of this fusion method is that it optimizes the *combined* output of all the participating experts, rather than evaluating the performance of each expert separately. This is significant in light of the finding of Belkin et al. (noted above) that one cannot always judge whether a given IR method will make a positive contribution to combined performance based solely on evaluating the given method separately.

In the Bartell model, each expert *i* returns a numeric estimate $E_i(Q, D)$ of the degree of relevance of document *D* to query *Q*. They combine these estimates into a single overall estimate, $Re(Q, D)$ of the degree of relevance of *D* to *Q*. In this paper, they use a linear combination of the estimates, e.g., for three experts, they have:

$$Re(Q, D) = \Theta_1 \bullet E_1(Q, D) + \Theta_2 \bullet E_2(Q, D) + \Theta_3 \bullet E_3(Q, D)$$

Their goal is then to find values of the parameters Θ_i "so that the overall estimates result in the best ranking of documents possible." Optimization is based on a training set, i.e., a training set of documents, a set of training queries, and a relevance judgment for every document retrieved by a given training query. The relevance judgment is expressed as a preference relation, i.e., user prefers D_1 to D_2 for any pair of documents retrieved by a query *Q*. (This is the same kind of preference relation used by Rhagavan and Sever [SIGIR '95] as discussed above in regard to reuse of

optimal queries.) If only the usual two-valued judgment, relevant or non-relevant, is available, then the preference relation reduces to preferring the relevant to the non-relevant document. "The goal of the optimization is to find parameter values such that the [combined] system ranks document D_1 higher than document D_2 whenever D_1 is preferred by the user to D_2." A gradient-based numerical optimization technique is used. (Note that this is very similar to Rhagavan and Sever's use of the preference relation in a "steepest descent" search for an optimum query in query space.)

Most of the examples above combined two or more document classifiers and studied the performance of the combination relative to that of the individual classifiers. Lee also identified certain cases where combination would be effective, and factors contributing to the success of actual experimental combinations. However, Vogt et al. [SIGIR '98] studied more comprehensively the factors contributing to effective combination. They limited their study to linear combinations, and also limited themselves to combinations of two classifiers. The combinations they studied were derived from TREC5 ad hoc query data. Since there were 61entries in the ad hoc competition, Vogt et al. were able to form (61*6))/2 = 1830 pairs for each query. They studied 20 queries for a grand total of 1830*20 = 36,600 "IR system (method) combinations." Although they drew their data from the ad hoc query competition, the fact that they combined systems on a per-query basis means that the results are more applicable to the routing application.

Their theoretical approach was to identify a set of method performance features. Some of the features were measures of the performance of an individual system (method), e.g., average precision. Others were pairwise measures. For example, Guttman's Point Alienation (GPA) is a measure of how similar two document rankings are to each other. Another pairwise measure employed was the *intersection*, i.e., the number of documents retrieved by both methods. Following Lee, they also computed $R_{overlap}$ and $N_{overlap}$ (see above). The former measures the proportion of relevant documents retrieved by both systems; the latter measures the proportion of non-relevant documents retrieved by both systems. They then performed a multiple linear regression, using the actual TREC5 data as the training set, the method performance features computed from this set (for all system pairs and a given query) as the independent, i.e., predictor, variables of the linear regression equation, and the average precision of the optimal combination (for the given query) as the dependent variable of the equation, the variable to be predicted. The regression then computes coefficients for the predictor variables in this equation. These coefficients can be interpreted as indicating how much each predictor contributes to the overall estimate of the dependent variable.

The results they obtained indicate that the best time to combine two systems (methods) linearly is when (1) at least one system exhibits good performance, (2) both systems return similar sets of relevant documents, and (3) both systems return dissimilar sets of non-relevant documents.

12.3 Fusion of Results Obtained by Multiple Versions of the Same Method

In the previous section, we discussed techniques for combining multiple classification methods, where the training set used to set the parameters of each method was the same, but the IR or machine learning algorithm was different for each classifier. In this section, we discuss approaches where the training set, the machine learning (ML) method, and the underlying IR method are the same, yet multiple classifiers are obtained. This is accomplished, e.g., by taking multiple samples from the training set ("resampling") with replacement and using each sample as

a new training set ("*bagging*"), or by weighting the training documents differently in each training session ("*boosting*"). Note that since reweighting the training set is equivalent to changing the number of occurrences of each document in the training set, boosting can also be viewed as a "resampling" method. A number of variants of these approaches are known, mostly derived from the machine learning community. Breiman [TR, 1996] characterizes this entire family as Perturb and Combine (P&C) methods, i.e., *pertur*b the training set a number of times to create a number of new training sets, generate a classifier for each training set created by perturbation, and then *combine* these classifiers.

In *bagging*, [Breiman, ML, 1996] one selects *N* documents at random from the training set "with replacement," where *N* is the size of the training set. The phrase "with replacement" means that after each document is taken, it is (in effect) put back, so that all the documents of the training set are available the next time a document is selected. In other words, each document is taken from the full, original training set. Since *N* documents are selected, the "new" training set will be exactly the same size as the original training set. However, since each of the *N* documents is chosen at random from the original set, some documents may be chosen more than once, while others may not be chosen at all. Hence, the new training set will be different from the original. This procedure can be repeated as many times as desired, to produce a set of training sets, each of size *N*. Each training set is chosen independently of the others, so that the order in which the training sets are chosen, or used for training, is immaterial. Each training set is then used to train a classifier, using the same IR or ML method. Hence, a set of classifiers (commonly called an "*ensemble*" in this context) is generated. Each classifier is then executed against any new document, and their results are combined. A common method of combination is "voting," i.e., if the classifiers have been trained to determine whether a new document *d* belongs to class *C* or not, the choice is "yes" if more classifiers choose *C* than choose "not *C*." Another common method is to average the classification scores produced by all the classifiers in the ensemble. (Of course, *C* may be relevance to a given topic *T*. On the other hand, the classifiers may be trained to select among multiple classes, C_1, C_2, etc.)

Breiman [TR, 1996] argues that the main effect of bagging is to reduce classification error due to *variance*. This is the degree to which the classification estimate varies with the data the classifier is being asked to classify. [Witten et al., DM] [Opitz et al., 1999] [Friedman, DM&KD] In other words, it is a measure of how dependent the classifier is on the particular training set chosen, which may be unrepresentative of the larger population the classifier may be required to judge.(This overdependence on the training set is called "overfitting.") Opitz et al. [1999] following Bauer and Kohavi [1999], argue that bagging also reduces *bias* error, the *average* difference between the output of the classifier and the output of the "target" function the classifier is trying to learn.

Boosting, in contrast to bagging, generates a *series* of classifiers, ordered in the sense that each classifier is generated based on the performance of earlier classifiers in the series. [Opitz et al., 1999] In a powerful version of boosting called Ada-Boosting [Schapire et al., 1998], each classifier is trained on the same training set, using the same IR algorithm. However, the documents in the training set are weighted, and the weights assigned to the training set for generating classifier CL_i are based on the performance of the previous classifier CL_{i-1}. Specifically, after each classifier, CL_{i-1}, is executed, the documents in the training set are reweighted so that the weights of the

documents that it misclassified are increased, and the weights of the documents it classified correctly are decreased. This new weight vector and the associated training set are the input for training CL_i. Hence, it is hoped, CL_i will be better at classifying the documents that were previously misclassified. This process is repeated for T iterations, resulting in T classifiers, CL_1 to CL_T. (T is chosen by an ad hoc rule.) A weight w_i **i**s assigned to each classifier CL_i. The final "boosted" classifier for classifying new documents is a weighted *vote* of these T classifiers. That is, for a given new document d, each classifier CL_i votes +1 if it classifies d as relevant, -1 if it classifies d as non-relevant. The vote of each classifier CL_i is multiplied by its weight, w_i. These T weighted votes are then summed. The classification of d is relevant if the sum is positive, non-relevant if the sum is negative. Note that if the weights are all equal (which they usually are not), this is equivalent to classification by majority vote.

The underlying IR algorithm, which may be any simple algorithm chosen by the developer, is called a "weak learner." The goal of boosting is to combine a set of weak learners into a single "strong learner." (The terms "weak learner" and "strong learner" have technical definitions. [Breiman, TR, 1996]) The weak learner that Schapire et al. use is the presence or absence of a term in a given document. If the term is present, the document is assumed to be relevant, i.e., belongs to the class for which the classifier is being developed. If the term is absent, the document is assumed to be non-relevant. The algorithm "learns" from the training set at stage i by choosing the term t that minimizes the misclassification error, $err_i(t)$. The error $err_i(t)$ is defined as the sum of the weights of documents in the training set that either contain the term t but are non-relevant, or do not contain t but are relevant. Schapire et al. define a "term" to be either a single word, or a bigram, i.e., two consecutive words. Actually, the learner doesn't always choose the t that minimizes $err_i(t)$. Rather, it chooses the optimum term $t=t_{opt}$ that minimizes either $err_i(t)$ or $1\text{-}err_i(t)$, because a term with a very *high* misclassification error is distinguishing relevant from non-relevant documents just as well as a term with a very low error; it is just getting its classification decisions reversed, consistently calling relevant documents non-relevant, and non-relevant documents relevant. (By contrast, a term with a misclassification error close to 1/2 is very poor at distinguishing relevant from non-relevant documents.) Hence, classifier CL_i follows the simple rule that a new document is relevant if it contains t_{opt} and non-relevant if it does not contain t_{opt}.

The document weights are maintained as a probability distribution over the training set, i.e., the sum of the weights always equals one. (Hence, the misclassification error may be interpreted as a probability of misclassification.) So initially, each document in the training set is given a weight of $1/N$ where N is the number of documents in the training set. Thereafter, each time the documents are reweighted, the weights are also normalized so that their sum remains equal to one. The documents are reweighted at stage i, based on whether they were correctly classified or misclassified by the i-th classifier, CL_i, which learned using the i-th set of weights. Each document that was correctly classified by CL_i has its weight multiplied by e^{-ai}, and each document that was *mis*classified by CL_i is multiplied by e^{ai}, where ai is defined as $(1/2)\ln((1\text{-}e_i)/e_i)$. The effect is that if the error e_i is minimized, the smaller it is the larger ai is, and correspondingly, the more drastically document weights are modified, the weights of correctly classified documents going down, and the weights of misclassified documents going up. Hence, the next classifier generated, CL_{i+1}, will be tend to be better at classifying the documents that were misclassified by CL_i. On the other hand, if $1\text{-}e_i$ is minimized, e_i is maximized. If e_i is > 1/2, ai will be negative. The closer e_i is to one, the more negative ai will be, and correspondingly, the more drastically document weights

will be modified in the opposite direction, weights of correctly classified documents going up, and weights of misclassified documents going down. (In other words, if the term is present in a document, its weight will go down if the document is, in fact, non-relevant.)

A explained above, after T iterations, the resulting T classifiers, CL_1 to CL_T are combined by weighted voting. Each classifier CL_i is multiplied by a weight wi. This weight w_i = ai as defined in the previous paragraph. Since ai is a very large positive number for very low classification errors, i.e., for very good classifiers, a very large negative number for classification errors close to one, i.e., for classifiers that misclassify consistently, and close to zero for classifiers that don't do anything consistently, the effect is to weight classifiers by their effectiveness.

How many iterations *T*, i.e., how many classifiers, does boosting require? Schapire et al. have no theoretical basis for choosing the value of *T*, so they use a simple empirical rule: Iterate until the training error reaches a minimum (which may be zero). Call this number of iterations, T_0. Then run another 10%, i.e., run (1.1,)T_0 iterations, generating (1.1,)T_0 classifiers. Note that since each classifier distinguishes relevance from non-relevance on the basis of a single term, the effect of generating (1.1,)T_0 classifiers is to create a final weighted vote classifier based on (1.1,)T_0 terms, (1.1,)T_0 document features. In general, the harder the classification "problem," i.e., the harder it is to learn to recognize the target class, the greater T0 will be, and hence the greater the number of features in the final classifier.

The boosting scheme described above makes use of document weights, and (at the final stage) classifier weights, but it does not make use of term weights, as most traditional IR algorithms do. Schapire et al. say that they are studying ways of incorporating term weights into their boosting algorithm.

One easily remedied problem with boosting as described above is that it gives equal credit to classifying relevant and non-relevant documents correctly. In practice, it is often more important to recognize relevant documents. For example, if there are very few relevant documents, a "dumb" classifier that classifies every document as non-relevant will exhibit a very low classification error, although it will certainly not be very useful! Hence, it is desirable to tell the boosting algorithm that correct classification of relevant documents is more useful than correct classification of non-relevant documents. Schapire et al. accomplish this very simply by modifying the initial distribution of weights. Specifically, instead of giving each document an initial weight of $1/N$, each relevant document is given a weight of $(u_{rel+}\text{-}u_{rel-})/Z_0$, and each non-relevant document is given a weight of $(u_{nrel-}\text{-}u_{nrel+})/Z_0$, where u_{rel+} is the utility of classifying a relevant document correctly, and u_{rel-} is the utility (in this case, the harm) of misclassifying a relevant document. Similarly, u_{nrel-} is the utility of correctly classifying a non-relevant document, and u_{nrel+} is the utility (i.e., harm) of misclassifying a non-relevant document. Z_0 is a normalization factor, set so that the sum of all the initial weights is one as usual. By adjusting these four initial weights, the relative utility of correctly classifying relevant and non-relevant documents can be set as desired.

Breiman [TR, 1996] characterizes the family of boosting algorithms, including Ada-Boosting, as Adaptive Resampling and Combining or *arcing* algorithms. It has already been observed that both boosting and bagging involve *resampling* (to provide multiple training sets from the original set) and *combining* of the classifiers generated from these multiple sets, e.g., by averaging or weighted

voting. The term “adaptive” refers to the fact that in boosting algorithms, each classifier learns from the performance of previous classifiers. Breiman hypothesized that the effectiveness of arcing came from use of adaptive sampling, and *not* from the particular reweighting function employed. To demonstrate this, he experimented with other reweighting approaches. He found that a reweighting method which he dubbed arc-x4 worked as well as Ada-boosting (which he dubbed arc-fs (in honor of the original developer, Freund and Schapire).

In arc-x4, the reweighting of a given document d_j at the i-th stage (to generate the weights that will be used for training classifier CL_{i+1}) depends on the number of times d_j has been misclassified by *all* the i classifiers generated up to that point. Specifically, the weight of document d_j (or equivalently the probability p_j that d_j will be selected for the training set of classifier CL_{i+1}) is given by:

$$p_j = \frac{1 + m_j^4}{\sum_{k=1}^{N} (1 + m_k^4)}$$

where m_j is the number of times d_j has been misclassified by all of the previous i classifiers, and N is the number of documents in the training set. After T iterations, the resulting T classifiers are combined by *unweighted* voting.

13. User Interaction

Users interact with IR engines in many ways. They formulate queries or routing requests. They review the results (if any) returned by the engines. They refine their original requests. They generate “profiles” reflecting their interests and preferences. They build training sets, and train IR engines to classify documents. They set parameters to guide the engines, e.g., retrieval thresholds, cluster sizes or numbers.

Much of this material is covered elsewhere in this report, e.g., relevance feedback for query refinement, high-speed clustering methods for interactive clustering, the variety of query capabilities provided by Web IR and research engines, etc. But much of this discussion is conducted from the perspective of the IR engine, and its developer. Here, we will consider user interaction from the point of view of the user, and the researchers who are trying to make the user’s interactions more convenient and effective.

13.1 Displaying and Searching Retrieved Document Sets

Most IR engines return retrieved data in the form of a list of documents, ranked according to similarity to the topic or query for which they were retrieved, or the probability of relevance to the given topic/query. To make it easier for the user to scan this list, it is normally presented as a list of document surrogates, i.e., each document is represented by its title, or a short summary, each perhaps associated with its computed similarity or probability of relevance.

However, the number of retrieved documents may be very large, especially when the document collection is the huge number of pages comprising the Web, and all the databases to which Web pages act as gateways. Moreover, in many cases, the precision is low, due to the limitations of the existing technology on which IR engines are based, and the inexperience of human users. Hence, the relevant documents retrieved (if any) may be far down on the list returned to the user. Some systems may give the user the ability to limit the number of documents returned, either by setting a similarity/probability threshold, or by specifying the maximum number of documents to be returned. However, limiting the number of documents returned won't improve the precision; if the precision is low, an arbitrary cutoff point may simply prevent the system from returning the relevant documents the user wants to see.

Another problem with a simple ranked list is that it gives few clues to which documents are closely related. These relationships depend, in general, on many attributes, e.g., many document terms, as well as external attributes such as author, date of publication, etc. In other words, a ranked list only represents one dimension. The user would like to see documents positioned according to many dimensions. (Of course, this applies equally to the original collection against which the query was executed.)

An alternative is to organize the retrieved document set so that the user gets the "big picture" quickly and visually, and can zero in rapidly on the desired documents, regardless of how far down the ranked list they are. The big picture also enables the user to see which documents are closely related.

Veerasamy et al. [SIGIR '96] [SIGIR '97] display the retrieval set as a matrix. The rows correspond to key words from the query. The columns correspond to retrieved documents, ordered by rank, i.e., the leftmost column corresponds to the highest ranking document, the 2nd column corresponds to the document in rank 2, etc. The elements of the matrix are small vertical bars; the height of the bar for query word (row) *i*, and retrieved document (rank) j, is the weight of word *i* in document j. The effect is that "one gets an immediate idea of how the different query words influence the document ranking." One can see immediately which query terms are well-represented in high-ranking documents, and which are not. This may lead the user to modify the query by adding new words, dropping ineffective words, re-weighting query terms, etc. If two terms are closely related to each other and to the intended topic in the user's mind, but exhibit low and dissimilar distributions in the retrieval set, this becomes immediately obvious to the user; she may be able to improve retrieval and ranking, by modifying the query to specify the words as a phrase, or specifying that they must satisfy a proximity condition. In this way, documents in which the key words co-occur in close proximity are favored, and receive higher ranking.

In their SIGIR '97 paper, Veerasamy et al. describe a carefully controlled experiment to measure the effectiveness of their retrieval set visualization technique. The nature of the experiment, and the measures defined for interactive measurement are as interesting and significant as the results themselves. And the experimental results tell us as much about the human task of making relevance judgments as about the value of the visualization tool. They used a portion of the TREC data, and ten TREC information topics (queries). Veerasamy et al. used the INQUERY 2.1p3 engine as the common IR engine. They controlled for precision, on the assumption that the task of recognizing a relevant document is significantly different (and harder!) than the task of recogniz-

ing a non-relevant document. Hence, each user was given, for each topic, a high-precision and a low precision document set. (The high precision set was the first 60 documents retrieved by INQUERY. The low precision set consisted of the documents ranked 90 to 150 by INQUERY.) To control for the effect of the visualization display, the high precision set was further divided into an even-ranked set (ranks, 2, 4, etc.) of 30 documents, presented to the user with the visualization tool available, and an odd ranked set (ranks 1,3, etc.) of documents, presented to the user without the visualization tool. A similar division was made in the low precision set. For a given topic, each user was tasked to judge the relevance of the documents in each of four sets: high-precision even rank, high-precision odd rank, low-precision even rank, and low-precision odd rank. They were told that they were testing the effectiveness of the visualization tool. They were *not* told that all four sets came from the same collection, and *not* told that some were high precision, some low precision. Finally, they were given a monetary incentive to judge relevant accurately, and to judge quickly; the users who ranked best in a score that measured both accuracy and speed in completing a task received a sum of money.

Veerasamy et al. defined several measures of human interactive effectiveness: *Interactive precision* is defined as the proportion of documents judged relevant by the user that were also judged relevant by the TREC judges. *Interactive recall* is defined as the ratio of documents judged relevant by the user to documents judged relevant by the TREC judges. *Accuracy* is defined as the number of correct relevance judgments minus the number of incorrect judgments. Correctness means agreement with the judgment of the TREC judges, both with regard to relevance and non-relevance. Hence, in all cases the judgment of the TREC judges was treated as absolute "truth." So, the difference between interactive precision and recall, and their more traditional counterparts is that the interactive versions measure a *user's* relevance judgments rather than an IR system's judgments.

The Veerasamy experiment showed that users can "identify document relevance *more accurately* with the visualization tool than without." The effect of the visualization tool on *accuracy* is about the same for high precision and low precision document sets. The visualization tool also improves the time required to judge relevance (about 20% improvement), but this effect is much more pronounced for low precision sets than for high precision sets. Finally, the experiment showed that the visualization tool produced a significant improvement in *interactive recall* (and in the speed of identifying relevant documents as well), but only a minimal improvement in *interactive precision*. However, users achieve a much higher absolute *accuracy* for low precision document sets than for high precision document sets independently of whether they use the visualization tool, showing that their ability "to identify non-relevant documents as non-relevant is much higher than their ability to identify relevant documents as relevant." In other words, non-relevant documents are easy to recognize, while it takes extra effort to identify a document as relevant. This effect is much stronger than the influence of the visualization tool.

Note by the way, the interplay of *interactive recall*, *interactive precision*, and *accuracy.* The improvement in *interactive recall* means that the visualization tool is helping users to correctly recognize a higher proportion of the actual relevant documents. The corresponding minimal improvement in *interactive precision* means that the improvement increase in relevant documents identified is counterbalanced by a proportionate increase in non-relevant documents falsely identified as relevant. Hence, the concurrent improvement in *accuracy* must mean that the visualization

tool substantially helps the users in correctly classifying documents as non-relevant, classifying more non-relevant documents correctly as non-relevant, and fewer relevant documents incorrectly as non-relevant.

Hearst's *TileBar* display paradigm [ACM SIGCHI, 1995] may be compared and contrasted with the Veerasamy approach. In Hearst's display, each row corresponds to a retrieved document. Each document is represented as a series of adjacent non-overlapping segments called *tiles*. The order in which tiles are displayed in a row is the order in which they occur in a document. As explained in the section on query-document similarity, tiles are multi-paragraph segments such that each tile is about some sub-topic of the document, and the boundary between successive tiles represents a change in topic, as measured by the *tf*idf* similarity measure. In the tilebar display, each tile is represented as a small rectangle. Its shading on a grey scale from white to black represents the sum of the frequencies of all the query terms, white representing the complete absence of the query terms, and black representing a heavy concentration of the query terms. Hence, the user can see at a glance whether a given document is largely about the given query (much black throughout the row), whether it has passages relevant to the given topic (isolated black sections separated by much white), whether it has passages that may be about the given topic (grey sections), and so on. The user can see not only how much of the document is relevant, but also where the relevant passages are, e.g., at the beginning of the document, in the middle, etc. Similarly, the user can see at a glance which of the set of documents displayed are most likely to contain relevant passages, or to be largely about the given query.

As a further refinement, the user can see the document set displayed relative to several sets of query terms. For example (the example is Hearst's), the user may be interested in "computer-aided medical diagnosis." She may supply three sets of query terms, one set relating to medicine and patients, a second related to tests and diagnoses, and a third related to computer software. The *TileBars* display for a given document will show three rectangles for each tile, arranged vertically one above another. The degrees of shading of the rectangles for a given tile immediately tell us how much the tile is about each of the sub-queries. If all three rectangles for a given tile are black or dark grey, there is a good chance that the corresponding passage is about all three of the specified sub-topics. On the other hand, if the dark rectangles for one sub-topic are in completely different tiles from the dark rectangles for another sub-topic, then the document is less likely to be relevant to the user's topic, although it might score high on a conventional document similarity ranking. For example, the document might discuss both software and medical diagnosis, but the references to software might have nothing to do with its application to medical diagnosis.

Note that the Hearst display, unlike the Veerasamy display, is *not* a term-by-document matrix, or even a tile-by-document matrix. Indeed, a tile "column" would be meaningless, since each document is composed of its own unique set of tiles. But documents, i.e. rows, can be compared with respect to the distribution and shading of their respective tiles. Moreover, the display of each document in Hearst's display represents not merely term occurrence as with Veerasamy, but local term co-occurrence within the document's tiles.

Both Veerasamy and Hearst give the user a visual display of each of a set of individual documents. The user can study the properties of an individual document, or compare documents within a set. By contrast, another way to give the user an overview of the retrieval set is to *cluster* the

documents. Instead of seeing a (perhaps very long) list high-ranking documents, the user sees a modest number of document *sets*, each set clustered by some measure of content similarity that one hopes corresponds to topic similarity. Each group is identified by key-words, phrases, or other labels, that (again, one hopes) tell the user what topic(s) each cluster is about.

Clustering a very large retrieval set retrieved from an even larger set such as the Web imposes certain requirements. First, pre-processing (which can serve to speed up clustering - see the section on clustering above) is impossible. The original collection, e.g., the internet, is far too large (and dynamic) to pre-cluster. The retrieval set itself also cannot be pre-processed because its content is not known until the IR engine executes a user query. Second, the clustering must be fast; specifically, it must not add substantially to the time required for retrieval by the IR engine, which for Web retrievals is typically a matter of seconds.If clustering adds minutes or hours, the additional time would usually far outweigh the benefit of clustering. Third, the cluster labels should enable the user to pick out the best cluster(s) very rapidly. Finally, selecting the best cluster(s) should substantially improve the precision of the user's search, i.e., the effective precision if the user examines the documents in the best cluster(s) first, should be much better than if the user merely searched down the ranked list returned by the IR engine.

STC clustering, described above in the section on clustering, has been developed with just those requirements in mind. It is linear-time, not as good as the constant-time and almost-constant-time methods described in the section on clustering, but probably as good as can be achieved without pre-processing. Moreover, it is *incremental*, which means that clustering can proceed while the data is being retrieved and documents are being returned to the user. By the time the last document in the retrieval set arrives, the clustering can therefore be nearly done. (This assumes of course, that the documents can be clustered as fast as they arrive at the user's site, which is in fact the case, in the Web retrieval test reported by the STC developers, Zamir et al. [SIGIR '98] No actual test with real users was conducted in the reported research, so it remains to be determined how effective the cluster labels (strings of consecutive words shared by the documents in a cluster) prove to be identifying cluster topics and topic relevance to a real user. However, intuitively, strings of words should prove more informative than individual key words, and could always be supplemented with titles (where applicable) and statistically derived terms. In any case, the STC study made use of the experience of other researchers, who did provide document clusters to real hunan users. These experiences indicated that a user could select the "best" cluster first about 80% of the time. Hence, the STC researchers calculated precision on the assumption that the user was able to rank the topic clusters by number of relevant documents. On this assumption, they compared STC with several other linear time clustering methods, and one classic $O(N^2)$ method. STC was the clear winner. However, it should be stressed that this was a comparison of cluster quality, i.e., the best STC clusters contained more relevant documents than the best clusters produced by the other methods, *not* a comparison of the user ability to select the best clusters. It should also be noted that the queries employed by the researchers were generated by the researchers themselves, the queries were executed via real Web engines against the actual Web, and relevance judgments were assigned to the retrieved data by the researchers. Thus, the queries and test data were not a very large standardized test set such as the TREC data so widely employed in IR research,

13.2 Browsing a Document Collection

The term “browsing” implies that the user starts searching the data without a clear-cut end goal in mind, without clear-cut knowledge of what data is available, and very likely, without clear-cut knowledge of how the data is organized. She may have a rough goal in mind or perhaps no goal at all, or many possible goals. If she has a rough goal at all, it isn’t clearly defined enough to be formulated as a query. She searches the data as fancy takes her, formulating and modifying goals, as she encounters data or categories of interest.

The browsing method depends, in the first place, on whether or not the collection to be browsed has been manually indexed, i.e., whether or not human indexers have assigned subject categories to each document. A very popular example of manual indexing is the Web service, Yahoo (discussed below in the section on Web IR engines). However, much IR research has been devoted to accessing collections that have not been manually indexed. Let us consider first some browsing techniques that can be applied to collections that are only indexed by IR engines.

The *information space* browsing paradigm allows the user to visualize a vector space (such as the spaces discussed in an earlier section), and move around freely in that space (or what comes to the same thing, to manipulate the space itself). The human user is accustomed to moving around in a three-dimensional space in the real world. She is also accustomed to moving a cursor around in the two-dimensional space of computer monitor screen, using a device such as a mouse. However, to apply the “movement in space” metaphor to IR browsing, several problems must be surmounted.

The most obvious difficulty is that the number of dimensions in a typical IR vector space is much greater than two or three. Even with dimension reduction techniques such as LSI, the number of dimensions may be 50 to 200, far more than a human can readily visualize. Hence, a number of key dimensions, e.g., especially important query or document terms must be selected. If more than three dimensions are selected, the additional dimensions must be mapped into visual characteristics other than spatial coordinates. Each document is located in space by its spatial coordinates, and represented by a visual object, often called a *glyph* or *icon*. The additional dimensions can be represented by such visual characteristics as color, shape, texture, degree of opacity, etc. [Ebert, CIKM’95] Note that while each of these characteristics can vary according to a linear scale, opacity can effectively act as a filter. That is, on an opacity scale, a glyph varies from opaque to transparent. But a transparent (or near transparent) object disappears from the screen; hence, it will be effectively filtered out.

Viewing a glyph-based information space, and browsing in such a space, is significantly enhanced by the Stereoscopic Field Analyzer (SFA) [Ebert et al., IEEE Graph, 1997]. It is trivial to represent two spatial dimensions on a two-dimensional computer monitor screen. Three dimensions can be represented by the use of perspective, and manipulated, e.g., rotated and translated, via mouse control. SFA improves on these techniques in three ways. First, it provides a true 3-D stereo effect, by rendering the information space twice, once for each eye, and viewing the space through Liquid Crystal Shutter Glasses. Second, the user is given a tracking control, equipped with buttons. By moving this control with her hand in actual physical space as she sits in front of the monitor, and pressing the buttons to grip and release the information space, the user can manipulate the entire 3-D space, both rotating it in 3 dimensions so that the space may be effectively viewed from any direction, and translating it, i.e., moving the entire space up, down, left,

right, away from the user, and toward the user (the latter two movements corresponding to zooming out and zooming in). Third, the user is given another hand control to be manipulated with her other hand. This control can be used for finer manipulation, such as sweeping out a section of the space for closer examination, or pointing at a particular glyph (which may represent a single document or a cluster of documents).

One inherent limitation of the SFA/Information Space approach is that only three dimensions can be manipulated and browsed directly with the two manual controls. This is not merely a limitation of SFA. It is also a limitation of the human perceptual capability. We live and perceive in a three-dimensional world. However, SFA provides flexibility by permitting the user to specify which of the document attributes are to be mapped into each of the three spatial dimensions. The user can also specify what range of attribute values is to be mapped into the corresponding axis of the information space display. This capability can be used for such purposes as filtering out uninteresting ranges, e.g., a large cluster of documents near the origin of coordinates. Other attributes can, of course, be mapped into other visual cues as noted above: color, shape, texture, etc. These dimensions can not be manipulated with the 3-D tracker, but could be controlled separately, e.g., by graphical sliders.

A completely different approach to browsing, the “scatter/gather” method [Cutting et al., SIGIR ‘92], is based on a different metaphor, that of alternating between consulting the table of contents of a book (to get an overview of what is available), and consulting its index (to find the page or section dealing with a specific, narrow topic). The “table of contents” is generated (conceptually) by clustering the document collection. The labels or summaries that identify each cluster form the table of contents. The hope is that the documents that cluster together will be about a common topic, and that the label will identify that topic (or topics) to the user. This is called the “scatter” phase, because the documents, initially comprising a single collection, are *scattered* into multiple clusters. Then, the user scans the cluster labels (the “table of contents”) and selects the cluster(s) that interest her most. This selection process is the “gather” phase, because the user is *gathering* the selected clusters together into one document collection, a subset of the original collection. Next, the system clusters (scatters) again, but this time the clustering is applied to the subset collection. Hence, the clustering will be finer-grain, identifying sub-topics within (and perhaps across) the topics selected by the user. Hence, a new finer-grain table of contents is produced. Once again, the user selects (gathers) clusters (topics) of particular interest. The user repeats this scatter/gather process until she has narrowed her focus down to one or more specific topics for which she wants to read or scan the actual documents. Or perhaps, summaries, or abstracts, or the cluster labels themselves, at a fine level of detail, are sufficient to tell her what she wants to know. At any stage in this *scatter/gather* sequence, the user can employ an alternative focused search strategy, e.g., a key-word or boolean query to select particular documents from a cluster representing a topic of interest to the user. This corresponds to looking up a specific term or narrow topic in an index, the second part of the metaphor of alternating table of contents overview and index lookup.

At any level of detail, the user can back up to a higher level, and select different topics to pursue, initiating a new gather/scatter/gather sequence.

Various techniques can be used to generate labels for a cluster. Cutting et al. use a *cluster digest*. The digest of a given cluster *C* consists of the *m* most central documents in *C*, and the *w* most central words in *C*. The most central documents are those most similar to the cluster centroid (which they call the cluster *profile*). The highest weighted terms can be selected either from the cluster profile, or from the profile of its most central documents. The centroid (or profile) of the cluster is the normalized sum of the term vectors describing the documents of which the cluster is composed. Cosine similarity is used to compute the similarity of a document in the cluster to its profile. Term weight for a given term in a given document is computed as the square root of the term frequency.

Since *scatter/gather* requires "on the fly" re-clustering (scattering) of clusters selected (gathered) interactively, a rapid clustering method and algorithm is essential. Cutting et al. use *buckshot* and *fractionation*, two linear time clustering methods described above, in the section on heuristic clustering. These algorithms are used to find cluster centers rapidly. Each document is assigned to the closest center. Then, various refinement techniques are applied, e.g., once each document has been assigned to a center, the centers can be re-computed, and then each document can once again be assigned to the closest center. This process can be repeated indefinitely. Other refinements include splitting clusters that fail some simple coherency criterion, and joining clusters that have a sufficient number of *topic* (highly weighted) words in common. Finally, since even a linear time clustering method can be too slow for interactive clustering if the collection to be clustered is large, they use computationally expensive pre-processing, specifically the computation in advance of a cluster hierarchy before runtime, to achieve constant-time clustering [Cutting et al., SIGIR '93] during an interactive scatter/gather session. This constant-time method is discussed in the section on heuristic clustering above.

13.3 Interactive Directed Searching of a Collection

In contrast to *browsing*, "directed" searching means that the user has a specific information need. (This is the usual assumption of both ad hoc querying and routing.) "Interactive" directed searching means, of course, that instead of merely formulating and kicking off a single query and examining the results returned by the IR engine, the user engages in an interactive process, either to formulate the original query, or to refine the query on the basis of the initial results returned.

Relevance feedback, discussed above in the sections on Query Expansion and Query Refinement, is the classic method of improving a query interactively. Here, a variation of relevance feedback, and the use of clustering for query refinement are discussed.

Aalbersberg [SIGIR '92] proposes a simplified form of interactive relevance feedback that he calls "incremental relevance feedback." Most IR systems that support relevance feedback perform the reformulation of the query automatically, concealing the mechanics and often the reformulated query from the user. However, in conventional systems, each time the user executes the (original or reformulated) query, she sees a set of *N* retrieved documents, typically 10 to 20, from which she must select those she judges relevant. After she has judged the *N* documents for relevance, she requests an automatic reformulation of the query, and execution of the reformulated query. In Aalbersberg's system, the user is not aware of her query being reformulated at all. The user sees one document at a time. She designates that document as relevant or not. If it is relevant,

its title is added to a “Results” window of relevant documents. She then sees another document, and again judges its relevance, and so on. After the user has viewed and judged *N* documents, the titles of that fraction N_R that have been judged relevant are in the *Results* window. Hence, she has the sense that she is merely judging a series of documents that the system believes may be relevant to her original query (or the information need that query is intended to represent). However in actual fact, the first document she sees is the highest ranking document in the list returned after executing the original query. Thereafter, each time the she judges the relevance of the current “best” document, this judgment is immediately used to reformulate the current query. This reformulated query is immediately executed, invisibly to the user. The next document submitted to the user for relevance judgment is the highest ranking document returned for the reformulated query.

Aalbersberg uses the Rocchio formula for query reformulation. However, this formula takes on a very simple form in Aalbersberg’s system, since each stage involves modifying the query vector by either adding a single document vector (if the current document is judged relevant), or subtracting a single document vector (if the current document is judged non-relevant). In either case, the document vector represents the single document the user has just judged, multiplied by the appropriate constant (*B* or *C*) from the Rocchio formula.

Above, the use of clustering for browsing a document collection, the so-called *scatter/gather* method, was discussed. Earlier, in the section on clustering, the use of hierarchical clustering for a directed search was discussed. By drilling down through the hierarchy, the user can focus on the small number of documents, in a cluster at the lowest level of the hierarchy, about the topic that concerns him. Roussinov et al. [SIGIR ‘99] suggest another interactive use of clustering: to help the user refine and reformulate his query.

The scheme is to submit a simple natural language query to a Web IR engine (Roussinov et al. use Alta Vista). Their system fetches the 200 highest ranking documents from the list returned by the IR engine. These documents are automatically clustered, using an unsupervised clustering technique. (Roussinov et al. use the self-organizing map technique. Another, possibly better technique would be the STC clustering method discussed earlier in the section on Incremental Clustering. The essential characteristics of the clustering method is that it must be unsupervised, and that it must generate labels for each cluster that can aid the user in rapidly identifying the content of a given cluster. Speed is another important characteristic of a clustering method for on-line clustering of retrieved results. However, Roussinov et al. are only clustering the top 200 documents, so they don’t need as fast a method as Zamir et al. who use STC to cluster a much larger retrieval set.) The system then displays for the user the cluster labels and “representative terms associated with each cluster.” The user selects from this display those labels and terms that seem relevant to his original query (or to the current information need the query was intended to express). The selected terms and labels may also suggest additional terms that belong in the query. He types these additional words or phrases. The system then uses the selected and typed terms and labels to create a set of new or reformulated queries, which it then submits to the IR engine. Multiple iterations of this process are supported.

14. IR Standard - Z39.50

Z39.50 is a national (ANSI/NISO) standard for information retrieval. Its two primary functions are search and retrieval. These functions are initiated by an entity called the Origin, which is contained in an application called the Client, residing on the Client system. The Origin communicates with the Target, which is contained in an application called the Server, residing on the Server system. The Server is the database provider. [ANSI/NISO Z39.50, 1995] "Unlike other Internet protocols such as HTTP or WAIS, Z39.50 is a session-oriented protocol. That means that a connection to a Z39.50 server [from a Z39.50 client] is made and a persistent session is started. The connection with the server is not closed until the session is completed." [LeVan, OCLC]

> Searching is the selection of database records, based on origin-specified criteria, and the creation by the target of a result-set representing the selected records. Retrieval, idiomatically speaking, is the transfer of result set records from the target to the origin [but see below]. [ANSI/NISO Z39.50, 1995].

Z39.50 started life as a standard of the library community, specifying a protocol for "search[ing] and retriev[ing] USMARC-formatted bibliographic records … However, the standard has grown considerably … Today, there are organizations using Z39.50 to deliver full-text documents based on natural language queries." [LeVan, OCLC]. Today, Z39.50 is evolving to meet the complex requirements of the IR community. To the extent that it satisfies these needs, and is supported by a wide variety of commercial and governmental IR providers, it will become (and appears on the way to becoming) the standard language for accessing IR engines. As such, it may play a role in the IR community analogous to the role of SQL in the structured DBMS community.

In comparing Z39.50 to SQL, one essential caveat is in order. As the preceding sections of this paper make abundantly clear, there is an uncertainty in Information Retrieval which is not present in retrieval from a DBMS. This is reflected in a basic difference between SQL and Z39.50. The SQL standard defines, at least in principle, the semantics as well as the syntax of the "black box" retrieval that is to be performed by a conforming DBMS in response to an SQL query. What does that mean? Suppose that the same data, i.e., the same tables (or "relations"), are loaded into three distinct DBMSs, implemented by three distinct DBMS vendors, but all conforming to the same level or version of SQL. Suppose further that an SQL query is formulated against this data (and data structure) and executed by each of the three DBMSs. Then as long as the query conforms to the level of SQL supported by the three vendors, and doesn't use any non-standard vendor-specific features, *exactly the same data should be returned by each DBMS*. The internal details of how the query is executed may vary considerably from one DBMS to another, depending on how its optimizer works, how the tables are indexed, and so on; correspondingly, the response time may vary substantially from one DBMS to another. But exactly the same data should ultimately be returned.

Plainly, this is not (and cannot possibly be) the case with Z39.50 (and will be even less the case as Z39.50 evolves to support more powerful and diverse IR engines and queries). The behavior of IR engines varies far too widely. A term vector submitted to a term-based vector space IR engine will produce a different result than the same vector submitted to an LSI-based vector space engine.

The result will be different again, if the IR engine is based on a probabilistic model, and will vary from one probabilistic model to another. An extended boolean query will produce different results depending on which boolean model it employs. (And different results than a strict boolean engine which must ignore clause weights!) Even two IR engines that use the same term-based vector space approach may differ if they employ different index weighting schemes, employ different stop lists, employ different query expansion schemes, employ different query/document similarity measures, etc.

So, it should be understood that what Z39.50 provides is a consistent way of talking to diverse IR engines. The results returned may vary widely depending on all of the factors mentioned above, and more.

14.1 Searching via Z39.50

The Z39.50 search process starts (as noted above) with the specification by the origin of search criteria. These criteria are specified by a Z39.50 query. The queries currently supported by Z39.50 are called Type-1 and Type-101. The functionality of a Type-1 query is described briefly in this section. (Type-101 is functionally identical to Type-1. The only difference is that the definition of Type-101 is independent of the version of Z39.50, i.e., it works with both Z39.50-1992 and Z39.50, 1995, known as version 2 and version 3 respectively.) Z39.50 also supports some other query types with grammars that are severely limited in extensibility, are not widely used, and are not mandatory in the standard; these other query types should probably not be used and are not discussed further here. [LeVan, 1995]

Z39.50 allows the sender to specify strict term-based boolean queries using the operators AND, OR, AND-NOT, and Prox. The latter is a proximity operator that tests whether two terms are (or are not) within a specified distance of each other, where the distance is measured in units and the possible choices for the unit include: Character, Word, Sentence, Paragraph, Section, Chapter, Document, Element, Subelement, ElementType, Byte or privately defined unit. The order of the terms may be specified. [ANSI/NISO Z39.50, 1995]

Each operand of the boolean query may consist of a term and a list of attributes that qualify the term. Attributes specify something about the semantics of the given term. The attributes are drawn from an “attribute set.” An attribute set specifies a “list of the types of things that can be searched for.” [LeVan, OCLC] A number of attribute sets have been defined, and other sets can be defined in the future. A query can draw attributes from more than one set. The core attribute set, reflecting the origins of Z39.50 in the library community, is called “bib-1.” The bib-1 attribute set has six types: Use, Relation, Position, Structure, Truncation, and Completeness. “The “Use” attribute allows the client to specify how the term would have been used in the records to be retrieved.” For example, the term might be used as a Title, as (the name of an) Author, etc. At present (in the 1995 version of the standard), 99 values of Use are defined. Most of them are clearly related to bibliographic reference, e.g., Dewey and other classification numbers, Date of Publication, etc. Some have more general applicability, e.g., Personal Name, Corporate Name, Conference Name. Name Geographic, etc.

The Structure attribute specifies the structure, e.g., WORD, PHRASE, DATE, NUMERIC STRING, FREE-FORM-TEXT, etc. The POSITION attribute specifies the position of the term in the structure, e.g., FIRST IN FIELD. (These attributes are actually represented in Z39.50 by numeric codes.) [ANSI/NISO Z39.50, 1995]

The end result of the Z39.50 search process is "the creation by the target of a result-set representing the selected records." These result set records are then transferred by the target to the source during the retrieval process.

Note that Z39.50 Type-1 queries are always structured, term-based queries. Z39.50 does not support unstructured queries, e.g., documents as queries. (But see the relevance feedback feature of the proposed Type 102 query discussed below.)

14.2 Retrieval via Z39.50

> [T]he "transfer of a result set record" more accurately means: the transfer of some subset of the information in a database record (represented by that result set entry) according to some specified format [called a retrieval record].
>
> Z39.50 retrieval supports the following basic capabilities:
>
> * The origin may request specific logical information elements from a record (via an element specification …).
>
> * The origin and target may share a name space for tagging elements … so that elements will be properly identified …
>
> * The origin may request an individual element according to a specific representation or format …
>
> * The origin may specify how the elements, collectively, are to be packaged into a retrieval record … [ANSI/NISO Z39.50, app. 14]

The structure of a retrieval record may be hierarchical, e.g., may include sub-fields, sub-sub-fields, etc. "An origin might request, for example, 'the fourth paragraph of section 3 of chapter 2 of book1.'" Or the retrieved data might be more conventional structured data, e.g., a product availability field may contain a "distributor" sub-field, which may, in turn, contain the sub-sub-fields for the name, organization, address, and phone number of the distributor.

14.3 Type 102 Ranked List Query (RLQ) - A Proposed Extension to Z39.50

A number of IR features discussed extensively in this paper are notably lacking from the Z39.50 query capability discussed above: extended boolean queries, weighting of terms or clauses, rank-

ing of the retrieved results, relevance feedback, etc. These “ranked searching technologies [are] used by the majority of large-scale commercial information providers and information industry software vendors. This includes 80-90% of the mainstream commercial ranked searching technologies …” The proposed type 102 Ranked List Query (RLQ) has been designed to meet these requirements. [Type 102, 1995] This query has been developed by the Z39.50 Implementor’s Group (ZIG), which includes such organizations as Chemical Abstracts Service, Clearinghouse for Networked Information Discovery & Retrieval (CNIDR), Excalibur Technologies Corp., Knight-Ridder Information Services, LEXIS-NEXIS, National Institute of Health (National Library of Medicine), and West Publishing Company.

A weight may be attached to each operand in a Type 102 query. A Type 102 query “is a recursively defined structure of operators and weighted operands.” Since the query is recursively defined, a clause, e.g., an operator and its associated operands, can itself be an operand, and hence can itself be weighted. The weight attached to an operand “specifies the value to be placed on the operand with respect to its importance in selecting records from the designated collection(s).” [Type 102, 1995]

Type 102 supports operators that may take more than two operands (as required by some extended boolean models).

Type 102 supports extended, i.e., relevance ranked, boolean operators (called “relevancy-based” operators in the type 102 spec). For example, instead of a strict boolean AND operator, there is an operator called “rqAND.” Each operator and its associated operands comprise a clause. A number may be attached to each clause. These numbers determine the degree to which the clause is to be given a strict or extended boolean interpretation. Note that some servers may ignore these numbers. These numbers should not be confused with term weights.

Type 102 supports the retrieval of ranked output, i.e., each result record is associated with a Retrieval Status Value (*RSV*) which is a measure of its degree of relevance to the given query. Moreover, the type 102 query allows the user to limit the number of records retrieved, either by specifying the number of records to be returned, e.g., the top-ranked *N* records, or by threshold value, e.g., all records with *RSV* above threshold value *RTHR*. Note that the interpretation of the *RSV* is server-dependent, e.g., it might be a cosine similarity in one system and a probability of relevance in another. Also note that while the result set will normally be ordered by *RSV*, other orderings, e.g., by date, can be requested.

A Type 102 query can be applied to one or more record collections. (The Type 102 spec uses the term “record” instead of “document.”) The query can restrict the collections to which the given query is to be applied, or specify particular collections to which it is *not* to be applied. (The Type 102 spec uses the terms “collection” and “database,” apparently interchangeably.)

A Type 102 query can specify the degree to which recall is to be emphasized (at the possible price of loss of precision).

A Type 102 query may specify whether the original query may be reformulated, e.g., expanded, by the retrieval engine. Moreover, the query may specify that only the reformulated query is to be

returned. Alternatively, the query may specify that only the retrieved records are to be returned. Or, both the retrieved records and the reformulated query may be requested.

A Type 102 query allows the user to specify relevance feedback info, either within the original query or within a resubmission of a reformulated query. The feedback info takes the form of a list of records with a relevance measure (in the range from -1 to +1) attached to each record. Notice that the negative numbers allow the user to specify the degree to which given records are undesirable.

The Type 102 query can request the return of demographic data pertaining to the collection being queried, or to the result set, or to the retrieved record, etc. The collection-level metadata that can be returned includes: number of records, number of unique terms (either including or excluding stopwords), total number of term occurrences, total number of records in which each term occurs, and total number of occurrences of each query term in the collection.

Type 102 supports proximity as does Type 101. However, in Type 102, there is no boolean proximity operator. Instead, a proximity condition (called a "qualifier") is attached to a boolean clause to indicate that all operands within that clause (structured operand) must be satisfied within the same proximity unit.

15. A Brief Review of some IR Systems

This section is a brief review of some of the leading commercial and research IR engines. The engines reviewed here are chosen to be representative, not exhaustive.

15.1 LEXIS/NEXIS

LEXIS/NEXIS is a commercial system for retrieving legal (LEXIS) or newspaper (NEXIS) documents.

LEXIS/NEXIS [qrel] supports traditional "strict" boolean queries, i.e., booleans that return exact matches only. Specifically, it supports queries formulated with the boolean operators (called "connectors" in LEXIS/NEXIS) OR, AND, and W/n. The latter is a proximity operator, e.g., *homeless W/5 shelter* specifies that *homeless* and *shelter* must occur within five words of each other. It also supports two "wild card" characters (called "universal" characters in LEXIS/NEXIS): The character ! specifies any suffix that can be added to the root word, e.g., "*transport!* finds *transportation, transporting, transported*, etc." The character * specifies any single character. It "must be filled in if its in the middle of a word, but not if it's at the end. (EXAMPLE: *wom*n* finds *woman, women; transport*** finds *transport, transports*, *transported*, but not *transportation*, etc.)"

More recently, LEXIS/NEXIS has followed the trend toward natural language queries (called "FREESTYLE™ search descriptions" in LEXIS/NEXIS); these queries do not require (or permit) boolean connectors. This is, in essence, the vector space approach described earlier in this paper. It "identifies significant terms and phrases from your search description, removes irrelevant words from your search description [e.g., applies stoplists, etc.], applies a formula that weighs the statis-

tical importance of the significant terms and phrases from your search description and compares them to the documents in the library and file(s) in which you are searching [e.g., weights the significant terms and computes the similarity of query to documents in the target collection] — the more uncommon or unique the word, the greater the statistical weight [e.g., uses *tf*idf* weighting or the like]."

LEXIS/NEXIS provides a number of ways of qualifying or enhancing a natural language query. The user can tell LEXIS/NEXIS to treat two or more consecutive words as a phrase by bracketing them with quotation marks. In addition, LEXIS/NEXIS itself will recognize certain word combinations as phrases, and put quotation marks around these combinations automatically; the user can override this feature by editing out the quotation marks. The user can specify that certain words or phrases are mandatory, i.e., they must appear in any retrieved document. (Note that such a feature is meaningless in a strict boolean query, since the boolean operators themselves determine whether a given term is mandatory, and under what conditions it is mandatory.) The user can specify "restrictions," i.e., constraints (other than mandatory words) that must be satisfied by retrieved documents; for example, a legal document may be constrained by date or court. The legal user may invoke an online thesaurus of legal terms. "A list of the terms in your search description for which synonyms [or alternative forms] are available will appear." The user has the option of displaying the synonyms and alternative forms for a given word, and adding any of these additional terms that she chooses. Hence, query expansion via thesaurus is manual, with the thesaurus providing online guidance, but the user deciding which terms (if any) to add. Finally, the user may specify how many documents to retrieve. (Again, note that this feature would be meaningless for a strict boolean query, since a strict boolean determines a set of documents that exactly satisfy the query; there is no notion of degree of relevance in a strict boolean retrieval.)

LEXIS/NEXIS provides some result display options that are only available (indeed, only applicable), to a natural language, i.e., vector space, query. For example, the user can display "the most heavily weighted block of text — the portion that most closely matches [her] search description." More interesting is the "WHY" option. This option "shows how your search was interpreted …,displaying the order in which your terms were ranked, the total number of retrieved documents with each of your terms, … and the importance assigned to each term." Note that this is closely related to the Z39.50/type 102 query features that allow you to look at the system expansion of your query, and the demographics of its terms.

15.2 Dialog

DIALOG [QT] is a commercial system for retrieving documents from databases in such topic areas as: Business, Intellectual property/Law/Government, Medicine and Pharmaceuticals, News, People, Sciences, Social Sciences & Reference, and Technologies.

The user selects a topic. Then she selects a database (or group of databases) within the topic. The search options vary with the database. For example, options for a newspaper database include: Subject (keyword), Title/Lead Paragraph, Author, Journal Name, Section/Subject Heading, and Limit options.

DIALOG supports a strict boolean query capability very similar to that of LEXIS/NEXIS, e.g., AND, OR, and w (proximity) operators. Like LEXIS/NEXIS, DIALOG supports a wild card character (?) that can only be used to specify any suffix to a common root, e.g., “smok?” will find smoke, smoker, smokers, smoking, etc. There is no thesaurus; it is up to the user to think of appropriate synonyms.

In “menu” mode, the user enters a term and any synonyms connected by “OR”. Then she can *modify* the query by either *broadening* it, i.e., adding additional terms implicitly connected by “OR”, or *narrowing* it, i.e., adding additional terms implicitly connected by “AND.”

In “command” mode, the user can generate nested boolean expressions. To make the expressions simpler to read and generate, the user generates terms, e.g., “smoking OR tobacco,” “heart disease OR heart attack”. Each term is assigned an id by DIALOG, e.g., the first term may be assigned the identifier “S1,” and the second term may be assigned the identifier “S2.” The user can then generate compound boolean expressions using these identifiers, e.g., “SELECT S1 AND S2.” DIALOG will now assign an identifier, e.g., S3, to the compound expression “S1 AND S2.” At each stage, DIALOG tells the user how many documents are retrieved, e.g., how many are retrieved by the term “smoking” by itself, how many are retrieved by “smok? OR tobacco,” how many are retrieved by “S1 and S2,” etc. In this way, the user can decide when he has limited his retrieval set sufficiently. At all stages, retrieval involves a strict boolean match.

DIALOG allows the user to save a query. Thereafter, if the query matches a new document that has been added to the given database, the user is alerted. The set of saved queries for a given user is called an “alert profile.”

15.3 Dow Jones News/Retrieval

Dow Jones News Retrieval [Dow QR] is a commercial system that can search up to 1900 news sources, e.g., newspapers, newsletters, news magazines, etc., general interest and specialized. As with the other commercial engines described here, it supports strict boolean queries with a somewhat broader set of operators, e.g., AND, OR, NOT, SAME, NEAR, etc. A query can be further restricted by specifying a date, categories and subjects, document sections, and specific sources, e.g., specific publications. The system displays a set of available subject and category codes; not all codes work in all publications. Similarly, not all document section types are available in all publications.

Retrieved documents can be sorted, highlighted, etc. One can retrieve a hit paragraph rather than an entire document. One can retrieve the headline and lead paragraph, or the full text of an article.

15.4 Topic

Topic [Topic Intro] is a commercial IR engine, marketed by Verity, Inc. In contrast to the three commercial IR services described above, Topic is not an IR service maintaining indexed document collections, but a stand-alone IR tool that can be used by any purchaser to provide IR ser-

vices. Verity also markets an application program interface to Topic, the Topic Development Kit (TDK). This allows Topic to be incorporated into application systems, and other vendor products.

Before a collection of documents can be searched by Topic, it must be loaded "into" Topic, a process that involves sophisticated indexing. In this respect, of course, it resembles most other IR search engines, whether commercial or research, as well as the information services described above.

The basic text search condition or query in Topic is called a "topic" (formerly called a "concept tree"). Topics are hierarchically structured. Each topic has a name which is the root of its "tree". Below the root are any number of child sub-topics, also named. A sub-topic may itself have any number of named sub-topics. Hence, there may be any number of levels of sub-topic. At the lowest level (the leaves of the tree) are "evidence topics" which specify the actual words or phrases for which Topic is to search each document in a given Topic collection. For example, the terms *ballet, drama, dance, opera, and symphony* may be evidence topics for a sub-topic named *performing-arts*. "[T]he sub-topics *performing-arts, film, visual-arts,* and *video* [may be] children of the *art* topic. The *art* [sub-]topic itself [may be] a child of the *liberal-arts* [sub-]topic." The *liberal-arts* topic might be the root topic. Alternatively, "[t]he *liberal-arts* topic could in turn be a child of successively higher parent topics within the [topic] structure." [Topic Intro]

Topic performs relevance ranking as described below. When a topic is executed against a given collection, it is evaluated against each document, and the document is assigned a score in the range 0.01 to 1.0. The higher the score, the better the document matches the topic (according to Topic's matching formula, of course). Documents are returned to the user in descending order of score.

An operator may be associated with each topic or sub-topic node. There are three classes of operator. The evidence operators specify the string or set of strings for which each document is to be searched. Hence, they only appear at the lowest level, i.e., below any of the other operators. For example, "WORD" specifies a word, actually a string of up to 128 alphanumeric characters, but usually an ordinary word or number, e.g., "microchip" or "80386." The STEM operator specifies the usual stemming, e.g., to stem "transport" is to search for "transports," "transported," "transporting," etc. (Note that stemming must be specified explicitly for each evidence word which means that the user can avoid stemming of a given word if he wishes. In contrast, some of the other systems discussed above and below performed stemming automatically.) The WILDCARD operator allows specifying of search patterns. It uses a richer set of wild-card characters than the commercial services described above, perhaps because the creator of a topic is assumed to be more sophisticated than the typical user of those IR services. Wildcard characters may occur anywhere in a search pattern, not just at the end, and support single characters (?), zero or more characters (*), any one of a specified set of characters, any character in a range (e.g., [A-F]), etc. The NOT operator may be used to exclude documents that contain a specified word or phrase.

Above the evidence operators in precedence are the proximity operators: PARAGRAPH, SENTENCE, PHRASE, NEAR and NEAR/N. Each of these operators specify two or more words that must satisfy the given proximity constraint. A proximity operator may be assigned to any sub-topic above the evidence level. PARAGRAPH and SENTENCE are self-explanatory. PHRASE

specifies a string of consecutive words, e.g., “arts and crafts”. NEAR/N specifies that the specified words must not be separated by more than N words. NEAR differs from NEAR/N in that “[d]ocument scores are calculated based on the proportion of instances found in relation to the size of the region containing the words … Thus, the document with the smallest region containing all search terms always receives the highest score.” An ORDER operator can be used with SENTENCE, PARAGRAPH, and NEAR/N to specify that the search terms must occur in a specified order.

Above the proximity operators in precedence are the “concept” operators, which are the boolean operators: AND, OR, and ACCRUE. The AND operator is a strict boolean, i.e., it only selects documents that contain all its children (operands). In other words, it contributes a score of zero for each document that does not contain all its children. However, if all its children *are* present, the score returned by AND is not simply 1.0 (as it would be for a conventional strict boolean) but will be the minimum of the scores of its children. OR is also a (kind of) strict boolean in the sense that it returns a score of zero only if all its children have scores of zero, e.g., if its children are words and phrases, at least one must be present in the document if the OR is to contribute a non-zero score to that document’s total score. Moreover, the score it contributes to a given document if any of its children are present does *not* depend on how many of the children are present. However, it is not 1.0 either; instead, it is the maximum score of any of its children. ACCRUE is a (kind of) extended boolean OR, i.e., the more of its children are present in a document, the higher the score that it contributes to that document. However, the score it returns is the maximum of its children’s scores (like an OR) plus a little extra for each child that is present.

A further degree of ranking can be specified by using the MANY operator in conjunction with an evidence or proximity operator to rank “documents based on the density of the search terms they use.” In other words, MANY normalizes term frequency by document length so that “a longer document that contains more occurrences of a word may score lower than a shorter document that contains fewer occurrences.”

Finally, the user may assign weights in the range of 0.01 to 1.0 to operators. Specifically each child of a logical operator (AND, OR, and ACCRUE) may be assigned a weight. Since a logical operator may be a child of another logical operator, the logical operators themselves may be assigned weights. Similarly, evidence operators (WORD, STEM, etc.,) may be assigned a weight. Proximity operators may not be assigned a weight. Weights determine the relative importance of search terms or higher level children. For example, the score for a given document contributed by a given AND operator is not merely the minimum score of any of its children but rather the product of that score and the weight assigned to the child having that minimum score.

Finally, there are a number of operators that only apply to structured fields of a document, e.g., title, subject, author, etc. These operators do not rank documents but filter them, e.g., one can specify only documents by a given author or only documents whose titles contain a given sequence of words.

15.5 SMART

The SMART system [Salton & McGill,1983], developed at Cornell, is the “granddaddy” of IR systems that (1) use fully automatic term indexing, (2) perform automatic hierarchical clustering of documents and calculation of cluster centroids, (3) perform query/document similarity calcula-

tions and rank documents by degree of similarity to the query, (4) represent documents and queries as weighted term vectors in a term-based vector space, (5) support automatic procedures for query enhancement based on relevance feedback. SMART has been widely used as a testbed for research into, e.g., improved methods of weighting and relevance feedback, and as a baseline for comparison with other IR methods.

Note that extended boolean retrieval, i.e., the *p*-norm method, was developed in the SMART "shop", although it does not appear to be formally incorporated into the SMART testbed.

All of the above topics have been discussed extensively above; the discussion need not be repeated here.

15.6 INQUERY

INQUERY [Callan et al., DB&ExSysApp, 1992] is a probabilistic IR research system, developed at the University of Massachusetts, and "designed for experiments with large [text] databases." INQUERY is based on the inference network model, which is discussed in an earlier section of this paper. Here, a brief overview of the INQUERY system will be given.

> [The inference net model] implemented in the INQUERY system emphasizes retrieval based on combination of evidence. Different text representations (such as words, phrases, paragraphs, or manually assigned keywords) and different versions of the query (such as natural language and Boolean) can be combined in a consistent probabilistic framework. [Callan et al., IP&M, 1995]

The INQUERY document parser analyzes the overall structure of the document, converts it to a canonical format, and identifies those sections to be indexed. Then, it performs lexical analysis to extract words, fields, etc., "recognizes stop-words, stems the words, and indexes the words for retrieval." Stop-words are not indexed but they are retained in the text so that subsequent textual analysis (syntactic analysis, feature recognition [see below]) may make use of them." [Callan et al., IP&M, 1995] (See discussion above of Riloff's work on the use of stop-words in semantic analysis.)

INQUERY feature recognizers (earlier called "concept recognizers") "search text for words that correspond to simple semantic components," e.g., numbers, dates, person names, company names, country names, U.S. cities, etc. The set of feature recognizers is open-ended. The number recognizer maps multiple forms of a number, e.g., 1 million, or 1000000, or 1,000,000, into a common, canonical format. The company name recognizer "looks for strings of capitalized words that end with one of the legal identifiers that often accompany company names (e.g., "Co," "Inc," "Ltd," …)." [Callan et al., DB&ExSysApp, 1992] In addition to such heuristics, databases, e.g., of known person or city names, are used.

"Queries can be made to INQUERY using either natural language, or a structured query language." Query pre-processing includes stop-phrase removal, stop-word removal, stemming, and

"conversion of hyphenated words and sequences of capitalized words into proximity constraints". [Callan et al, IP&M, 1995] (The latter corresponds to the fact that, e.g., hyphenated words, are indexed as separate words but with their textual positions retained.)

Queries are expanded using an INQUERY tool called PhraseFinder, which builds a database of "pseudo-documents" based on a given collection of actual documents.

> Each pseudo-document represents a *concept*, in this case a noun sequence, that occurs in the document collection. The "text" of the pseudo-document consists of words that occur near the concept in the document collection. For example, a PhraseFinder document for a *Wall Street Journal* collection contains an *amnesty program* pseudo-document indexed by *1986, act, control, immigrant, law* … A query is expanded by evaluating it against a PhraseFinder database, selecting the top ranked concepts, weighting them and adding them to the query. [Callan et al, IP&M, 1995]

Concepts are ranked by performing a conventional match between the given query and the collection of pseudo-documents. The concepts associated with the highest ranking pseudo-documents are added to the query. The assumption is that concepts, e.g., noun sequences, co-occurring with some of the same terms in both a query and a document context, e.g., a pseudo-document, may be related semantically. Hence, if "amnesty program" co-occurs with "immigrant" and "law" in both a query and a document, it probably refers to the same entity in both places. Evidently, PhraseFinder performance is sensitive to how near to a concept a term must be to be included in its "pseudo-document."

INQUERY 3.0 [INQ 3.0, ACSIOM, 1995] supports a number of structured operators. Operators can be nested within operators. Each of these operators returns a belief value, e.g., a score or weight, or a proximity list that can be converted into a belief list. The beliefs returned by the clauses of a structured query contribute to the belief that the given document satisfies the information need expressed by the total query in which these clauses occur. The primary boolean operators, #and and #or, are extended booleans. (By convention, all the INQUERY operators start with #.) For example, the interpretation of #and is that "[t]he more terms contained in the AND operator which are found in a document, the higher the belief value of that document." Plainly, extended booleans are more in tune than strict booleans with the probabilistic nature of INQUERY, since they return a degree of satisfaction of the boolean condition rather than an all-or-nothing true or false. There are also some strict boolean operators: #band, and #bandnot. (The latter is satisfied if the first term is in a given document and the second term is not.) There is also a #not command which awards higher belief to a document that does not contain its operand terms.

There are several proximity operators: The Unordered Window operator #uwN requires that its operand terms co-occur in the document in any order but within a window of *N* words. The Ordered Distance operator, #odN, is similar except that in addition to co-occurring within a window of *N* words, the terms must occur in the specified order within that window. The #phrase operator evaluates terms to determine if they occur together frequently in the collection. If they do [i.e., if the phrase occurs frequently in the collection *as* a phrase], then they are required to occur

in the specified order within a three word window, i.e., #phrase is evaluated as #od3. Otherwise, #phrase reduces to #sum (see below); the more operand terms a given document contains, regardless of their proximity, the higher belief (rank) the document receives. The #passage operator is similar to #uwN except that instead of looking for an *N*-word window in which all the specified terms occur, it looks for the "best" passage, i.e., the *N*-word window that most nearly satisfies the specified operands; "the document is rated based on the score of its best passage."

A synonym operator, #syn, allows the user to specify that its operands are to be treated as synonymous terms.

There are two sum operators: Sum (#sum) and Weighted Sum (#wsum). The former sums the beliefs of its operands. The latter takes a set of weight/operand pairs and computes the weighted sum, i.e., $W_1*T_1 + W_2*T_2 + \ldots + W_n*T_n$. (One can also specify a scaling factor, an overall weight, for the entire weighted sum.) A weighted sum allows the user to say that some operands, e.g., terms, are more important than others, e.g., contribute more by their presence in a document to the belief that the document satisfies the information need expressed by the query. The beliefs computed by these sum operators are normalized, e.g., the weighted sum is divided by the sum of the specified weights.

Note that, in general, an operator such as a weighted sum can be attached to any node of the inference network. If it is attached to a node of the query network, it is a component of the query, and the operands are the parents of the given node, which may be lower level (more nested) query components, or term representation nodes for the document under consideration. If a given document is instantiated, each term is assigned a belief, e.,g., it may be a *tf*idf* weight or a one (for strict boolean evaluation). The evaluation of the query will then evaluate the specified weighted sum which will then sum the beliefs of each query term in the document, weighting them by both the belief in the term, and the weight assigned by the user to the given term in the query. The result may be equivalent to a cosine similarity calculation, or a strict boolean evaluation, or an extended boolean evaluation, or something more complex, depending on the operator and weights assigned to each node.

Table 3 below summarizes some of the important characteristics of the IR systems described in this section, as determined by their documentation.

Table 3: Characteristics of IR Systems

	Lexis/ Nexis	Dialog	Dow Jones	Topic	SMART	INQUERY
Strict Boolean[a]	Y	Y	Y	Y	N	Y
Extended Boolean[b]	N	N	N	Y	N	Y
Proximity Operator	Y	Y	Y	Y	N	Y
Terms/ Keywords[c]	N	N	N	Y [d]	Y	Y[e]
Wild Card Terms[f]	Y	Y	N	Y	N	N
Stemming	N	N	N	Y[g]	Y[h]	Y[i]
Phrase	Y	Y	Y	Y	Y	Y
User-assigned Weights	N	N	N	Y	Y	Y
NL Query[j]	Y	N	N	N	Y	Y
Ranked Output[k]	Y	N	N	Y	Y	Y
Probabilis-tic	N	N/A	N/A	N	N	Y
Document ranking by Similarity	Y	N/A	N/A	Y	Y	Y

a. Exact Match (does not produce ranked output).
b. Does not require exact match (produces ranked output).
c. Roughly equivalent to a strict boolean with all terms connected by OR's.
d. ACCRUE operator is roughly equivalent to a keyword vector but uses a similarity function different from cosine similarity.
e. Unweighted sum operator (#sum) roughly equivalent to term vector
f. Wildcard terms can be used as an alternative to stemming, or in addition to stemming.
g. Stemming can be requested explicitly for any given word.
h. Automatic stemming
i. Automatic stemming
j. May be subject to semantic analysis, thesaurus expansion, or merely reduced to a term vector.

k. Ranked output may be produced by evaluating extended boolean, query/document similarity, or probability that document satisfies query.

16. Web-Based IR Systems

16.1 Web-Based Vs. Web-Accessible IR Systems

Commercial services such as DIALOG and LEXIS/NEXIS provide retrieval of documents (or abstracts) from a repository owned by the company providing the service. The repository may be specialized by subject, e.g., LEXIS provides access to legal reference material such as judicial opinions. The repository may, like NEXIS or Dow Jones News/Retrieval, provide access to articles appearing in any of a specified set of publications. Or like DIALOG, it may provide access to any of a broad but predetermined set of subject categories, and then within each category, to a predetermined set of publications. In all of these cases, the service preselects the subject categories, the publications, and the documents or articles from each publication to be made available to the user. These services are available via the Internet, but the user of such a service is restricted to the documents available in the repository maintained by the given service. In return for a fee, the user receives the benefit of the of the editorial judgment of the service, i.e., in selecting appropriate publications, selecting appropriate documents from each publication, indexing documents appropriately, etc. In this section, we consider a different class of retrieval services, those services (usually free) that provide access, at least in principle, to everything publicly available on the Internet, more specifically, on that very large portion of the Internet known as the World Wide Web (WWW), or more simply, the Web.

16.2 What a Web-Based IR Engine Must Do

Web retrieval engines do not maintain their own document repositories; the Web itself is their repository. They do build and maintain indexes to the Web. Since the Web is very large, very distributed, and grows and changes rapidly, one of the great challenges faced by these engines is to generate and maintain indexes that are sufficiently exhaustive (ideally, that cover the entire Web), sufficiently up-to-date, and sufficiently accurate. Since these search engines are available to, and widely used by, anyone with internet access, from a casual “surfer” of the Web, to a layman searching for information on a particular topic, to professional researchers and librarians, these general-purpose Web IR engines face a second great challenge: to provide interfaces simple enough for the layman but powerful enough for the professional. Since they cater to any user of the Web (rather than a more specialized class such as lawyers, medical researchers or journalists) and since the number and kind of topics, documents, and document collections on the Web is virtually unlimited, they face a third great challenge: they must be able to retrieve documents on any topic whatever. The first of these challenges is unique to IR engines that search a very large web, either *the* Web, i.e., the Internet Web, or a large corporate *intranet* web. The second and third challenges are exactly those addressed by most of the IR research described in this report.

16.3 Web Characteristics Relevant to IR

The Web consists of a very large and growing set of information units called “pages.” A Web page is a computer file; it may be the same size as an ordinary book page, but it may also be very much larger, and may even contain many book pages. These Web pages are woven together by (1) a common scheme for addressing pages, Universal Resource Locators (URLs), (2) a common protocol, Hypertext Transfer Protocol (HTTP) that allows a web client program on the user’s computer to request a page P1 by URL, and a web server on the computer where P1 is located to respond to the request by sending a copy of P1 to the user’s computer, and (3) a common standard for specifying the structure of a page, Hypertext Markup Language (HTML). HTML is a “markup” language, which means that each component of a page, e.g., its title, its author, abstract, figures, etc., is explicitly identified within the text of the page itself. A component is identified and delimited by tags preceding and following the given component. (This tagging process is called “markup.”) For the present discussion, only a few essential features of Web pages should be noted.

First, the components that occur and are tagged within an HTML page may be URLs of other pages. Since a given page may contain many URLs of other pages, and since many pages may contain the URL of a common, popular page, the pages comprising the Web are linked together in an elaborate structure of arbitrary complexity. This complexity explains the use of the term “Web.” But in fact, the structure of the Web is far more complex, far less orderly, than any actual spider web in nature.

Second, the presence in a given page, *P1*, of a URL pointing to a second page, *P2*, implies some association between the two pages. But there are no general uniform rules, let alone enforcement mechanisms, for ensuring that there is some reasonable connection, e.g., by author or topic, between any two pages linked by URL. Virtually any individual or organization can create a “home page,” a page expressing the interests or concerns of the given individual or organization. Any author of a page can link her page to any other, e.g., an individual may choose to link her page to other pages on diverse topics that she personally “likes.” *P2* may have been created by the author of *P1*, or by another individual working for the same organization, and may be on the same computer as *P1* (in which case they are said to be part of the same web site). But P2 may just as easily be a page created by a completely different author, and be located on a computer located in a different part of the world. A sentence in *P1* containing a tagged URL to *P2* may, but need not, explain the reason for the reference. Even if such an explanation is present, it will probably be in natural language, not readily interpretable by any search engine. The URL itself is a string of text, consisting of a number of standardized components in a standardized order. The most common URL format specifies the official registered internet name (variously called host name or server name or domain name) of the computer that provides access to the page being addressed, and the pathname (within the file structure of the given host) of the file where the page actually resides. The host name is itself a string of component names, ideally chosen to identify and give some clue(s) to the organization maintaining the host, e.g., “www.microsoft.com” is a computer serving as a gateway to all the computers within the commercial organization “Microsoft,” and “`www.cs.umbc.edu`” is a Web page for the Computer Science department of the University of Maryland at Baltimore County, an educational institution. Similarly, the file path name is a series of directory and sub-directory names, ideally chosen to give some clues to the subject or author or

purpose of the given page, e.g., “galaxy/Arts-and-Humanities//Performing-Arts/Drama-and-Theater.html” specifies a directory named “galaxy,” a sub-directory within “galaxy” named “Arts-and-Humanities,” a sub-directory within “Arts-and-Humanities” named “Performing-Arts,” and a file within the “Performing-Arts” directory named “Drama-and-Theater.html” containing URL‘s to numerous drama and theater resources. (Of course, to access this file, you need the other essential component of the URL, the host name, which happens to be “galaxy.einet.net.”) If these names are well chosen, their components can serve as keywords for an IR engine (or a human searcher). But again, there are no universal standards or enforcement mechanisms to ensure that the names of which a URL is composed are well-chosen or informative. In particular, these names are not chosen from any kind of controlled vocabulary.

Third, although the Web does not enforce any uniform or consistent semantic structure, its complexity makes possible the linking or grouping together of pages along multiple dimensions, e.g., a given page may be grouped with other pages by the same author (because the author’s home page contains URLs to all his publications), and also grouped together by topic (because a page devoted to the given topic contains the URLs of other pages devoted to the same topic). A page *P1* may refer by URL to page *P2* because *P2* deals with a topic mentioned or discussed in *P1*. Simultaneously, *P1* may contain a URL reference to page *P3* because *P3* deals with a different topic mentioned or discussed in *P1*. *P3*, in turn, may contain a URL reference to *P4* which deals with another related topic, or a special aspect, a sub-topic, of the main topic of *P3*. In this way, the URLs may specify a very elaborate network of citations by subject, author, etc. A user can trace such a citation path exactly as she would in a traditional library, but much more quickly and easily, e.g., by a succession of mouse clicks on highlighted URLs as they appear on her screen. On the other hand, the citations in a given page are only as valuable as the judgment of the author of the page, who (in the decentralized, anarchic world of the Web) is not subject to review by professional publishers or reviewers.

Fourth, a URL may point to a file that is *not* a Web page. This may mean that a Web server is available on the host, but the page is not in HTML format, and hence does not contain as much “metadata” as a document that has been properly “marked up.” In that case, the page can be retrieved by a client that supports HTTP (a Web “browser”), but it may require some additional program to display the file, e.g., an ordinary text editor for an ASCII text file, an image viewer for a file in some standard image format, etc. Or, it may mean that the file cannot be retrieved via the HTTP protocol because its host does not contain a Web (HTTP) server. In that case, some other kind of server must be used to retrieve the given file. The two most common non-Web servers are FTP and Gopher. An immense number of files, both textual and binary files, are available for transfer free of charge to the local host of any Internet user via an FTP server. (FTP stands for File Transfer Protocol).Textual files may contain any conceivable kind of textual or numeric data, including documents on any conceivable subject and program source code for distribution. Binary files may contain images, video, audio, executable programs, etc. The files at an FTP site may be arranged in a structure, typically hierarchical, that can be browsed by the Internet user. Any file that the user encounters in her browsing that “looks” interesting (perhaps on the basis of its pathname), can be retrieved using the FTP server and *really* looked at. However, these FTP files are not Web pages, and hence will *not* contain URL links to files (whether Web pages or FTP files) at other sites. The only browsing that can be done at an FTP site is up or down the local hierarchy of FTP files.

Gopher servers provide a hierarchy of menus. When the user selects an item from a given menu, e.g., with a mouse click, she gets either another menu or a data file. The data file may be any of the types that can be accessed via FTP. However, the file selected by a menu selection may be on a different host than the one where the menu was located. Hence, Gopher supports a form of "hyperlink" equivalent to a URL link between Web pages. In fact, the Gopher servers, menus, files, and links to files at other Gopher servers formed (and still form) an early form of hyperspace, called gopherspace, still quite useful and widespread, although it has been superseded as *the* hyperspace by the Web.

Finally, a Web page may be a gateway to a set of structured databases, textual databases, or other services outside of the Web itself. In other words, once such a page is reached, further access to a database or service may involve servers other than web servers, and local links other than URLs. Moreover, further access may depend upon payment of a fee, e.g., as in the case of the commercial databases described above. For example, you can reach the DIALOG home page free of charge. From there, you can jump freely to other pages that advertise the products and services that the DIALOG company provides commercially. The user can log on to these services from a DIALOG Web page. However, logging on requires a password and hence an account with DIALOG; this, in turn, requires payment of a fee. Hence, DIALOG databases and services can be accessed from the Web, but it is a commercial service and its databases are not a part of the Web.

An example of an intermediate form of commercial information source on the Web is the on-line bookseller, "amazon.com." From this company's web site, the user can freely browse through a very extensive book catalog, searching for books by title, author, or subject. When the user finds a book of interest, she may find not only the normal bibliographic data, but links to such additional information as reviews by customers, an interview with the author, etc. The user only incurs a charge when she orders a book.

By contrast, there is an extensive non-commercial on-line source of books (and magazines, journals, manuals, catalogs, etc.) called the Online Book Initiative (OBI), at the Gopher site "gopher.std.com." This resource is wholly free, and volunteer-run. Its advantages and disadvantages follow from this fact. The primary advantage is that the Gopher menus will lead you not to bibliographic data but to the actual text of an item of interest (which can be as short as a poem, or as long as a full-length novel). This text can be downloaded to the user's own computer, and either read right on her monitor screen or printed for more convenient reading later and elsewhere. On the other hand, the OBI is by necessity limited (except by accident) to material in the public domain, which is, of course, still an immense resource. Moreover, the material is archived in whatever format it is received from volunteer contributors, which means that it is in diverse formats, e.g., plain ASCII text, compressed text, HTML documents, sets of files archived together with the tar utility and then compressed, etc. The selection, though extensive, is necessarily haphazard, dependent on what has been submitted by volunteer contributors who may have scanned the text into electronic form on their own time, at their own expense. So, there is no rational or consistent basis for determining why this author or subject is represented, and that author or subject is not. Finally, there is no extensive or consistent indexing. The top-level menu mixes together authors and subjects in simple alphabetical order.

16.4 Web Search Engines

A good many free IR search engines are available on the Web. These engines allow a user to submit queries and retrieve a (usually ordered) list of Web pages that are (one hopes) relevant to the query.

Virtually all IR engines, commercial, free, or the research engines described in most of this report, work by indexing the document collection(s) to which retrieval is to be applied. With a relatively small collection, it may be possible to generate the index terms for a given document, its descriptors, dynamically, as the collection is being searched. (In a routing application, where the documents to be routed or classified or filtered are not available when the query is generated, there is no other choice.) But for a very large collection, indexing in advance is essential. For a huge set of collections such as the information sources available through the Web, no other course is possible. Hence, all of the prominent Web IR engines generate indexes; moreover, given the dynamic nature of the Web, they must constantly be updating their indexes, as new web sites are created, new information sources are created at existing web sites, information sources are updated, Web pages are created, updated, or deleted, etc.

There are two basic alternatives for creating an index:

(1) The index can be handcrafted by professional indexers as librarians have been doing for many years. This has the obvious advantage that human judgment is employed in deciding what a given document or resource is "about," i.e., what descriptors are appropriate for the given document. Moreover the index can be orderly, arranged by topic in a systematic and hierarchical fashion. The indexers can also exercise some editorial judgment with regard to what documents are worth indexing. (This can be either an advantage or a disadvantage, depending on whether the user wants expert judgment interposed between herself and the Web.)

(2) The index can be generated automatically, e.g., using techniques such as those discussed in this report. The big advantage of this approach is that it permits (relatively) more complete coverage. The Web is so immense and dynamic that it is virtually impossible for human indexers, working at human speeds, and taking time to exercise human judgment, to cover the Web completely.

Both approaches are in use on the Web today.

Another issue is whether the resource to be indexed is the Web as a whole, or some more specialized resource, available through the Web. If the resource is more specialized (and perhaps intended primarily for use by professionals), manual indexing is both more practical (because the resource though large, is still much smaller than the entire Web, and the topic area is likewise limited), and more worthwhile (because of the importance of the resource). An example of a free specialized resource for a field of great importance (medicine) is MEDLINE, "one of the world's largest biomedical databases with over 8,000,000 references to journal articles in all fields of medicine and related disciplines." MEDLINE is produced by the National Library of Medicine. Free advertiser-supported Web access is available through a commercial company. On the other hand, Lexis provides Web access (but not free access) to a major information resource in another major discipline, law.

The discussion that follows focuses on free, general-purpose Web IR engines, i.e., engines that attempt to index and access the entire Web.

16.4.1 Automated Indexing on the Web

There are two steps to automated indexing on the Web. First, the documents to be indexed must be found. Second, index terms must be generated for each document. Step two involves the kind of statistical and NLP techniques discussed earlier in this report. Step one is unique to a large hyperspace such as the Web, gopherspace, or a large corporate intranet. The documents, in this case Web pages, are not conveniently aggregated into one or a few collections, relatively static (documents added or deleted slowly), and stored at a few predefined sites. Instead, they are distributed over an immense and rapidly changing set of sites, linked together in complex structures by URLs. New web sites are frequently created, old sites are deleted or moved, new pages are frequently created, updated, or deleted at a given site. Hence algorithms must be developed for traversing the web structure at frequent intervals to find currently existing pages, so that step two, indexing can be performed on these pages. To carry the "web" metaphor a bit farther, the programs that execute these traversal algorithms are often called "spiders." Other names for them are "robots," "agents," "crawlers," "worms," etc. (Note that whether the Web is being traversed by a human user or by a robot, "traversal" from page *P1* to page *P2* does *not* mean that the human or robot is physically moving from the site of *P1* to the site of *P2*. Instead, it means that a web client or robot extracts the URL of *P2* from *P1*, issues a request for *P2* using this URL, and receives a copy of *P2* from the web server at the host computer where *P2* is located. Mobile agents capable of physically traveling from one network host to another actually do exist; however, they are not normally employed for Web indexing.) A robot that explores the Web to accumulate URLs is sometimes called a "discovery" robot.

A discovery robot begins its search with one or more popular known pages. These may be hand-selected by a human guide. Common starting points are Netscape's What's New or What's Cool pages, because these pages are obviously not subject-specific, and normally point to a wide variety of unrelated pages. Thereafter, the robot "automatically traverses the Web's hypertext structure by retrieving a document and recursively retrieving all documents that are referenced [by URL in the retrieved document]." However, in practice, the strategy is not quite so simple.

The Web structure is both complex and non-uniform. That is, different sites or regions of the Web may be structured according to different organizing principles. Hence, there is no one "best" way for a robot to traverse the Web. Moreover, some sites may have a "deep" structure, i.e., a structure with many levels of URL link. A site may also have many links to related pages, or pure digressions. Hence, a robot that attempts to traverse every path at a given site, or every path emanating from a given site, may devote an inordinate amount of time to the given site, at the cost of not getting around to many other sites. If the given site is organized around a given topic, this means that the robot will be devoting too much time to one topic at the expense of many others. Moreover, the site may have been created by an individual or organization with extensive expertise in the given topic, or by one eccentric individual with a quirky personal view of the topic. They are all the same to the Internet.

Furthermore, the robot should not request too many pages from a given site in rapid succession, lest it overburden (and perhaps even crash) the web server. Besides, the computer at that site may have other work to perform. So a robot may issue requests only at carefully calculated intervals, e.g., one request a minute, set a limit to the number of requests to a given site on one "visit," or request only a sample of pages at one visit to the site. Remember that a robot will usually be returning many times to a given site to keep its index updated, so on each visit it will be able to request some pages it didn't request on its previous visit.

In general, the robot may employ either a breadth-first or a depth-first strategy. The former (aimed at the broadest though perhaps shallowest coverage of the Web) means going only one level deep from a given page, before going back to the given page and looking for another link. The latter (aimed at deeper coverage of individual topics) means going up to N levels deep. N may be considerably greater than one, but some depth limit must be imposed for the practical reasons mentioned above. In either case, when a new page is retrieved, the robot extracts all URLs in the new page and adds them to its growing URL database. Of course, by limiting N to a value not too much greater than one, by limiting the maximum number of retrievals from a given site, and by judicious sampling of a the URLs in a given page, the robot may employ a strategy intermediate between pure breadth-first and pure depth-first.

Once the robot has explored all the links in a given page to whatever depth and proportion its strategy dictates, the next question is what page to explore next. In a breadth-first strategy, the robot will give preference to URLs that point to hosts the robot has not visited before, or hosts it has not visited recently, e.g., hosts it has not visited on the current update pass, or the last few passes. Given a choice among many URLs on the same host, the breadth-first robot will choose the ones with the shortest pathnames. This is based on the theory, often justified for well-designed sites but by no means guaranteed, that the pages at a given site are organized in a traditional subject-hierarchy. Hence, if two pathnames are of the same length, differing only at the name of the last (lowest) sub-directory, they are assumed to point to pages differing in subject at that level. On the other hand, if two pathnames differ in length with the longer name being an extension (more sub-directories) of the shorter name, then the longer name is assumed to point to a page dealing with a sub-topic of the subject dealt with by the page to which the shorter name points. Hence, given the assumption that different hosts deal with different subjects, and the assumption that pages at a given host are organized hierarchically by subject, a breadth-first strategy will maximize subject coverage. In any case, it will maximize web site coverage.

On the other hand, depth-first strategy can maximize coverage of "important" sites, by whatever criterion of importance is used to select the original pages. In particular, if the starting pages are selected by subject, then the depth-first approach can maximize coverage of the selected subjects.

An IR engine can speed up its rate of URL accumulation, and hence its coverage of the Web and its ability to update its index frequently and stay up-to-date, by running multiple robots in parallel, each robot traversing a different part of the Web. The speed can be further increased by running its robots on different computers. For example, "Open Text uses 14 64-bit servers 'working in tandem' to create and store its index."

A discovery robot need not save all the URLs it discovers. It may use characteristics of the URL or of the page itself to determine that the page is not "worth" indexing, and hence choose to discard its URL. The criteria used to determine whether a page is worth indexing are generally not documented. Some URLs will be discarded because the are "dead," i.e., they point to pages that no longer exist.

Given the complex structure of the Web, in which many pages may point to the same popular page, a robot will often discover the same URL more than once. Hence, an important feature of building the URL database is sorting the URLs and removing duplicates. A further complication is that, in many cases, the same page may be replicated at multiple sites, or at the same site with multiple URLs (aliases). Hence, it is not enough to eliminate duplicate URLs; it is becoming increasingly important to recognize two pages reached by different URLs as being identical, either because they are literally the same page reached via two different URLs, or because one page is an identical copy of another. (The practice of copying popular pages is common.) To make matters still more difficult, one page may be a copy or near-copy of another, yet not an identical copy. For example, copies may differ in format, e.g., one copy may be in HTML and another copy in Postscript. Or one copy may be a slightly older version of another. Hence, algorithms for detecting near-duplicates have been developed. [Shivakumar et al., WebDB98] [Shivakumar et al., 1995] These algorithms are generally based on computing "fingerprints" for documents, either at the whole-document level, or at finer levels of granularity, e.g., paragraphs, lines of text, words, etc. Whatever the chosen level of granularity, documents are divided into "chunks" at that level. Each chunk is replaced by some compact representation, its "fingerprint." Documents, e.g., Web pages, are compared according to a similarity function based on the number of chunks they share. Algorithms vary according to the level of chunk, the method of representing the chunks, and the similarity function. The lower the level of the chunks, the greater the ability to detect partial overlaps, but the greater the chance of detecting "false positives," documents that are falsely said to be similar. Note that the chunk level, the similarity function, and the similarity threshold above which two documents are said to be similar, will vary with the user's purpose. The threshold for discarding duplicate or near-duplicate Web pages may be quite different from the threshold for detecting possible plagiarism.

Once a database of URLs has been accumulated, the discovery robot (or another robot) can "harvest" the URLs, i.e., retrieve each page by URL and index it, using techniques such as those discussed elsewhere in this report. IR engines can vary, not only in the order in which they find and save URLs, but also in how they order the URLs in their database, and hence the order in which they retrieve the pages to which these URLs point for harvesting. They can also vary in how they intertwine URL accumulation and harvesting. Since a URL database will contain both URLs of pages that have never been visited and harvested, and pages that have already been harvested at least once but must be revisited to update their index entries, harvesters may also vary in the priority they give to unvisited pages and the frequency with which they revisit pages. For example, a harvester may start with the oldest URL not yet visited (retrieved). It may retrieve and index that page, and then go back to its URL database to retrieve the next oldest unvisited URL. It may continue in this way until there are no more unvisited pages, and then go on to revisit pages, starting with the page that has gone the longest time without a revisit. On the other hand, since the discovery robot is continuing to update the URL database with new URLs, the harvester may need to balance visiting "new" pages, with revisiting "old," already visited, pages.

Or (as a narrower, more focused strategy to fill in subject gaps in its index), the harvester may start with a page judged relevant to a given topic (perhaps on the basis of its URL or its existing index entry), and accumulate and harvest all the pages linked to the given starting page. In this strategy, harvesting and accumulating new URLs are intertwined.

Since Web pages are HTML tagged, the harvester can use these tags as guides (an option not available to IR research engines such as those discussed elsewhere in this report, that index arbitrary collections of text). For example, since URLs are always tagged, the harvester can apply its indexing techniques to the text of each URL that occurs *in* a given page, to generate index terms *for* the given page. (Of course, it can also index the text of the URL that *points* to the given page.) It may generate index terms from other components of the page itself, e.g., a component tagged "title," or a component labeled "description." It may index the text of "hyperlinks" (also called simply "links"); a hyperlink is the highlighted or underlined text that a user sees, and on which she clicks to invoke the underlying URL and retrieve the page to which the URL points. It may also generate index terms from the full text of each page; remember that a page may be a lengthy file. The harvester may also take HTML tags into account in weighting index terms, e.g., by giving a higher weight to a key word in a component with the tag "title."

Web IR engines vary considerably in what they index. For example, some engines do not index URLs, so that a query on a term that appears only in a page's URL, but not in the page itself, will fail to retrieve the given page with such an engine. Other engines may index a "description" or "body" component, but limit themselves to some maximum number of words within the given component. On the other hand, some IR engines, e.g., WebCrawler, AltaVista, Infoseek, do full-text indexing of the pages they index. Some IR engines, e.g., AltaVista, index word position as well as content, permitting support of proximity conditions in queries; others do not. Some IR engines use HTML tags, either to determine what text to index, or to determine what weight to give to a given index term; other IR engines ignore HTML tags.

Robot-based Web IR engines also vary in what kind of pages they will index. The robot can only index pages it can reach, i.e., pages that have URLs that are referenced in Web pages. As noted earlier, not all the pages to which URLs points are themselves Web pages, formatted in HTML. Some engines only index true HTML-formatted Web pages. Some will also index Gopher pages, FTP pages, or simple text (ASCII) pages. However, even if a robot indexes Gopher or FTP pages, it cannot automatically traverse a Gopher or FTP site. Hence, it will only index those Gopher or FTP or simple ASCII pages that it can reach directly by URL.

Some primarily robotic Web IR engines, e.g., WebCrawler, Infoseek, Excite, also allow the author of a Web page to submit its URL. Hence, these engines may index pages that their normal robot searches would not have discovered, or would have discovered much later. However, such human submissions represents only a small part of the index of such an engine.

What algorithms do the robot-based Web IR engines use to harvest summaries and index terms from the pages whose URLs they accumulate? Detailed information on this subject is generally not available. Some engines use conventional stop lists, e.g., Excite, HotBot, Lycos, WebCrawler, to eliminate common words of little value as page descriptors. Some engines, e.g., InfoSeek, use

statistical or NLP techniques to weight common stopwords lower than more significant words or to weight words that are rare on the Web higher. Some use conventional stemming techniques to normalize index terms. A number of engines, e.g., Lycos, AltaVista, use term frequencies within the page being indexed. Since Lycos also collects the total number of words in a page, it may be presumed that it normalizes term frequencies. Some engines limit the number of words or lines of text they index. For example, Lycos indexes the "first 20 lines," but doesn't specify unambiguously what it means by that phrase. Most engines build a summary or surrogate record. This record almost always contains the title. It often contains a fixed amount of text from the beginning of the document, e.g., the first 50 words. Alternatively, it may contain the most "weighty" words, according to some algorithm based on statistical frequency, position, or both. Position may refer to relative position in the page, e.g., words near the beginning of the document count for more; or, position may refer to the HTML-tagged components in which it appears. In particular, some engines look for a component (if any) tagged "description" by the page's author, and use its contents (if it exists) as a page summary instead of generating a summary automatically, e.g., InfoSeek accepts a description of up to 200 words. An engine may use HTML tags in various other ways, e.g., AltaVista effectively indexes terms by both relative position in the document, and by the HTML-tagged component(s) in which it appears; this enables AltaVista to support both proximity searches (term A must be near term B) and queries specifying that a given term must occur in a given tagged component. On the other hand, other engines treat meta-tags as just ordinary text for indexing or summary purposes.

Despite the variety of criteria used by the various robot-based engines to harvest pages and build their indexes, most of them have one thing in common: The descriptors by which they index pages are primarily words or phrases actually contained in the pages being harvested. (Of course, they may also use other descriptors, e.g., the date a page was created or discovered or harvested, or words in the URL *o*f the given page.) Excite differs from most other engines in that it uses LSI (see section on LSI) to build its index. This means that it creates an index of "concepts," derived statistically by co-occurrence from the actual words. Hence, two pages may be indexed by the same concepts even though they differ substantially in the keywords they contain, provided that their keywords co-occur with many of the same words in other pages.

One problem for the user seeking to understand IR engine behavior is that most of the popular general-purpose IR engines, though free, are nonetheless proprietary. Hence, the source code is usually not available, and the documentation with respect to strategies for gathering URLs and indexing pages is frequently incomplete or inaccurate, and sometimes non-existent. These "free" engines are usually commercial enterprises, supporting themselves either by advertising or by selling their software to private organizations for use on corporate intranets. So, they have an incentive to attract users more successfully than competitors. Their documentation is, at least partly, a form of advertising.

Moreover, the fact that these free, commercial IR engines are advertising-supported has another, curious consequence. The company operating the engine is providing a service, which, to the extent that it works well, will provide links to other Web pages. Yet, the effectiveness of advertising and the number of advertisements the company can display to the user (not to mention the possibility that the user will follow a link provided by one of the advertisers) depends on keeping the user at the web site of the IR engine itself as long as possible. Hence, the web sites of these IR

engines tend to grow, adding information and entertainment resources locally, so that the user will have more reason to remain at the given web site, instead of using the IR engine!

Finally, it is noteworthy that general web IR engines do not (and have found that they cannot) charge fees for their service (because users will not pay them), while commercial database services can and do charge fees. Clearly, customers are willing to pay for effective human indexing of a large, well-chosen, set of databases. Just as clearly, customers are not willing to pay for IR search engines to the Web. Although it is potentially an enormous information resource, the Web is currently too broad, shallow, chaotic, and unfocused. And, search engines based on robotic indexing are currently too unreliable, returning too many references, or too few, or the wrong references, or references poorly ranked.

16.4.2 Manual Indexing on the Web

The preceding section discussed free, general purpose Web IR engines that generate their huge indexes primarily by executing robots: discovery robots that traverse the Web looking for URLs and build a URL database, and harvester robots that retrieve the pages to which the collected URLs point, and index each page using the kind of techniques discussed elsewhere in this report. However, there are also free general purpose Web IR engines that employ a staff of professional indexers to specify a hierarchy of subject categories similar to that found in a traditional library, e.g., the famous Dewey Decimal System, and index Web pages in terms of these subject categories. In other words, human beings determine the subject categories which serve as index terms and the words used to name the categories, so the pages are indexed by a controlled predetermined vocabulary. (By contrast, the robot-based engines generate index descriptors from the content of the pages themselves, using statistical and positional clues.) Similarly, human beings organize the index terms into major categories, sub-categories of the major categories, sub-sub-categories, etc., for as many levels as seems appropriate. Hence, a human user can search down the subject hierarchy to find the narrow, specific subject category in which she is interested. (By contrast, the robot based engines do not determine any logical structure among the index terms they generate. It may happen that an index term generated for one page may be a sub-category of an index term generated for another page, but the robot(s) that generate the terms won't know that or hence can't tell the user.) Finally, human beings determine what subject categories should index a given page. Since human knowledge and judgment is involved, a page may be indexed to a subject category, even though no name of that category appears in the given page or its URL.(By contrast, most robot-based IR engines will only index a page to terms that actually occur in the page or its URL. One robot-based engine, Excite, claims to be using the statistical technique, LSI. As discussed earlier in section 6.5, such co-occurrence techniques can sometimes recognize the similarity of a query to a document even though the document contains no words that are in the query. But this is a long way from true human understanding.)

The best known, most widely used web IR engine that employs manual (human) indexing is Yahoo! (the exclamation point is part of the name). Since Yahoo! is not robot-based, it depends on voluntary human submission to obtain the URLs it indexes. (Actually, just as some of the robot-based engines also accept manual submissions, so Yahoo! also operates a robot. However, just as the manual submissions represent a small part of the harvest for robot-based engines, so robot-retrieved pages represent a very small part of Yahoo's index.) The human author or publisher of a

Web page can submit its URL to Yahoo! using an automated procedure available at the Yahoo! site. She browses the Yahoo! subject hierarchy looking for an appropriate subject category or sub-category, one that correctly describes the page she is submitting. When she reaches the page corresponding to the category she considers most descriptive of her page, she clicks on a "suggest URL" button (which is available at the bottom of every page). This causes her to enter the "add URL" procedure with the current category as the "preferred" category. She can suggest additional categories that also describe her page. She can provide a title, and a short textual description of the page. And of course, she specifies the URL of the page she is submitting.

A number of characteristics of Yahoo! should be noted:

First, each subject category is itself a Web page. The category structure is created by linking the page for each category to the pages for all its sub-categories through the usual URL/HTTP mechanism. Of course, the page for a given category is also linked to any external Web pages that the given subject category indexes. Hence, when a Yahoo! user clicks on an entry in a subject category page, she is requesting the page whose URL is associated with that entry, and making the usual "hyperlink" jump typical of the Web.

Second, a submitted Web page can be linked to a category at any level, e.g., it can be linked to the main category, "Social Science," or the sub-category, "Anthropology_and_Archaeology," or the sub-sub-category, "Archaeology," or the still lower sub-category, "Marine Archaeology."

Third, a page may be linked to several categories or sub-categories. In that case, it will be reachable from each of those categories.

Fourth, the Yahoo! indexers (called "surfers by the company) will review each submitted page, and may override the authors own choices for preferred or additional subject categories, changing, adding, or deleting categories from the set chosen by the submitter.

Fifth, if the submitter can't find any category appropriate to describe her page in the existing Yahoo! subject category structure (or thinks some new category is needed as an additional descriptor), she can suggest one or more new categories, and indicate where these suggested new categories should be inserted within the existing structure. The Yahoo! staff may accept, reject, or modify these suggestions.

Sixth, sub-categories as well as external Web pages can be referenced from multiple higher-level category or sub-category pages; in other words, a given subject category can be a sub-category of more than one higher-level category. For example, the category "Aphasia" can be found as a sub-category of "Linguistics_and_Human_Languages," which is a sub-category of the main category, "Social_Science." However, it can also be found as a sub-category of "Diseases_and_Conditions," which is a sub-category of the main category, "Health." When a sub-category appears in more than one place in the subject category hierarchy, one of those references is the "primary" reference; clicking on any other occurrence of the sub-category will get the user to the same page as clicking on the primary reference. In the above example, clicking on "Aphasia" under "Linguistics_and_Human_Languages" will cause a hyperlink jump to the same "Aphasia" Web page as clicking on "Aphasia" under "Diseases_and_Conditions." However, the actual position of

the “Aphasia” Web page in the Yahoo! directory structure, as determined by the pathname component of its URL, is under the “Diseases_and_Conditions” directory.

Seventh, each external web page is represented in the Yahoo directory by a surrogate page containing the title, description (if any), and the URL. Clicking on the URL in this surrogate page will cause a normal hyperlink jump to the actual, external Web page selected by the user. Some surrogate pages may contain URLs of multiple external Web pages. Such a surrogate page may viewed as a lowest level subject category page, i.e., a page that contains references only to external pages and none to other subject categories. For example, the “Aphasia” page contained references (at the time of writing) to four external Web pages dealing (at least partly) with the subject of aphasia. One of these references is to the home page of an organization devoted to the treatment of the disease. Another is to a National Institutes of Health (NIH) paper on aphasia. And so on. This set of references could be updated at any time, new references added, obsolete references deleted. Naturally, new references to aphasia will only be added if someone submits (“suggests” is the term currently used by Yahoo!) a new Web page dealing (at least partly) with aphasia.

The Yahoo! approach has inevitable drawbacks as well as advantages. Since Yahoo! is primarily dependent on voluntary submissions, its coverage of the Web is inevitably very incomplete and uneven. If the user wants to issue a query on a subject that does not fit any of Yahoo’s existing categories, or is an unanticipated hybrid of those categories, she may be out of luck. On the other hand, if the user’s query leads her to one of Yahoo’s sub-categories, it is likely that a high proportion of the pages returned will be relevant to the query. Moreover, a relevant page indexed by Yahoo! may often have links to other relevant pages, not indexed directly by Yahoo! So, Yahoo! is often a good place to start a search. Pages on a given topic retrieved via Yahoo! may not only be linked to other pages. Their content may suggest good search terms to use in a query via one of the robot-based engines.

Finally, it should be stressed that Yahoo!, despite the “value added” by extensive human indexing, and development of a subject category hierarchy, is nevertheless a “free” service, supported by advertising rather than user fees. Evidently, customers will not pay fees unless the service supplies human-generated or selected databases as well as human indexing.

16.4.3 Querying on the Web

IR engines on the Web don’t break any new ground relative to the research engines discussed elsewhere in this report.

Many IR engines provide two levels of query formulation, a “basic” level, and an “advanced” level (which may enjoy a fancier name like “power” level). The basic level is typically a set of keywords, combined logically by a default boolean “OR.” In other words, they are effectively term vectors. The query terms are normally words; some engines, e.g., WebCrawler, InfoSeek, AltaVista, also support phrases, typically by enclosing a sequence of words in quotation marks. Some engines allow the user to precede a term by a plus (“+”) or a minus (“-”). The plus may be a weak form of AND, i.e., the designated term must be present in the page. The minus may be equivalent to NOT, i.e., the designated term must not be in the page. (Lycos uses the minus a little

differently; putting a minus in front of a keyword “a” means, not that pages containing the term “a” are excluded, but that they will be ranked lower on the list of pages returned to the user.)

The advanced level usually offers some form of boolean query, with the operators AND, OR, and NOT. Some engines also support parentheses to control the order of evaluation of operators. Often, proximity operators are offered too. For example, WebCrawler offers both a NEAR and an ADJ (for adjacent) operator. The condition “a NEAR/N b” says that terms a and b must occur within N words of each other; the condition “a ADJ b” says that a must immediately follows b, in that order. By contrast, AltaVista’s NEAR operator does not allow the user to specify the degree of proximity; it is always a separation of 0 to 10 intervening words. Open Text’s “a FOLLOWED BY b” requires that a and b occur in the specified order, but they need not be adjacent, only within a certain degree of proximity; as with AltaVista, the user cannot specify this degree. Lycos offers an interesting enhancement to the traditional boolean AND (or looking at it from a different perspective, an enhanced degree of control of a “soft” boolean); the user can specify how many of the search terms (from 2 to 7) must be present in a given page.

Some engines support stoplists, although their documentation usually doesn’t tell you what words are on the list. Other engines effectively generate their stop lists statistically, ignoring words found to be too common on the Web. And some engines provide no stoplist at all. Similarly, some engines, e.g., InfoSeek, provide stemming, but their stemming algorithms are not publicly documented; others do not provide automatic stemming at all. Some engines provide a wildcard character to allow the user to some (limited) stemming on her own. For example, AltaVista supports an asterisk (“*”) either at the end of a word, or in the middle (but it must be preceded by at least three characters). So “run*” can find “run,” “runs,” “running,” etc., while “labo*r” finds both “labor” and “labour.” On the other hand, Lycos provides only truncation at the end of a word, e.g., “run$.” However, truncation is the default in Lycos; a period must be specified to inhibit truncation.

The nature of the Web allows IR engines to support certain more specialized query conditions. For example, HotBot supports querying by Internet domain; the “domain” is the final component of the host name in a URL, e.g., “edu” is the domain of educational institutions such as universities, “com” is the domain of commercial organizations, “mil” is the domain of military organizations. (Note that the entire Internet address of a host computer is also called a “domain” name; hence, the components are sometimes called “sub-domains,” and the righthand component is then called the “top-level” sub-domain.) These “old” domain names were developed when the Internet was largely restricted to the U.S. As the Internet has grown and become international, a “new” set of two-character domain codes based on country has been developed, e.g., “uk” is the United Kingdom, “ca” is Canada, “jp” is Japan, etc. HotBot allows the user to restrict its search to (1) a given organizational domain, e.g., “edu” restricts the search to hosts belonging to educational institutions, (2) a given country, e.g., “jp” restricts the search to Japanese hosts, or (3) a given geographical region, e.g., “Europe” would restrict the search to domain codes corresponding to countries in Europe. HotBot also supports a query condition restricting the search to pages containing URLs pointing to files of a given *media type*. By convention, the media type of a file is specified by a suffix called an “extension.” For example, the filename of an HTML formatted page will end with the extension “.html” or “.htm.” The name of an image file in Graphics Interchange Format will end with the extension “.gif,” etc. Hence, a HotBot user can restrict her search so that e.g., only

pages containing URLs ending with ".gif" will be returned. (Remember that a URL address consists of a hostname followed by a pathname, and that the pathname consists of a series of directory and sub-directory names terminating in the actual filename; hence, the extension will be a suffix of this filename.)

AltaVista takes advantage of the HTML format of a conventional Web page to support query conditions based on field. For example, the user can specify that search of a a page for a given word, phrase, or boolean combination, should be restricted to the title, the body of text that the user sees, the URL *of* the given page, a URL within the given page, the host name in the URL of the given page, or a link within the given page pointing to a specified kind of file such as a Java applet or an image file., etc. Similarly, InfoSeek allows the user to restrict the term search of a given page to the title, or the URLs of links within the body of the page. An infoSeek URL search returns all pages whose URLs contain the specified term. The InfoSeek user can also specify a "site" search, which means that the search retrieves only those pages that contain the given term in the hostname of their respective URLs. This feature can be used to retrieve all pages at a given "site," i.e., all pages available at a given host. But it can also be used to retrieve all pages at a number of hosts having the specified component(s) in common. And, Open Text supports searches on title, URL (similar to the InfoSeek URL search), and "First Heading." The latter restricts the search to the first HTML component tagged "<HEADING>."

Since Yahoo! indexes pages by an elaborate hierarchy of human-generated subject categories, the natural way for a user to generate a query to Yahoo! is to traverse this subject category index from general to more specific category until she finds the narrow category that best categorizes the subject in which she is interested. However, the user may want to go straight to a desired category, without browsing through the subject hierarchy. Also, she may want a category that doesn't exactly fit any of the predefined categories. Hence, Yahoo! also provides a keyword and boolean search capability similar to those provided by the robot-based search engines. This search is initially applied to Yahoo's own index, i.e., to the names of categories in the subject hierarchy, and to the words that appear in the surrogate records generated at the lowest level of the Yahoo! index. However, if the search is not satisfied by the Yahoo! index, Yahoo! passes the search to AltaVista, which searches its own, very large, robot-generated index. (Naturally, going to AltaVista increases the likelihood that the user will get a "hit," but decreases the chance that the higher-ranked hits will actually be about the subject that interests the user, since the AltaVista's indexing is automated rather than human-generated.)

On the other hand, some robot-based engines, e.g., InfoSeek, Lycos, and WebCrawler, also provide a limited human-generated hierarchical subject category index. In the case of Lycos, the index is limited to the most "popular" sites, where popularity is measured by the number of links from other sites exceeding a given threshold. WebCrawler's index provides reviews of sites that its human operators have somehow judged "best of the net." In general, a user who wants to browse by subject category will do better to use Yahoo's far more extensive and well-organized subject index.

How do these Web IR engines compute the score, i.e., similarity, of each indexed Web pages relative to a given query? As with indexing, the Web engines usually don't document their similarity/ranking algorithms. The one clear (and desirable) point is that all of the major engines *do* rank

their results. Given the large number of results they may return, ranking is a necessity. As with the research systems described elsewhere in this report, the ranking is usually far from perfect.

Generally speaking, the engines rank pages (as one would expect) according to the number of "hits," e.g., on a simple OR of a set of keywords, the highest ranking pages will tend be those that contain the largest number of specified terms, or the largest number of occurrences of the search terms. More generally, the highest ranking pages will tend to be those that satisfy the most of the specified search conditions. Some engines, e.g., WebCrawler, InfoSeek, give higher weight to terms that are less common on the Web. Some weight query terms, i.e., search terms, more highly if they are in a "significant" position in a given page, e.g., Lycos and InfoSeek weight terms more highly if they appear in the title, AltaVista and InfoSeek weight terms more heavily if they occur near the beginning of the page, HotBot gives higher weights to terms in <title> and <Meta> components, etc. Lycos weighs search terms more heavily if they are in close proximity. Some engines, e.g., InfoSeek, AltaVista, Hotbot, weight terms more heavily based on frequency of occurrence. Hotbot bases its weighting on normalized term frequency, i.e., a given term frequency will count more in a short page than in a long page. Some engines allow the user to weight a query term explicitly, e.g., Excite allows the user to attach a weight to a query term either by specifying the weight as a number: "Gates^3," or by repeating the term: Gates Gates Gates." InfoSeek weights more heavily those terms that appear earlier in a given query, so the user can weight query terms implicitly, but can't assign relative numeric weights.

Excite claims to use LSI. Therefore, its query-similarity algorithm must differ in at least one essential respect from that of the other engines. It must compute a concept similarity rather than a word or phrase similarity. In other words, pages must be ranked according to the degree of similarity between the (statistically derived) concepts by which they are indexed, and the (statistically derived) concepts computed from the user's query. Ordinary term weights still remain significant, but their significance is the effect they have on the statistical derivation of the concepts.

A number of engines assign a relevance score as well as a ranking. As noted earlier, relevance scores generally have no absolute, independent meaning to the user unless they are probabilities of relevance; none of these Web engines claim to be computing such a probability. Typically, engines assign scores in the range 0 to 100. However, InfoSeek appears to claim that all its relevance scores are computed against an objective standard. This means that the relevance score of the highest ranking page in one search can be compared against the corresponding score in another search. If the highest score in search one is 42, and the highest score in search two is 77, then the highest ranking page in the second search is a lot more likely to be relevant than the highest ranking page in the first search. Moreover, the best score, e.g., in search one, may be far below 100. Hence, InfoSeek can tell the user that the even the best score is not too likely to be relevant to her query. By contrast, many other engines will compute scores that only compare pages retrieved in a given search for pages matching a given query. Hence, the highest score will be at or close to the maximum, e.g., 99 or 100, no matter how poor the query-page match.

Finally, it should be noted that some engines provide a limited form of relevance feedback. The user can designate one of the pages retrieved by a previous search as satisfying her needs especially well, and ask for more pages like the designated one. The engine then retrieves pages as similar as possible to the "good" one. Excite calls this feature 'Query by Example." Open Text

call its corresponding feature ‘Find Similar Pages.” However, these two engines compute page-page similarity (and query-page similarity) differently. Open Text identifies the most frequently occurring words in the designated page, and looks via its index for other pages containing those words, especially pages that contain those in the <TITLE> or first <HEADING> components. EXCITE uses LSI to map the designated page into LSI concepts, and then searches its index for other pages associated with the same concepts. As noted earlier, two pages may be about the same concepts even though they differ in the words they contain, provided that they both contain words associated statistically with those shared concepts.

16.4.4 Meta-Querying on the Web

As the previous sections illustrated, Web IR engines vary enormously in how they traverse the Web to discover URLs, how frequently they update their list of discovered URLs, how they index the pages to which those URLs point, how often they harvest the discovered pages to update their index, what kinds of queries they allow the user to formulate, how they interpret the user’s queries, how they compute the similarity between a user’s query and the pages they have indexed, and how they order, i.e., rank, the pages they retrieve for presentation to the user. Moreover, the Web itself changes and grows very rapidly. Hence, no one engine can possibly index the Web completely. Moreover, engines vary enormously in what parts or proportion of the Web they cover, how deeply they cover a given part of the Web, how well (or easily) they will permit the user to formulate a query that expresses her needs, how well they will do at retrieving relevant pages, or ranking the pages by degree of relevance, etc. Hence, it is clear that for many kinds of query, the user will do best by searching with more than one engine.

But precisely because of all this variation, it can be a tedious chore to execute multiple engines and collate the results returned by these engines. One response to this problem is to provide a meta-engine that allows the user to generate a single query from a single interface. MetaCrawler {Selberg & Etzioni, WWW Journal, 1995] [Etzioni, AAAI-96, 1996] is an example of such a meta-engine. It accepts a single query from the user, translates it into queries for multiple IR engines, and issues these queries to their respective engines, which then execute the queries in parallel. Typically, each engine returns a ranked list of URLs from its index. MetaCrawler collates these URL lists, cleans up the result, and presents a single result list to the user. Etzioni [ibid] uses the metaphor of a “food chain.” The Web IR engines are herbivores, presumably because they “graze” directly on the information in the Web itself. A meta-engine like WebCrawler is a carnivore, “feeding” off the herbivores. (Happily, it does not consume the herbivores in the process! The metaphor breaks down at that point.) Applications that invoke WebCrawler or some other general-purpose meta-engine, are still higher up the food chain.

The MetaCrawler query interface is similar to the interfaces of the Web IR engines it drives. Its basic level supports “any” (boolean keyword “OR”), “all” (boolean keyword “AND”), and “phrase” (exact match on series of words). It also offers a pull-down menu choice between searching the Web and various more specialized searches. e.g., stock market quotes. The “power” search allows the user to restrict the search to a high-level domain, either a geographic domain by continent, or one of the three traditional domains: com, edu, and gov. This domain restriction is similar to that provided by HotBot but the set of choices is more limited. The more “refined” search uses conventional punctuation: phrases are enclosed in quotes, a + prefixing a word or phrase means

the term must be in each page returned, a - prefixing a word or phrase means that the term must not be in each page returned. A number of the IR engines discussed above support similar rules involving +, -, and quotes.

What "value added" does MetaCrawler offer?

1. The user sees only a single interface, and only has to learn a single query syntax. MetaCrawler translates the user's query into the diverse syntaxes of the various engines it knows. This raises an obvious question. What if MetaCrawler's own query interface offers a feature not supported by some of the engines it drives? For example, MetaCrawler supports phrase searches; some of the engines described above do not. What should MetaCrawler do? There are several possibilities: MetaCrawler can approximate the phrase, e.g., with a boolean "AND" of the terms comprising the phrase. Then, MetaCrawler can either (a) return all retrieved pages, with those that satisfied a mere "AND" ranked lower than those that satisfied a strict phrase query, or (b) test retrieved pages for a strict phrase condition itself, during post-retrieval processing. The phrase test would only have to be applied to pages retrieved by engines that do not support exact phrase queries themselves. A third possibility (c) is to restrict the scope of exact phrase queries to those engines that *do* support them. Alternative (b) is more expensive in post-processing time; on the other hand, alternative (c) reduces MetaCrawler's coverage, one of its most attractive features. At the time of writing, commercial MetaCrawler said that it would like to avoid (c), but it was "under consideration."

2. Web coverage is broader than that which can be provided by any single IR engine. There are three reasons. First, as noted earlier, no engine can hope to cover the entire Web; it is simply too large and grows too fast. Moreover, different engines have different algorithms for traversing the Web, and discovering URLs. They vary too in the number of robots they employ, the degree of parallelism of these robots, the frequency with which the Web is traversed for update purposes, etc. It follows that there will be substantial variation in which URLs each engine has discovered and indexed, and how up-to-date the URL discovery and harvesting is. Second, IR engines vary substantially in how they harvest the pages whose URLs they have discovered, what descriptors they extract from a given page and put in their respective indexes. Third, they vary substantially in how they compute query-page similarity (relevance), and how they rank references to the retrieved pages for presentation to the user. The first reason means that two different engines may be applying a given query to sets of URLs that only partially overlap; many URLs may be unique to the index of one engine or the other. The second and third reasons mean that even if a given URL is present in the indexes of two different engines, a given query may retrieve it from one index but not the other; another query may have the reverse effect, retrieving the URL from the second index, but not the first. In other words, different engines have different strong points and weak points with regard to Web coverage, indexing, query evaluation and ranking. A meta-engine, by collating the retrieval results of many engines can achieve greater coverage than any one IR engine alone.

3. WebCrawler is lightweight. It employs no robots, does no discovery or harvesting, and does not build or maintain a huge Web index. It leaves all of that hard work, and the corresponding required storage capacity and processing power to the IR engines it invokes, engines such as those discussed in previous sections. This means that it would be quite possible for copies of Web-

Crawler, called WebCrawler "clients" by the creators of this meta-engine, to reside on user PCs. (In actual fact, MetaCrawler has followed the path taken by many IR engines, evolving from a University research project to a commercial engine. Copies of MetaCrawler are not for sale, although instructions are available in the MetaCrawler Frequently Asked Questions (FAQ) on how to make MetaCrawler the default search engine in MS Internet explorer, Netscape Navigator/ Communicator, etc. But such defaults only determine which search engine will be invoked preferentially. It doesn't put the meta-engine on the user's own system.) Because WebCrawler doesn't do all the work of a conventional IR engine, it can devote its resources to intelligent post-processing of the retrieved URLs.

4. The standard post-processing of a query that commercial WebCrawler performs includes collating, i.e., merging, the URL streams it receives from the various engines it invokes, removing duplicates, and *verifying* the URLs. Individual engines attempt to remove duplicate URLs from their indexes. However, the inevitable overlap of the indexes of different IR engines means that there will be considerable duplication when result sets from multiple engines are merged. WebCrawler removes these duplicates, but it also combines and normalizes (into a single number from 1 to 1000) the confidence/relevance scores assigned by the various engines that returned a given URL, and lists each of the engines that returned the given reference. The combination/normalization algorithm is not documented. Verifying a URL means reading the page to which it points, and determining that the page still exists. Given the dynamic nature of the Web, and the inevitable period between successive visits by a given engine to a given site, it is inevitable that each engine's index will contain some proportion of *dead* links, URLs pointing to pages that have been deleted or moved since they were last visited. WebCrawler can delete such dead links.

5. [Selberg & Etzioni, WWW Journal, 1995] suggest a variety of other kinds of post-processing that a MetaCrawler, especially a copy residing on a user's PC, could do. These include "rescor[ing] the page using supplementary syntax supplied by the user," clustering the results, "engag[ing] in secondary search[es] by following references to related pages," customized filtering, e.g., of X-rated pages, etc. At the organizational level, WebCrawler could cache retrieved pages, either those that were highly ranked, or those that one or more users designated as relevant, for sharing by members of the organization. This could facilitate collaboration or information exchange. Or, it could simply reflect a presumption that people within the same organization, will have shared professional interests, and hence will want to see many of the same pages. A client WebCrawler could also support scheduled, or data driven queries, e.g., "Retrieve reports daily on the XYZ company." Another possibility [Etzioni, AAAI-96, 1996] is that the client could engage in a dialog with the user to enable her to better focus her query. The existing commercial WebCrawler supports a limited form of customization, but since the meta-engine itself resides at a centralized web site, the customization consists entirely of placing a WebCrawler form on the user's system, and specializing the existing generic query capability.

In fairness, it should be noted that Maze et al. [1997] in their survey of Web IR engines advise *against* using meta-tools. Their argument is that since Web IR engines are so diverse in the query capabilities they offer, and in the way they interpret queries, e.g., one engine may recognize "AND" as a boolean operator while another does not, the user may be better off generating separate queries to each of two or three engines than issuing one query to a meta-engine. Of course,

that means investing the extra effort to learn the syntax of each engine. Only experience will tell.a given user which approach works better for her.

Applications that invoke MetaCrawler (or any comparable meta-engine) are on a still higher level of the food chain, analogous presumably to carnivores that feed off other lower-level carnivores. An example {Etzioni, ibid] is Ahoy!, a “softbot” (short for “software robot”) that finds the home pages of individuals, given their name and affiliation. In effect, it is a “White Page” service. Ahoy! uses specialized knowledge about the “geography” and nomenclature conventions of the Web, e.g., that the host name of a corporate home page will usually end with “name.com,” where “name” is the name or acronym of the institution; hence, it may try “www.go2net.com” as the home page of a company named “go2net.” Similarly, given the name “Smith” at UMBC’s Computer Science department, it may try “http://www.umbc.edu/smith.” It may learn by relevance feedback that the computer science department uses the prefix “cs” in its host names; thereafter, it will try “Nicholas” in the Computer Science department successfully as “http://www.cs.umbc.edu/~nicholas/.” Subsequently, it may learn by similar feedback that all home pages at the University of Washington are in a “homes” directory. Thereafter, given “etzioni” in the Computer Science department at the University of Washington, it will successfully try “http://www.cs.washington.edu/homes/etzioni.” Ahoy! also makes use of its knowledge of home page format, e.g., the knowledge that a home page title for an individual is likely to contain the individual’s last name. Plainly, Ahoy! could run on top of any individual Web IR engine, e.g., InfoSeek, but by “feeding” on WebCrawler, it increases its Web coverage, and hence its chances of finding the desired home page.

Moukas and Maes [AAMAS, 1998] provide an alternative approach to meta-retrieval on the Web, Amalthaea. Like MetaCrawler, Amalthaea feeds off existing Web IR engines rather than indexing the Web itself. However, it differs from MetaCrawler in several important ways:

1. A single Metacrawler engine provides mappings to a variety of Web IR engines. The commercial MetaCrawler is centralized; its creators envision a situation where each user could have her own copy of MetaCrawler, but even in that view, each copy of WebCrawler maps from a single query interface presented to the user into parallel queries issued to the selected Web IR engines. By contrast, Amalthaea is a community of agents, serving a given user. There are two classes of agents: Information Filtering Agents (IFAs) and Information Discovery Agents (IDAs). Each selected Web IR agent has one or more IDAs assigned to it. Each user has one or more IFAs assigned to her.

2. The Amalthaea user does not formulate queries as the MetaCrawler does. Essentially, all Amalthaea queries are “by example.” The pages exemplifying the user’s interests can be specified explicitly by the user, e.g., she can point Amalthaea to a page while browsing, or submit a list of favorite URLs (bookmarks in Web browser parlance), etc. If the browser maintains a history file of pages the user has visited, Amalthaea can check this file to determine patterns of interest, e.g., sites or pages frequently visited. From these examples, Amalthaea generates a profile, representing the given user’s interests. Currently, the profile is a set of weighted keyword vectors, each vector representing one of the interests. (The weights are based on the familiar tf*idf formula.) Amalthaea generates an initial IFA representing each interest, containing the corresponding keyword vector. IFAs issue keyword vector requests based on the interests they represent. IDAs

accept requests from IFAs, and translate them into queries for "their" respective Web IR engines. Each IDA knows how to issue queries to "its" assigned Web IR engine. An IDA returns retrieved pages to the IFA that issued the corresponding request. The IFA filters retrieved pages by computing the cosine similarity between its vector, and the vector of each retrieved page. If a page passes the IFAs filter, the IFA presents the user with a "digest" of the page, accompanied by a confidence factor (based on the similarity computation) indicating how confident the IFA is that the user will like the given page.

3. The biggest difference between Amalthaea and MetaCrawler is that Amalthaea is based on an artificial life/evolutionary paradigm. The population of IFAs and IDAs evolves, driven partly by random changes (mutation, crossover), and partly by the user's judgment of how well the pages she receives match her current interests. Pages are rated on a scale from one to seven. If a page is highly rated, the IFA that chose to pass it to the user, and the IDA that retrieved it and passed it to the IDA, receive "credit." The amount of credit an agent receives for a given document is proportional to its confidence that the user will like the document. If the user rejects the document, the agents get negative credit, again proportional to the confidence level. High-ranking agents (in evolutionary terms, agents possessing high "fitness") get to reproduce. Low-ranking agents are purged. Mutation and crossover ensure that there will always be some "new" agents. This increases the system's ability to explore parts of the Web that may have been previously neglected, and to track changes in the user's interests. An agent may be new either by having new keywords in its vector (its genotype, in evolutionary terms), or new weights on existing keywords.

16.4.5 Personal Assistants for Web Browsing

The Web IR engines and meta-engines described in preceding sections index each page discovered as a separate entity. The indexes they generate (or in the case of the meta-engines, feed off) reflect only to a very limited degree the hyperlink structure of the Web. Of course, the robot indexers necessarily traverse as much of the Web as they can. But the index descriptors generated for a given page do not reflect the Web structure in which the page is embedded, except to the limited degree that descriptors may be extracted from hyperlink references and URLs within the given page, and the URL of the given page. Nor do they reflect the experience of the many users who may have browsed a given area of the Web (except that some engines may provide lists, or even human reviews, of popular sites, i.e., sites that have been visited frequently).

A different class of engine, exemplified by WebWatcher [Armstrong et al., AAAI, 1995], attempts to guide the user as she browses the Web, suggesting at each page she reaches, where she ought to go next given her stated interests and the experience of other previous users.

WebWatcher monitors the user as she browses. It does this by modifying each URL in each page the user reaches, so that selecting a reference causes a jump to the WebWatcher server rather than to the selected reference. WebWatcher then records the reference before completing the hyperjump the user requested. WebWatcher also modifies the page as seen by the user. In particular, the page is modified to highlight existing references that WebWatcher thinks will be particularly interesting to the user given her stated goals, adds additional "suggested references" that WebWatcher thinks will be of interest to the user either because of her stated interests or because of their similarity to the current page, and adds a menu that allows the user to specify that the page is

interesting, that the user has reached her goal (that is, found the information for which she is looking), that she would like to see similar pages, or that she gives up.

The essential feature of WebWatcher that enables it to act as a "tour guide" is its ability to learn what pages would be good references for a given user, with stated goals, when she reaches a given page. WebWatcher learns by recording the choices of previous users when they reached the same page with similar goals. This set of previous choices becomes the "learning set." Effectively, WebWatcher asks the question, "What hyperlink selection would best satisfy the user's goal now that she has reached the current page?" or replaces it by the question, "What hyperlink is the user most likely to select, given her present goal, when she reaches the current page?" The latter question is one for which WebWatcher can be trained, given the previous experience (monitored by WebWatcher) of other users, including of course the current user if she has come this way before.

Over the course of WebWatcher research history, the researchers have tried a number of predictive algorithms. One algorithm is based on the idea that two pages *P1* and *P2* are likely to be closely related if the pages that contain references to *P1* are likely to contain references to *P2* as well. A mutual information measure was used to measure the similarity between references to *P1* and references to *P2*. Computing such a measure assumes that the topology of the Web with regard to the given pages is known, at least with regard to some well-explored or well-known area of the Web, e.g., a University web site.

Another approach used in WebWatcher is to represent each possible user choice as a traditional IR term vector. The terms for a given hyperlink in a given page and a given user goal are selected from (a) words contained in or describing the hyperlink (note that this includes not only the underlined words that the user sees, it also includes words in the goals of previous users who selected the given hyperlink), (b) words entered by the user to describe her goal at the start of a session, and (c) words contained in the given page. (In one reported example, the page words are obtained from the sentence (if any) containing the hyperlink, and words in headings (if any) enclosing the hyperlink.) The term vector for each hyperlink can be reduced to a number in various ways. One reported way is to assign a weight to each term in a given vector based on the familiar *tf*idf* weighting scheme. In this way, a term vector can be generated for each instance where a user reaches the given page and selects a hyperlink, either selecting or not selecting the given link. Two prototype vectors can be generated, a positive prototype by adding up all the vectors corresponding to cases where the user selected the given link, a negative prototype by adding up all the vectors corresponding to cases where the user did not select the given link. Each instance of user selection or non-selection is then evaluated by taking the cosine similarity of the instance vector and the positive prototype vector, the cosine similarity of the instance vector and the negative prototype vector, and then taking the difference of the two cosine values. The result is a numeric value for each hyperlink in the given page, and the given user goal. The hyperlinks can then be ranked by these numeric values, and the highest ranking hyperlink(s) suggested to the user. (Other methods of evaluating the instance vector for each page/hyperlink/goal combination have been tried. For example, the conditional probability that a link will be followed given that a certain word occurs in the instance vector can be estimated as the ratio of the number of occurrences of the word when the given hyperlink is selected divided by the total number of occurrences of the word. If these single-word probability estimates are assumed to be independent, then

they can easily be combined to compute the probability that the given link will be followed. Again, the highest probability link(s) can be suggested to the user.)

A third approach tried by the WebWatcher researchers is Reinforcement Learning (RL). Instead of learning the value of a hyperlink from the *tf*idf* values of keywords describing the goals of previous users who selected it (as in the previous approach), the RL method computes its value in terms of the *tf*idf* values of goal keywords encountered in pages reached, directly or indirectly, by selecting the given hyperlink, and the number of hyperlink jumps required to reach them. In other words, WebWatcher learns the value of a hyperlink from the successes or failures of users who selected it. However, success or failure is computed, not by user feedback, but automatically in terms of the *tf*idf* values of the goal words encountered as a result of selecting the given hyperlink, and the number of pages (i.e., hyperlink jumps) traversed to reach a page containing (some of) the goal keywords. A hyperlink that will get the user to a very good page (as measured in terms of the *tf*idf* values of the goal keywords it contains) in five jumps may be valued lower than another hyperlink that gets the user to a pretty good page in two jumps. Moreover, the value of each keyword that can be reached from the given hyperlink is measured separately. So, if the given hyperlink *H1* in page *P1* (pointing to page *P2*) can reach keyword 1 via the path {*P2*, *P3*}, keyword 2 via path {*P2*, *P4*}, and keyword 3 via path {*P2*, *P5*}, *H1* may rate higher than another hyperlink *H2* in *P1* that reaches a single page *P6* containing all three keywords with the same *tf*idf* values as in *P3*, *P4*, and *P5* respectively, but via five hyperlink jumps. Or putting it another way, if a third hyperlink *H3* can reach *P6* in two jumps {*P7*, *P6*}, *H3* may not be ranked any higher than *H1*. In other words, WebWatcher's RL algorithm does not appear to value higher a page containing *N* keywords than *N* pages, reachable via the same number of jumps, each containing one of the *N* keywords (assuming that the "*tf*idf*" values are the same). This seems to run counter to the usual practice in information retrieval, but of course could easily be changed. (The value of a hyperlink *is* increased if the goal keywords are in the hyperlink itself.)

WebWatcher learns relative hyperlink values for a given Web locale. For example, since it was developed at CMU, it has learned the CMU locale. On the other hand, a "personal WebWatcher" has also been developed that learns the preferences of a given an individual.

The existing WebWatcher only asks the user for her goals at the beginning of a session. As WebWatcher researchers point out, a more sophisticated version could engage in an ongoing dialogue with the user as she browses.

18.0 Acknowledgments

The author wishes to thank Claudia Pearce, Jonathan Cohen, and Ellen Voorhees for their encouragement, meticulous review, and many helpful comments, suggestions, and corrections which greatly improved the quality of this paper. The author also wishes to thank Charles Nicholas for his encouragement, and for recommending this paper for publication. Most important of all, he wishes to thank his wife, Gail, for the gift of her love, devotion, and encouragement.

Bibliography

Aalbersberg, I.J. Incremental Relevance Feedback, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 11-22, 1992.

Anick, P.J. Adapting a full-text information retrieval system to the computer troubleshooting domain, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 349-358, 1994.

Allan, J. Relevance feedback with too much data, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 337-343, 1995.

ANSI/NISO Z39.50-1995. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*, July 1995.

Armstrong, R., Freitag, D. Joachins, T., Mitchell, T. WebWatcher: A Learning Apprentice for the World Wide Web, *1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, Stanford, March 1995.

Bartell, B.T., Cottrell, G.W., Belew, R.K. Latent Semantic Indexing is an optimal special case of Multidimensional Scaling, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 161-167, 1992.

Bartell, B.T., Cottrell, G.W., Belew, R.K. Automatic combination of multiple ranked retrieval systems, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 173-181, 1994.

Bauer, E., Kohavi, R. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, Machine Learning, 36, pp. 105-139, 1999.

Belkin, N.J., Cool, C., Croft, W.B., Callan, J.P. The effect of multiple query representations on information retrieval system performance, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.339-346, 1993.

Belkin, N.J., Croft, W.B. Retrieval Techniques, *Annual Review of Information Science and Technology (ARIST)*, Volume 22, Elsevier Science Publishers B.V., 1987.

Bishop, Y. M. M., Fienberg, S.E., Holland, P.W. Discrete multivariate analysis: Theory and practice, MIT Press, Cambridge, MA, 1975.

Belkin, N.J., Croft, W.B. Information filtering and information retrieval: Two sides of the same coin?, *Communications of the ACM,* Vol35, No. 12, December 1992.

Berry, M.W., Dumais, S.T., O'brien, G.W. Using linear algebra for intelligent information retrieval, *SIAM Review,* Vol. 37, No. 4, pp. 573-595, December 1995.

Borgman, C.L., Siegfried, S.L. Getty's Synoname and its Cousins: A survey of Applications of Personal Name-Matching Algorithms, *Journal of the American Society for Information Science*, 43(7), pp. 459-476, 1992.

Breiman, L. Bagging Predictors, *Machine Learning*, 26, No.2, pp. 123-140, 1996.

Breiman, L Bias, variance and arcing classifiers, Technical Report 460, UC-Berkeley, at Berkeley, CA., 1996.

Buckley, C., Allan, J., Salton, G. Automatic routing and ad-hoc retrieval using SMART: TREC 2, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 45-55, March 1994.

Buckley, C., Salton, G. Optimization of relevance feedback weights, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 351-357, 1995.

Buckley, C., Salton, G., Allan, J. The effect of adding relevance information in a relevance feedback environment, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 292-300, 1994.

Buckley, C., Singhal, A., Mitra, M. New retrieval approaches using SMART: TREC 4, *Text REtrieval Conference-4*, Gaithersburg, MD, National Institute of Standards and Technology, November 1-3, 1995.

Buckley, C., Singhal, A., Mitra, M. Using Query Zoning and Correlation within SMART: TREC 5, *Text REtrieval Conference-5*, Gaithersburg, MD, National Institute of Standards and Technology, November 20-22, 1996.

Callan, J.P. Passage-level evidence in document retrieval, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 302-310, 1994.

Callan, J.P., Croft, W.B., Broglio, J. TREC and TIPSTER experiments with INQUERY, *Information Processing & Management*, Vol. 31, No. 3, pp. 327-343, 1995.

Callan, J.P., Croft, W.B., Harding, S.M. The INQUERY retrieval system, in *Database and Expert Systems Applications: Proceedings of the International Conference*, Valencia Spain, pp. 78-83, 1992.

Callan, J.P., Lu, Z., Croft, W.B. Searching distributed collections with inference networks, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 21-28, 1995.

Chang, Y.K., Cirillo, C., Razon, J. Evaluation of feedback retrieval using modified freezing, resid-

ual collection, and test and control groups, chapter 17, pp. 355-370, in *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice Hall, Inc., 1971.

Chakravarthy, A.S., Haase, K.B. NetSerf: Using semantic knowledge to find internet archives, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4-11, 1995.

Charniak, E. *Statistical Techniques for Natural Language Parsing*, 1997.

Chen, A. A comparison of Regression, Neural Net, and Pattern Recognition Approaches to IR, Seventh International Conference on Information and Knowledge Management, pp. 140-147, 1998.

Cohen, J. Highlights: Language- and Domain-Independent Automatic Indexing Terms for Abstracting, *Journal of the American Society for Information Science*, 46(3), pp. 162-174, 1995.

Cole, R., Eklund, P. Analyzing an Email Collection Using Formal Concept Analysis, *3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, 1999.

Cole, R., Eklund, P.W. Scalability in Formal Concept Analysis, *Computational Intelligence,* Volume 15, Number 1, pp.11-27, Blackwell Publishers, Oxford, 1999.

Cole, R., Eklund, P.W. Analyzing an Email Collection using Formal Concept Analysis, *Proceedings of the Knowledge and Data Discovery Conf. (KDD '99)* Industry Track Paper, 1999.

Cooper, W.S. Expected search length: A single measure of retrieval effectiveness based on weak ordering action of retrieval systems, *Journal of the American Society for Information Science*, 19, pp. 30-41, 1968.

Cooper, W.S. Inconsistencies and misnomers in probabilistic IR, *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 57-61, 1991.

Cooper, W.S. The formalism of probability theory in IR: A foundation or an encumbrance?, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 242-247, 1994.

Cooper, W.S. Some inconsistencies and misidentified modeling assumptions in probabilistic information retrieval, *ACM Transactions on Information Systems,* Vol. 13, No. 1, pp. 100-111, January 1995.

Cooper, W.S., Chen, Aitao, Gey, F.C Full text retrieval based on probabilistic equations with coefficients fitted by logistic regression, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 57-66, March 1994.

Cooper, W.S., Gey, F.C., Dabney, D.P. Probabilistic retrieval based on staged logistic regression, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.198-210, 1992.

Cowie, J., Lehnert, W. Information Extraction, *Communications of the ACM*, Vol. 39, No. 1, pp. 80-91, January 1996

Crestani, F., van Rijsbergen, C.J. Probability kinematics, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 291-299, 1995.

Crochemore, M., Rytter, W. *Text Algorithms*, pp.73-104, Oxford University Press, 1994.

Croft, W.B., Turtle, H.R., Lewis, D.D. The uses of phrases and structured queries in information retrieval, *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 32-45, 1991.

Cutting, D.R., Karger, D.R., Pederson, J.O., Tukey, J.W. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.318-329, 1992.

Cutting, D.R., Karger, D.R., Pederson, J.O. Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.126-134 1993

Damashek, M. Gauging similarity with n-grams: Language-independent categorization of text, *Science*, Volume 267, pp. 843-848, February 1995.

Date, C.J. *An introduction to database systems*, Addison-Wesley, Reading, Mass, 1981.

DARPA. *Proceedings of the 4th Message Understanding Conference (MUC-4)* Mclean, Va., Morgan Kaufmann, 1992.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R. Indexing by latent semantic analysis, *Journal of the American Society for Information Science*, 41(6), pp. 391-407, 1990.

DIALOG, A Quick Tour.

Dumais, S., Platt, J., Heckerman, D., Sahami, M. Inductive Learning Algorithms and Representations for Text Categorization, *Proceedings of the Seventh International Conference on Information and Knowledge Management (ACM CIKM)*, pp. 148-155, 1998.

Ebert, D. Realistic Interactive Visualization and Perceptual Cues for Information Visualization, *Workshop on New Paradigms in Information Visualization and Manipulation*, in conjunction with The Fourth International Conference on Information and Knowledge Management (CIKM'95),

Baltimore, MD., December 2, 1995.

Ebert, D.S., Shaw, C.S., Zwa, A., Miller, E.L., Roberts, D.A. Two-Handed Volumetric Document Corpus Management, *IEEE Computer Graphics and Applications*, July 1997.

Eklund, P., Wille, R. Text Data Mining and Discourse Analysis, *Small ARC Proposal*, 1999.

Efthimiadis, E.N. User Choices: A new yardstick for the evaluation of ranking algorithms for interactive query expansion, *Information Processing & Management*, Vol. 31, No. 4, pp. 605-620, 1995.

El-Handouchi, A., Willett, P. Techniques for the measurement of clustering tendency in document retrieval systems, *Journal of Information Science*, 13, pp 361-365, 1987.

Etzioni, O. Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 1322-1326, AAAI Press, Menlo Park, Calif., MIT Press, Cambridge, Mass., 1996.

Evans, D.A., Lefferts, R.G. Design and evaluation of the CLARIT-TREC-2 system, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 137-150, March 1994

Faloutsos, C., Oard, D.W. A survey of information retrieval and filtering methods, Technical Report CS-TR-3514, University of Maryland, College Park, MD, August 1995.

Feldman, S. Comparing DIALOG, TARGET, and DR-LINK, *ONLINE*, November 1996.

Florance, V. Information processing in the context of medical care, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 158-163, 1995.

Foltz, P.W., Dumais, S.T. Personalized information delivery: An analysis of information filtering methods, *Communications of the ACM*, Vol. 35, No. 12, pp. 51-60, December 1992.

Fox, E.A., Shaw, J.A. Combination of multiple sources, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 243-252, March 1994.

Friedman, J.H.On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality, Data Mining and Knowledge Discovery, Volume 1, Number 1, pp. 55-77, 1996.

Greiff, W.R., Croft, W.B., Turtle, H. Computationally Tractable Probabilistic Modeling of Boolean Operators, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 119-128, 1997.

Guthrie, J., Guthrie, L., Wilks, Y., Aidinejad, H. Subject-Dependent Co-Occurrence and Word Sense Disambiguation, *Proceedings of the 29th Annual Meeting of the ACL*, pp. 146-152, Berkley, California, USA, 1991.

Guthrie, L. A Note on Lexical Disambiguation. In *Corpus-Based Computational Linguistics*, Editors Clive Souter and Eric Atwell, pp. 227-238, Published by Rodopi, B.V., Amsterdam, 1993.

Hahn, H. *The Internet Complete Reference, 2nd Ed*., Osborne Mcgraw-Hill, Berkeley, 1996.

Haines, D., Croft, W.B. Relevance Feedback and Inference Networks, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 2-11, 1993.

Harman, D. User-friendly systems instead of user-friendly front-ends, *Journal of the American Society for Information Science,* 43(2), pp 164-174, 1992.

Harman, D. Relevance feedback revisited, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.1-10, 1992.

Harman, D. Overview of the Second Text REtrieval Conference (TREC-2), *Text REtrieval Conference-2*, Gaithersburg, MD, National Institute of Standards and Technology Special Publication 500-215, March, 1994.

Harman, D. Overview of the Third Text REtrieval Conference (TREC-3), *Text REtrieval Conference-4*, Gaithersburg, MD, National Institute of Standards and Technology Special Publication 500-225, April, 1995.

Hayes, P.J., Weinstein, S.P. Construe-TIS: A System for Content-Based Indexing of a Database of News Stories, 2nd Annual Conference on Innovative Applications of Artificial Intelligence, pp. 48-64, AAAI Press, Menlo Park, Calif., 1990.

Hearst, M.A. Tilebars: Visualization of term distribution info in full text info access, in *Proc. ACM SIGCHI Conf. on Human factors in computing systems*, pp 59-66,1995.

Hearst, M., Pedersen, J., Pirolli, P. Schutze, H. Xerox TREC4 Site Report, *Text REtrieval Conference-4*, Gaithersburg, MD, National Institute of Standards and Technology, November 1-3, 1995.

Hearst, M.A., Plaunt, C. Subtopic Structuring for Full-Length Document Access, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.59-68, 1993.

Hersch, W.R., Elliot, D.L., Hickam, D.H., Wolf, S.L., Molnar, A., Lechtenstien, C. Towards new measures of information retrieval evaluation, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 164-170, 1995.

Hoel, P.G. *Introduction to Mathematical Statistics,* 4th ed, John Wiley and Sons, New York, 1971.

Hull, D. Improving text retrieval for the routing problem using Latent Semantic Indexing, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.282-291, 1994.

Hull, D.A., Pedersen, J.O., Schutze, H. Method Combination for Document Filtering, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.279-287, 1996.

INQUERY 3.0, *Applied Computing Systems Institute of Massachusetts, Inc. (ACSIOM), Center for Intelligent Information Retrieval, University of Massachusetts Computer Science Department*, Amherst, Massachusetts, 1995.

Jacqemin, C., Royaute, J. Retrieving terms and their variants in a lexicalized unification-based framework, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 132-141, 1994.

Joachims, T., Mitchell, T., Freitag, D., Armstrong, R. WebWatcher: Machine Learning and Hypertext, in *GI Fachgruppentreffen Maschinelles Lernen, K. Morik, J.Herrmann, eds.*, Dortmund, Germany, August 1995.

Joachims, T. Freitag, D., Mitchell, T. A Tour Guide for the World Wide Web, *Proceedings of IJCAI97*, August 1997.

Katzer, J., McGill, M.J., Tessier, J.A., Frakes, W. Dasgupta, P. A study of the overlap among document representations, *Information Technology: Research and Development*, I, pp. 261-274, 1982.

Korfhage, R.R. *Information Storage and Retrie*val, John Wiley and Sons, New York, 1997.

Kretser, O.d., Moffat, A. Effective Document Presentation with a Locality-Based Similarity Heuristic, *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 113-120, 1999.

Kroll, E. *The Whole Internet: User's Guide and Catalogue*, O'Reilly and Associates, Sebastopol, CA, 1992.

Kupiek, J. MURAX: A robust linguistic approach for question answering using an on-line encyclopedia, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 181-190, 1993.

Leacock, C., Towell, G., Voorhees, E.M. Toward Building Contextual Representations of Word Senses Using Statistical Models, *Corpus Processing for Lexical Acquisition*, pp. 97-113, The MIT Press, Cambridge, Massachusetts, 1996.

Lee, J.H. Properties of extended boolean models in information retrieval, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information*

Retrieval, pp. 182-190, 1994.

Lee, J.H. Combining multiple evidence from different properties of weighting schemes, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 180-188, 1995.

Lee, J.H, Analyses of Multiple Evidence Combination, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 267-276, 1997.

Lenat, D.B., Guha, R.V. *Building Large Knowledge-Based Systems: Representations and Inference in the Cyc Project*, Addison-Wesley, Reading, Mass., 1989.

LeVan, R. Building a Z39.50 client, 1995.

Lewis, D.D. An evaluation of phrasal and clustered representations on a text categorization task, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 37-50, 1992.

Lewis, D.D. Evaluating and optimizing autonomous text classification systems, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 246-254, 1995.

Lewis, D.D., Spark-Jones, Karen. Natural Language Processing for Information Retrieval, *Communications of the ACM*, Vol. 39, No. 1, pp. 92-101, January 1996.

LEXIS/NEXIS Quick Reference.

Liddy, E.D. Discourse Level Structure of Abstracts, *Proceedings of the 50th ASIS Annual Meeting*, pp. 138-147, Learned Information, Inc., Medford, NJ, 1987.

Liddy, E.D. Structure of Information in Full-Text Abstracts, Proceedings of RIAO Conference, 1988.

Liddy, E.D. Enhanced Text Retrieval Using Natural Language Processing, *Bulletin of the American Society for Information Science*, Vol. 24, No. 4, 1998.

Liddy, E.D. Personal Communication, 1999.

Liddy, E. D., Myaeng, S.H. DR-LINK: A System Update for TREC 2, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 45-55, March 1994.

Liddy, E.D., Paik, W., Yu, E.S. Text Categorization for Multiple Users Based on Semantic Features from a Machine-Readable Dictionary, ACM Transactions on Information Systems, Vol. 12, No. 3, pp.278-295, July 1994.

Liddy, E.D., Paik, W., McKenna, M.,Yu, E.S. A Natural Language Text Retrieval System with Relevance Feedback, *Proceedings of the 16th National Online Meeting*, 1995.

Liddy, E.D., Paik, W., Yu, E.S., McKenna, M. Document Retrieval Using Linguistic Knowledge, Proceedings of RIAO Conference, Rockefeller University, NY, 1994.

Liddy, E.D. Textwise - KNOW-IT, White Paper, 1999.

Mann, W.C., Thompson, S.A. Rhetorical Structure Theory: Toward a functional theory of text organization, *Text*, 8(3), pp. 243-281, 1988.

Marcu, D. Building up Rhetorical Structure Trees, *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI*, pp 1069-1074, 1996.

Marcu, D. Personal Communication, 1999.

Mahesh, K. HyperText Summary Extraction for Fast Document Browsing, *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, Stanford University, pp95-103, 1997.

Martin, P., Eklund, P. Embedding Knowledge in Web Documents, *The Eighth International World Wide Web Conference*, May 1999.

Maze, S., Moxley, D., Smith, D.J. *Authoritative Guide to Web Search Engines*, Neal-Schuman Publishers, Inc., New York, 1997.

McGill, M., Koll, M., Norreault, T. An evaluation of factors affecting document ranking by information retrieval systems. *Syracuse, Syracuse University School of Information Studies*, 1979.

Miller, G.A. WordNet: An on-line lexical database, *International Journal of Lexicography*, 3(4), 1990.

Milic-Frayling, N., Lefferts, R., Evans, D.A. CLARIT TREC-4 report on ad-hoc experiments, *Text REtrieval Conference-4*, Gaithersburg, MD, National Institute of Standards and Technology, November 1-3, 1995.

Mooers, C.N. Zatocoding applied to mechanical organization of knowledge, *American Documentation*, 2, pp. 20-32, 1951.

Moukas, A., Maes, P. Amalthaea: An evolving Multi-Agent Information Filtering and Discovery System for the WWW, *Autonomous Agents and Multi-Agent Systems*, 1, pp. 59-88, 1998.

Mann, W.C., Thompson, S.A. Rhetorical Structure Theory: Toward a Functional Theory of Text Organization, *Text*, 8(3), pp. 243-281, 1988.

Marcu, D. Building Up Rhetorical Structure Trees, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI*, pp. 1069-1074, 1996.

Myaeng, S.H., Lopez-Lopez, A. Conceptual Graph Matching: A flexible algorithm and experiments, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 4, pp. 107-126, 1992.

Nelson, M.R. Fast String Searching With Suffix Trees, *Dr.Dobb's Journal*, pp. 115-119, August 1996.

Ogden, W. C., Bernick, P. Using Natural Language Interfaces, *Handbook of Human-Computer Interaction, Second completely revised edition*, CH7, pp. 137-161, Elsevier Science B.V., 1997.

Onyshkevych, B. Personal communication.

Opitz, D., Maclin, R. Popular Ensemble Methods: An Empirical Study, J*ournal of Artificial Intelligence Research* 11, pp. 169-198.

Paice, C.P. Soft Evaluation of boolean search queries in information retrieval systems, *Information Technology: Research and Development*, 3(1), pp. 33-42, 1984.

Paik, W., Liddy, E.D. Interpretation of Proper Nouns for Information Retrieval, *Proceedings of the ARPA Workshop on Human Language Technology*, pp. 309-313, Princeton, N.J., 1993.

Paik, W., Liddy, E. D., Yu, E., McKenna, M. Categorizing and Standardizing Proper Nouns for Efficient Information Retrieval, *Corpus Processing for Lexical Acquisition*, pp. 62-73, MIT Press, Cambridge, Mass., 1996.

Pearce, C., Nicholas, C. TELLTALE: Experiments in a dynamic hypertext environment for degraded and multilingual data, *Journal of the American Society for Information Science*, 47(4), pp. 263-275, 1996.

Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA, 1988.

Porter, M. An Algorithm for Suffix Stripping, *Program*, 14(3), pp. 130-137, 1980.

Porter, M. An Algorithm for Suffix Stripping, in *Readings in Information Retrieval*, Sparck Jones and Willett, eds., pp. 313-316, 1997.

Procter, P. (ed.). Longman Dictionary of Contemporary English, Harlow:Longman, 1978.

Quick Reference News/Retrieval for Windows

Raghavan, V.V., Sever, H. On the reuse of past optimal queries, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 344-350, 1995.

Riloff, E. Little words can make a big difference for text classification, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 130-136, 1995.

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

Robertson, S.E., Walker, S. Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval, Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 222-241, 1994.

Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M. Okapi at TREC-3, *Overview of the Third Text REtrieval Conference (TREC-3)*, NIST Special Publication 500-225, National Institute of Standards and Technology, Gaithersburg, MD, pp. 109-126, April 1995.

Robertson, S.E., Walker, S., Beaulieu, M.M., Gatford, M., Payne, A. Okapi at TREC-4, *The Fourth Text REtrieval Conference (TREC-4)*, NIST Special Publication 500-236, National Institute of Standards and Technology, Gaithersburg, MD, pp. 73-86, October 1996.

Robertson, S.E., Walker, S. On Relevance Weights with Little Relevance Information, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 16-24, 1997

Robertson, S.E., Walker, S. Microsoft Cambridge at TREC-9: Filtering track, *The Ninth Text REtrieval Conference (TREC-9)*, National Institute of Standards and Technology, Gaithersburg, MD, pp. 73-86, November 13-16, 2000.

Robot FAQ. http://info.webcrawler.com/mak/projects/robots/faq.html#what

Rocchio, J.J. *Document Retrieval Systems - Optimization and Evaluation,*. PhD thesis, Harvard, 1966.

Roussinov, D., Tolle, K., Ramsey, M., Hsinchun, C. Interactive Internet Search through Automatic Clustering: an Empirical Study, *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 289,290, 1999.

Salton, G. A simple blueprint for automatic boolean query processing, *Information Processing & Management,* Vol. 24, No. 3, pp. 269-280, 1988.

Salton, G. *Automatic text processing: The transformation, analysis, and retrieval of information by computer,* Addison-Wesley, Reading, MA, 1989.

Salton, G., Allan, J., Buckley, C. Approaches to Passage Retrieval in Full Text Information Systems, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 49-58, 1993.

Salton, G., Buckley, C. Term-weighting approaches in automatic text retrieval, *Information Processing & Management*, 24(5), pp. 513-523, 1988.

Salton, G., Buckley, C. Improving retrieval performance by relevance feedback, *Journal of the American Society for Information Science*, 41(4), pp. 288-297, 1990.

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

Salton, G., McGill, M.J. *Introduction to Modern Information Retrieval*, McGraw Hill Publishing Company, New York, 1983.

Saracevic, T. Evaluation of Evaluation in Information Retrieval, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 137-146, 1995.

Saracevic, T. RELEVANCE: A Review of and a Framework for the Thinking on the Notion in Information Science, *Readings in Information Retrieval*, Sparck Jones, K., Willett, P. Eds., Morgan Kaufmann Publishers, Inc., pp. 143-165, 1997.

Saracevic, T., Kantor, P. A study of information seeking and retrieving. III Searchers, searches and overlap, *Journal of the American Society for Information Science*, 39, 3, pp. 197-216, 1988.

Sanderson, M. Word sense disambiguation and information retrieval, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 142-151, 1994.

Schapire, R.E., Singer, Y., Singhal, A. Boosting and Rocchio Applied to Text Filtering, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 215-223, 1998.

Schutze, H., Hull, D.A., Pederson, J.O. A comparison of classifiers and document representations for the routing problem, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.229-237, 1995.

Schutze, H., Silverstein, C. A Comparison of Projections for Efficient Document Clustering, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 74-81, 1997.

Selberg, E., Etzioni, O. Multi-Engine Search and Comparison Using the MetaCrawler, Proceedings of the Fourth International Conference on the World Wide Web, pp. 195-208, 1995.

Shaw, W.M. Term-relevance computations and perfect retrieval performance, *Information Processing & Management*, Vol. 31, No. 4, pp. 491-498, 1995.

Shivakumar, N., Garcia-Molina, H. Finding Near-Replicas of Documents on the Web, *Proceedings of Workshop on Web Databases (WebDB'98) held in conjunction with EDBT'98*, March 1998.

Shivakumar, N., Garcia-Molina, H. SCAM: A Copy Detection Mechanism for Digital Documents, *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, Austin, Texas, June 1995.

Sibson, R. SLINK: an optimally efficient algorithm for the single link cluster method, *Computer Journal*, 16, pp 30-34, 1973.

Silverstein, C., Pederson, J.O. Almost-Constant-Time Clustering of Arbitrary Corpus Subsets, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.60-66, 1997.

Singhal, A., Buckley, C., Mandar, M. Pivoted Document Language Normalization, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.21-29, 1996.

Singhal, A., Salton, G., Mitra, M., Buckley, C. Document length normalization, *Technical Report TR95-1529*, Cornell University, 1995.

Sowa, J.F. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, MA, 1984.

Sparck Jones, K. Search term relevance weighting given little relevance information, Journal of Documentation, 35(1), pp. 30-48, 1979.

Strzalkowski, T. Natural Language Information Retrieval, *Information Processing and Management*,1994.

Strzalkowski, T., Carballo, J.P. Recent developments in natural language text retrieval, *The Second Text REtrieval Conference (TREC-2)*, NIST Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD, pp. 123-136, March 1994.

Strzalkowski, T., Carballo, J.P., Marinescu, M. Natural Language Information Retrieval: TREC-3 Report, *Overview of the Third Text REtrieval Conference (TREC-3)*, NIST Special Publication 500-225, National Institute of Standards and Technology, Gaithersburg, MD, pp. 39-53, April 1995.

Suen, C.Y. N-gram statistics for natural language understanding and text processing, *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1* (2), pp. 164-172, 1979.

Topic, An Introduction to Topics V 1.0, Verity, 1995.

Type 102 Ranked List Query Preliminary Specification, October 27, 1995.

Turtle, H. Natural language vs. boolean query evaluation: A comparison of retrieval performance, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.212-220, 1994.

Turtle, H., Croft, W.B. Evaluation of an inference network-based retrieval model, *ACM Transactions on Information Systems,* Vol. 9, No. 3, July 1991.

Ukkonen, E. On-line Construction of Suffix Trees, Algorithmica, pp. 249-260, September 1995.

van Dijk, T.A. News as Discourse, Hillsdale Lawrence Erlbaum Associates, 1988.

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

van Rijsbergen, C.J. Towards an information logic, *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp.77-86, 1989.

van Rijsbergen, C.J., Sparck Jones, K. A test for the separation of relevant and non-relevant documents experimental retrieval collections, *Journal of Documentation,* 29, pp 251-257, 1973.

Veerasamy, A., Belkin, N.J. Evaluation of a Tool for Visualization of Information Retrieval Results, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 85-92, 1996.

Veerasamy, A., Heikes, R. Effectiveness of a graphical display of retrieval results, *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 236-245, 1997.

Viles, C.L., French, J.C. Dissemination of collection wide information in a distributed information retrieval system, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 12-20, 1995.

Vogt, C.C., Cottrell, G.W. Predicting the Performance of Linearly Combined IR Systems, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 190-196, 1998.

Voorhees, E. M. The Cluster Hypothesis Revisited. Technical Report *TR 85-658*, Cornell University, Ithaca, NY, 1985.

Voorhees, E.M. Using WordNet to disambiguate word senses for text retrieval, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 171-180, 1993.

Voorhees, E.M. Personal communication, January 1997.

Voorhees, E.M., Gupta, N.K., Johnson-Laird, B. Learning collection fusion strategies, *Proceed-*

ings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 172-179, 1995.

Waller, W.G., Kraft, D.H. A mathematical model of a weighted boolean retrieval system, *Information Processing and Management*, 15, pp.235-245, 1979.

Wilkinson, R. Effective Retrieval of Structured Documents, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 311-317, 1994.

Wille, R. Conceptual Graphs and Formal Concept Analysis, *Conceptual Structures: Theory, Tools, and Application*s, pp. 104-208, Springer Verlag, 1996.

Willett, P. Recent trends in hierarchic document clustering: A critical review, *Information Processing & Management*, Vol. 24, No. 5, pp. 577-597, 1988.

Winkler, R.L., Hays, W.L Statistics: *Probability, Inference, and Decision, 2nd ed*., Holt, Rinehart, and Winston, New York, 1975.

Witten, I.H., Eibe, F. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufman Publishers, San Francisco, Ca., 1999.

Xu, J., Callan, J. Effective Retrieval with Distributed Collections, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 112-120, 1998.

Xu, J., Croft, W.B. Query Expansion Using Local and Global Document Analysis, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.4-11, 1996.

Yang, Y. Expert Network: Effective and efficient learning from human decisions in text categorization and retrieval, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 13-22, 1994.

Yannakoudakis, E.J., Goyal, P., Huggil, J.A. The generation and use of text fragments for data compression, *Information Processing and Management*, 18, pp. 15-21, 1982.

Yanoff, S. *Internet Services List*, Internet Manuscript, ‘finger yanoff@csd4.csd.uwm.edu’, 1993.

Yarowsky, D. One sense per collocation, Proceedings of the Human Language Technology Workshop, 1993.

Yochum, J. Research in passage level routing with LMDS, *Text REtrieval Conference-4*, Gaithersburg, MD, National Institute of Standards and Technology, November 1-3, 1995.

Yu, K.L., Lam, W. A New On-Line Learning Algorithm For Adaptive Text Filtering, *Proceedings*

of the Seventh International Conference on Information and Knowledge Management (ACM CIKM), pp. 156-160, 1998.

Zamora, E.M., Pollock, J.J., Zamora, A. The use of trigram analysis for spelling error detection, *Information Processing and Management*, 17, pp. 305-316, 1981.

Zamir, O., Etzioni, O. Web Document Clustering: A Feasibility Demonstration, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 46-54, 1998.

Zimmerman, H.J. *Fuzzy Set Theory and its Applications*, 2md edition, Kluwer Academic Publishers, 1991.