Improving Browsing in Digital Libraries with Keyphrase Indexes

Carl Gutwin[1], Gordon Paynter[2], Ian Witten[2], Craig Nevill-Manning[3], Eibe Frank[2]

[1] Department of Computer Science, University of Saskatchewan
57 Campus Drive, Saskatoon, Saskatchewan S7N 5A9, Canada
gutwin@cs.usask.ca

[2] Department of Computer Science, University of Waikato
Private Bag 3105, Hamilton, NZ
[gwp,ihw,eibe]@cs.waikato.ac.nz

[3] Department of Computer Science, Rutgers University
Piscataway, New Jersey 08855, USA
nevill@cs.rutgers.edu

Abstract

Browsing accounts for much of people's interaction with digital libraries, but it is poorly supported by standard search engines. Conventional systems often operate at the wrong level, indexing words when people think in terms of topics, and returning documents when people want a broader view. As a result, users cannot easily determine what is in a collection, how well a particular topic is covered, or what kinds of queries will provide useful results.

We have built a new kind of search engine, Keyphind, that is explicitly designed to support browsing. Automatically-extracted keyphrases form the basic unit of both indexing and presentation, allowing users to interact with the collection at the level of topics and subjects rather than words and documents. The keyphrase index also provides a simple mechanism for clustering documents, refining queries, and previewing results. We compared Keyphind to a traditional query engine in a small usability study. Users reported that certain kinds of browsing tasks were much easier with the new interface, indicating that a keyphrase index would be a useful supplement to existing search tools.

Keywords

Digital libraries, browsing interfaces, text mining, keyphrase extraction, machine learning

Introduction

Digital libraries and searchable document collections are widely available on the Internet. Unfortunately, people still have difficulty finding what they want in these collections (e.g. [13,3]). In particular, users often find *browsing* to be frustrating. In browsing, the user interacts with a collection and carries out searches, without having in mind a specific document or document set [5]. For example, one may wish to ascertain what a collection contains, whether there is any material on a certain topic, or what kinds of queries are likely to produce good results (see Table 1). Such tasks are difficult to carry out in digital libraries—

not because the problems are inherently hard, but because the search systems that are provided do not support browsing very well.

Task type	Questions
Collection evaluation	What's in this collection? What topics does this collection cover?
Subject exploration	How well does this collection cover area X? What topics are available in area X?
Query exploration	What kind of queries will succeed in area X? How can I specialize or generalize my query?

Table 1. Three types of browsing and related task questions

Standard search engines use a full-text inverted index to find documents that match the user's query, and use a ranking algorithm to order the documents for presentation (e.g. [36,37]). Although this approach is powerful—users can retrieve a document by entering any word that appears in it—full-text indexing causes several problems when browsing. First, most standard engines index individual words only, whereas people often use short topic phrases when exploring a collection (e.g. *solar energy*, *programming by demonstration*, *artificial intelligence*)[1] [39,43]. Second, standard engines return lists of individual documents, but this level of presentation is usually too specific for users who are browsing [35,46]. Third, standard engines do not provide support for forming new queries, even though browsing involves an iterative process of exploration and query refinement [40].

We have built a new search engine—called Keyphind—that uses a single simple approach to address all of these problems. Keyphind is based on key phrases (similar to those provided by authors) that have been automatically extracted from the documents in a collection. While the use of phrases in information retrieval is not new, Keyphind goes beyond previous systems by making the phrase the fundamental building block for the entire system—including the index, the presentation and clustering of results, and the query refinement process. The keyphrase approach allows people to interact with a document collection at a relatively high level. Topics and subjects are the basic level of interaction, both when forming queries and when viewing and manipulating query results. Browsing is supported by showing the user what topics are available for a particular query, how many documents there are in those areas, and all possible ways that a query can be legitimately extended. In our initial usability studies, the keyphrase approach was seen to be more useful for browsing tasks than a standard query engine.

In the first part of this article we discuss browsing in more detail, and review previous efforts to address the drawbacks of standard search engines. Then we introduce the Keyphind system, outline the text-mining technique used to extract key phrases from documents, and describe the ways that users can interact with the system. Finally we evaluate the keyphrase approach, and report on a usability study that compared browsing in Keyphind with browsing in a traditional full-text query system.

---

[1] For example, the typical query to the Infoseek search engine (www.infoseek.com) is a noun phrase with an average length of 2.2 words [23].

Browsing document collections

When people interact with large document collections, only part of their activity involves specific searches for particular documents. Many tasks are better characterized as "browsing"—information-seeking activity where the goal is not a particular set of documents (e.g. [5,30]). For example, people may want to learn about a collection, what it contains, and how well it covers a particular topic; they may want to find documents in a general area, but without knowing exactly what they are looking for; or they may simply be looking for something "interesting." Browsing is an inherently interactive activity: when people browse, their direction and activities evolve as they work, and they may take several different routes to their final destination.

In bricks-and-mortar libraries, people browse by walking among the stacks, looking at displays set up by library staff, even perusing the reshelving carts to see what other people have taken out. Although digital libraries preclude these physical activities, people still browse; they just do it through whatever means they can—namely the collection's query interface. Browsing in digital libraries is characterized by a succession of queries rather than a single search—that is, people determine their next query partly based on the results of the previous one [30].

This paper focuses on three kinds of browsing tasks that we have informally observed in our experience with the New Zealand Digital Library (www.nzdl.org): evaluating collections, exploring topic areas, and exploring possible queries. Table 1 shows some of the questions that people have in mind when they undertake these tasks. The following sections describe the tasks, outline why traditional search engines make the tasks difficult, and review prior research in each area.

Collection evaluation

When a searcher has several different document collections to choose from, but not enough time to search all of them in detail, they must determine which collections are most likely to provide them with the information they need. This process of assessment often involves browsing—both to gain an overview of the collection and to determine if it contains material in the area of interest [30]. As an example of the number of collections available for a particular search, the following are a few of the possibilities for finding literature on *browsing interfaces*:

- The ACM Digital Library (www.acm.org/dl)
- Internet search engines (e.g. AltaVista, Infoseek, HotBot)
- The HCI Bibliography (www.hcibib.org)
- The NZDL Computer Science Technical Reports Collection (www.nzdl.org)
- The Cornell Computer Science Technical Reference Library (cs-tr.cs.cornell.edu)
- The INSPEC citation database (local access)
- The NIST TREC document collection (nvl.nist.gov)
- The UnCover citation database (uncweb.carl.org/)
- Local online library catalogues (e.g. library.usask.ca)

There are many ways that these and other collections can be compared and evaluated: for example, size, currency, and quality of information are a few possibilities. Perhaps the most important criterion in choosing between collections, however, is coverage—what is in the

collection and whether or not it is likely to contain the user's desired information. With physical collections (that is, books), people often evaluate coverage by scanning tables of contents or by looking up their topics in the index (e.g. [29]).

In digital collections that use standard search engines, however, ascertaining what the collection contains is difficult or impossible. Although most systems provide a brief description of the collection's contents (e.g. computer science technical reports), they rarely display the range of topics covered. The only way to find this information is by issuing general queries and scanning the lists of returned documents, a slow and arduous process.

Several systems and research projects have sought to address this problem by providing information about the top-level contents of a document collection. These approaches can be split into two groups: those that provide topic information manually, and those that determine it automatically. Manually-determined categories involve the classification of each document by a human indexer, and so are laborious to create and maintain. Examples of manual categorization can be seen in commercial systems such as the Yahoo! collection (www.yahoo.com), and in some research collections (e.g. [14]). In contrast, automatic techniques create high level overviews by algorithmic examination of the documents in the collection. A variety of techniques for gathering and displaying this information have been investigated. The most well-known of these is vector-space document clustering, where document similarities are calculated and used to group similar documents together (e.g. [26,37]). The process can be repeated until the desired level of generality is reached (although it is not immediately apparent how to choose meaningful names or labels for the clusters [26]). Other researchers have used self-organizing maps to show both the main topics in a collection and their relative frequencies. For example, Lin [29] creates graphical tables of contents from document titles, and Chen and colleagues [6] create map representations of a collection of web pages. Finally, when a collection has an internal structure (such as a hierarchical web site), this structure can be used to generate an overview (e.g. [22]).

Exploring topic areas

A second common browsing activity is the exploration of a subject area (e.g. [5,35]). When exploring, users may try to gain an understanding of the topics that are part of the area, may wish to gather contextual information for directing and focusing their search, or may simply hope to come across useful material. Exploration is often undertaken by novices and people who need to learn about a new area; but it may also be undertaken by people who have had a search failure and are interested in the kinds of queries that will definitely succeed.

This kind of browsing is greatly aided by being able to see the range of material available about the subject, since people can then both learn about the domain and use visual recognition rather than linguistic recall to direct their activity. As Chang and Rice [5] state, "browsing in computer systems is characterized by searching without specifying; it is a recognition-based search strategy" (p. 243).

Traditional search systems make exploration difficult for two reasons. First, subject areas are often described with multi-word phrases, but the word-level indexes used in many systems deal poorly with queries where query terms are meaningful in relation to one another [44]. For example, a standard system will have poor precision when asked about a subject area like "computer architecture," where the meaning of the phrase is different from either

component. Second, ranking algorithms are designed to return documents relevant to the query terms, not to show the range of documents that exist. A list of individual documents is usually far too specific for users who are trying to explore an area (e.g. [35,40]).

The first of these problems has previously been addressed by including phrases as well as individual words in the inverted index (e.g. [36,44]). However, indexing all phrases consumes considerable space, so researchers have also considered various means of selecting only "important" or content-bearing phrases for indexing (e.g. [12,18]). We discuss the issues involved in phrase-based indexing in more detail later in the article.

The second problem—that systems return a list of documents rather than a range of topics—has been addressed primarily through the use of document clustering. In this approach, the search engine returns a set of clusters that match a user's query rather than a set of documents. Again, clustering may be based on existing manual document classifications (e.g. Yahoo) or on automatically-extracted information. Automatic approaches are primarily based on vector-space based similarity [26]. Clusters are then presented to the user, either as a list (e.g. [1,11]) or in some graphical format (e.g. [19,45]). One of the better-known clustering applications that seeks to support browsing is the Scatter/Gather system [11,34]. This system uses a linear-time vector-space algorithm to dynamically cluster the documents resulting from a user's query. The user can then select one or more clusters for reclustering and further analysis.

Query exploration

Browsing is an activity where the user's queries are not determined beforehand, but evolve as the user interacts with the collection [5]. Therefore, part of the browsing process involves gathering information about how different kinds of queries will work in the search system, and "translating an information need into a searchable query" ([3], p. 493). People may wish to find out what kinds of queries will succeed, how a query can be specialized or generalized, or what will happen if an existing query is changed.

In a traditional search system, however, there is little support for query exploration. Users have to compose their queries without any assistance from the system, and there is no indication of how likely a query is to be successful or of what kind of results it will produce. It is also difficult to use a previous query as the basis for a new one: since the inner workings of a ranking algorithm are rarely made explicit, it is not easy to predict how changes to a query will affect the size or relevance of the result set. Compounding the difficulty, a concept may be referred to in the collection by any of several related terms (the vocabulary problem [7,17]), and users must somehow determine the correct one. Without guidance in these areas, users will often issue queries that return either nothing or a multitude of documents—the "feast or famine" problem (e.g. [14]).

Researchers have attempted to provide guidance for query exploration in several ways. Support for query refinement often presents the user with a set of terms related to their query terms (e.g. [9,48]); they can then add the suggestions to their query, either disjunctively (to expand the results) or conjunctively (to specialize the results). In some systems, terms are automatically added to the user's query before it is returned, depending on the size of the result set (e.g. [49]). The related terms may come from a thesaurus, which gathers and records related terms in the collection (e.g. [10]), or from other lexical means (e.g. [1]). Hierarchical

and dynamic clustering can also be used to help the user form their next query. For example, the Scatter/Gather system regroups the results every time the user selects a set of clusters to examine further; so at any stage of the search, this always provides the user with a set of possibilities that they know will produce results.

A few systems also provide results previews—feedback about the results of a query before the query is issued (e.g. [14,20]). These techniques allow users to manipulate query parameters and see qualities of the results set before the query is issued (for example, the number of documents in the set or their distribution across one or more kinds of meta-data). Results previews help users understand which queries are likely to succeed, and enable powerful interaction techniques like dynamic querying [40], but require that a small and fast version of the entire index be available or downloadable to the local query client [20].

Supporting browsing tasks using phrases

Our approach to supporting these browsing tasks is to use key phrases as the basic unit for indexing, retrieval, and presentation. Keyphrases provide us with a simple mechanism for supporting collection evaluation, subject-area exploration, and query exploration. The advantages of this approach are that keyphrases can be automatically extracted from documents; that the resulting indexes are simple, small, and quickly built; and that the workings of the system are easy for users to understand. In the next section, we consider these issues in more detail. We describe the Keyphind system, the technique used to extract keyphrases from documents, and how the keyphrase indexes are built.

Keyphind

Keyphind (from "keyphrase find") is a system that permits browsing, exploring, and searching large collections of text documents (see Figure 1). Like other search engines, it consists of an index and a graphical query interface; however, both index and presentation scheme differ substantially from conventional systems. First, the index is built from key phrases that have been automatically extracted from the documents, rather than from the full text of the collection. Second, documents that match a query are grouped by keyphrase in the interface, and users can work with these clusters before looking at individual documents.

More detail on these techniques is given below. First we take a brief look at what it is like to use the system. Then we describe how it obtains phrases, builds its indexes, and presents documents to the user. The examples and screen snapshots that we discuss use a database built from approximately 26,000 computer science technical reports, whose source text totals one gigabyte. The reports are part of the *Computer Science Technical Reports* (CSTR) collection of the New Zealand Digital Library [50] (www.nzdl.org). Although the example collection is not enormous by today's standards, the algorithms and techniques used in Keyphind will readily scale to larger collections.

Using Keyphind

Keyphind's interface is shown in Figure 1. A user initiates a query by typing a word or phrase and pressing the "Search" button, just as with other search engines. However, what is returned is not a list of documents, but rather a list of keyphrases containing the query terms. Since all phrases in the database are extracted from the source texts, every returned

phrase represents one or more documents in the collection. Searching on the word *text*, for example, returns a list of phrases including *text editor* (a keyphrase for twelve documents), *text compression* (eleven documents), and *text retrieval* (ten documents) (see Figure 1). The phrase list provides a high-level view of the topics covered by the collection, and indicates, by the number of documents, the coverage of each topic.

Following the initial query, a user may choose to refine the search using one of the phrases in the list, or examine one of the topics more closely. Since they are derived from the collection itself, any further search with these phrases is guaranteed to produce results—and furthermore, the user knows exactly how many documents to expect. To examine the documents associated with a phrase, the user selects one from the list, and previews of the documents are displayed in the lower panel of the interface. Selecting any document in the preview list shows the document's full text. More detail on Keyphind's capabilities is given in later sections.

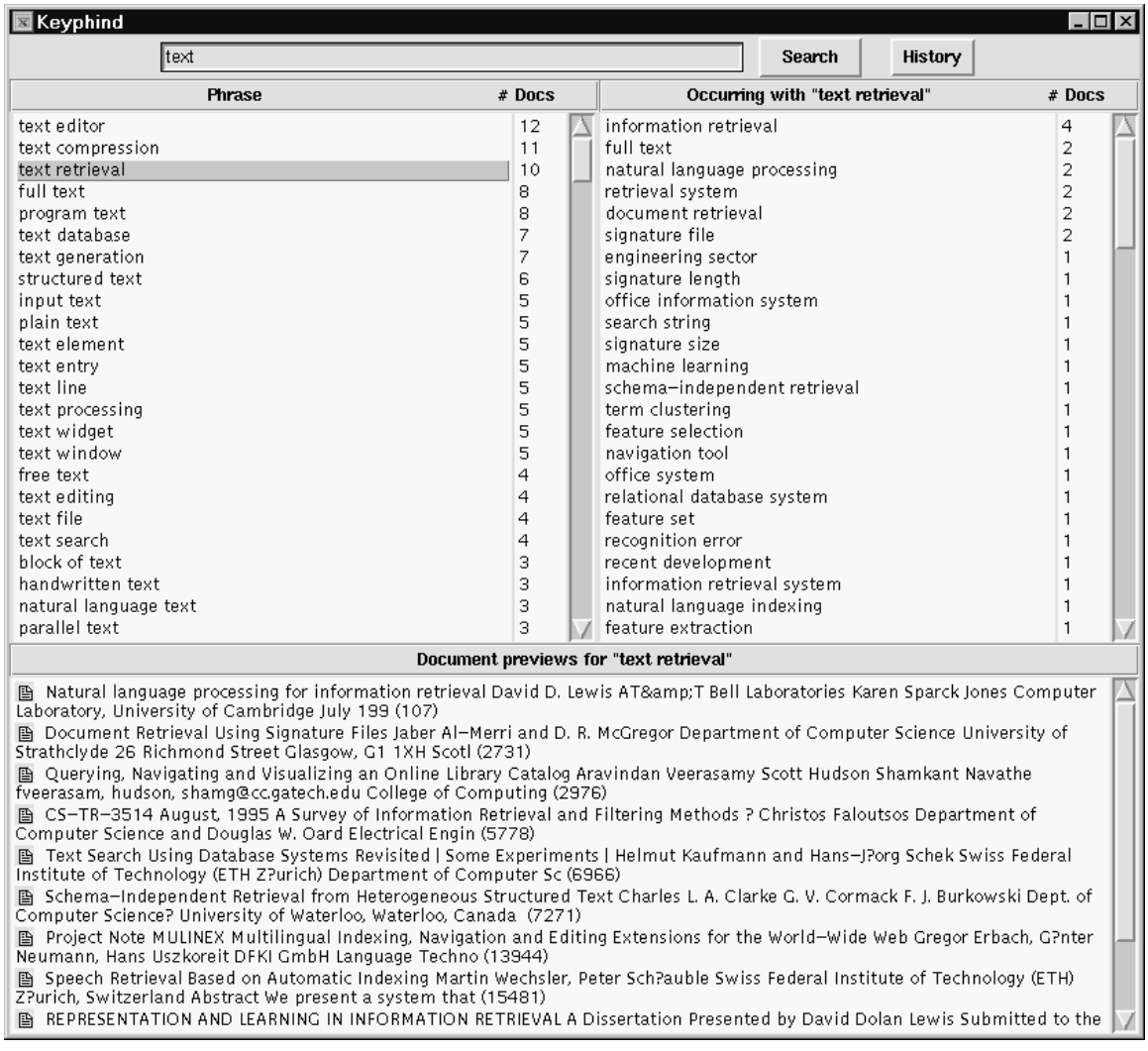

Figure 1. Keyphind user interface.

Automatic keyphrase extraction

Keyphrases are words or short phrases, commonly used by authors of academic papers, that characterize and summarize the topics covered by a document. Keyphrases are most often noun phrases of between two and four words. For example, the key phrases we have chosen for this article are *digital libraries*; *browsing interfaces*; *text mining*, *keyphrase extraction*, and *machine learning*. The index terms collected in "back-of-the-book" subject indexes also represent a kind of keyphrase. Keyphrases can be a suitable basis for a searchable index, since they represent the contents and subject matter of a document (e.g. [43]). In most document collections, however, author-specified keyphrases are not available for the documents; and it is laborious to manually determine and enter terms for each document in a large collection [10]. A keyphrase index, therefore, is usually feasible only if index terms can be determined automatically.

Automatic determination of keyphrases generally works in one of two ways: meaningful phrases can either be *extracted* from the text of the document itself, or may be *assigned* from some other source (but again using the document text as input). The latter technique, keyphrase assignment, attempts to find descriptive phrases from a controlled vocabulary (e.g. MeSH terms) or from a set of terms that have been previously collected from a sample of the collection (e.g. [38]). Assignment is essentially a document classification problem, where a new document must be placed into one or more of the categories defined by the vocabulary phrases [28,35]. In general, the advantage of keyphrase assignment is that it can find keyphrases that do not appear in the text; the advantage of keyphrase extraction (the technique used in Keyphind) is that it does not require an external phrase source. We now consider keyphrase extraction in more detail.

There are two parts to the problem of extracting keyphrases: first, candidate phrases must be found in the document text, and second, the candidates must be evaluated as to whether they are likely to be good keyphrases. A variety of techniques have been proposed for each of these steps, as described below.

Finding candidate phrases

Since any document will contain many more multi-word sequences than keyphrases, the first step in extracting keyphrases is finding reasonable candidate phrases for further analysis. There are two main approaches to finding phrases in text: syntactic analysis and statistical analysis [15,42]. Syntactic analysis parses the text and reports phrases that match predetermined linguistic categories. Often, these categories define noun-phrase forms that can be determined using a part-of-speech tagger (e.g. [2,41]). More sophisticated analysis is also possible: for example, Srtzalkowski and colleagues [43] look for head-modifier pairs, where the head is a central verb or noun, and the modifier is an adjunct argument of the head.

In contrast, finding phrases through statistical analysis does not consider the words' parts of speech. Instead, this approach calculates co-occurrence statistics for sequences of words; those words that appear frequently together are likely to indicate a semantically-meaningful concept [42]. This technique generally uses a stop-word list to avoid finding phrases made up of function words (e.g. *of, the, and, then*). In its simplest form, statistical analysis simply finds word sequences that occur more frequently than some fixed threshold [42].

Evaluating candidate phrases as keyphrases

The second step in finding keyphrases involves determining whether a candidate word sequence is or is not a keyphrase. This process generally means ranking the candidates based on co-occurrence criteria and then selecting the top-ranked phrases. The techniques used are similar to those for term weighting and feature selection in vector-space models, where a few components of a multi-dimensional term vector are chosen to represent the document [18,26]. Two common criteria for ranking candidates are simple frequency (normalized by the length of the document), and TF×IDF measures, which compare the frequency of a phrase in the document with its rarity in general use. Other possible criteria include entropy measures [35], first occurrence in the document (see below), or appearance of the phrase in "important" sections of structured text (e.g. title tags in HTML documents). Once phrases are given a score based on these criteria, the best keyphrases are those candidates with the highest ranks.

However, one problem of evaluating candidates is that when several attributes contribute to a phrase's score, it is difficult to construct a simple model for combining the variables. Machine learning techniques can be used to build complex decision models that can take several attributes into account, and this is the approach used by Keyphind. Machine learning has been used for a variety of purposes in retrieval systems (e.g. [8,35]), but only one other project to our knowledge uses it for ranking keyphrases. This is the Extractor system [47], which uses a genetic algorithm to determine optimal variable weights from a set of training documents. In the current project, we use a technique that is faster and simpler than genetic algorithms, but provides equal performance. Our extraction process is described below.

Keyphrase extraction in Keyphind

We have previously developed a system called Kea [16] to extract keyphrases from text[2]. It uses machine learning techniques and a set of training documents to build a model of where keyphrases appear in documents and what their distinguishing characteristics are. In particular, we treat extraction as a problem of supervised learning from examples, where the examples are documents in which every phrase has been manually classified as a keyphrase or non-keyphrase. This manual classification of phrases is accomplished using the set of author-specified keyphrases for the document. The learning process results in a classification model, and this model is used to find keyphrases in new documents.

Like other approaches, Kea extracts keyphrases using the two steps described above: first identifying candidate phrases in a document's text, and then ranking the candidates based on whether they are likely to be keyphrases. However, we differ from previous work in that we rank candidates using a model built with the Naïve Bayes machine learning algorithm [25]. The steps involved in training and extraction are illustrated in Figure 2; below, we describe how Kea finds and ranks phrases, and then discuss the construction of the Naïve Bayes model in more detail.

---

[2] Several versions of Kea have been implemented; the version described in [16] differs from the present description in that it uses an extensive stopword list rather than a part-of-speech tagger to identify candidate phrases in a document.

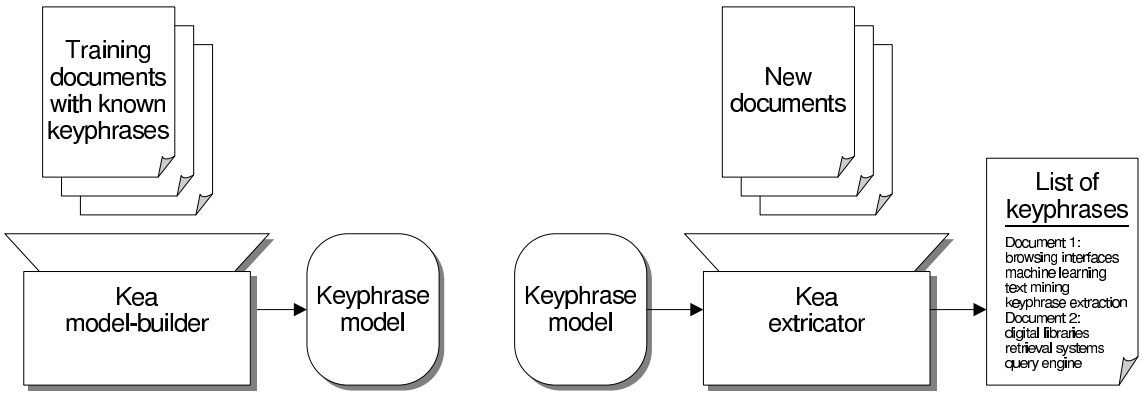

Figure 2. Kea keyphrase extraction process

Finding phrases

The process used to find candidate phrases for Keyphind is similar to that used in earlier projects (e.g. [31,41]). First, documents are cleaned and tokenised, and then tagged using the Brill part-of-speech tagger [4]. The tagger adds a part-of-speech indicator (e.g. /NN for noun, /VB for verb) to each word. Words are not currently stemmed, although we do fold plurals to singular forms (e.g. "libraries" to "library") and standardize words that have different spellings (e.g. "labor" to "labour"). Second, all phrases matching certain lexical patterns are reported. In particular, we accept any sequence of words consisting of a string of nouns and adjectives with a final noun or gerund. For example, "probabilistic machine learning" contains an adjective followed by a noun followed by a gerund. This pattern was proposed initially by Turney [47] and is similar to others used in the literature (e.g. [31]); in our own experiments with author-generated keyphrases, the pattern covered more than 90% of a test set of more than 1,800 examples.

Building the Naïve Bayes ranking model

The machine learning technique that we use to build the ranking model requires a set of training documents where positive instances (that is, "true" keyphrases) have already been identified. For our training data, we used papers from the CSTR where the authors have already specified a set of key words and phrases. Although the author's choices are not always the best examples of good keyphrases (for example, there are often good keyphrases in the text that were not chosen by the author), this method is simple and readily available.

The process of building the ranking model involves four preliminary steps. First, author keyphrases are removed from the training documents and stored in separate files. Second, candidate phrases are found in the training documents as described above (note that some author keyphrases do not appear in the document text). Third, we calculate values for three attributes of each phrase:

- distance (*d*) is the distance from the start of the document to the phrase's first appearance (normalized by document length)
- term frequency (*tf*) is the number of times the phrase appears in the document (normalized by document length)
- inverse document frequency (*idf*) is the phrase's frequency of use in the domain of the collection. This attribute is approximated using a random sample of 250 documents from the collection

Term frequency and inverse document frequency are combined in a standard TF×IDF calculation [51]. These attributes were experimentally determined to be the most useful in characterising keyphrases [16]: that is, keyphrases are more likely to appear early in a document, to appear often in the document, and to appear less often in general use. Fourth, each candidate phrase is marked as a keyphrase or a non-keyphrase, based on the author-specified keyphrases for that document.

Once these preliminary steps are completed, the Naïve Bayes machine-learning algorithm compares attribute values for the known keyphrases with those for all the other candidates, and builds a model for determining whether or not a candidate is a keyphrase. The model predicts whether a candidate is or is not a keyphrase, using the values of the other features. The Naïve Bayes scheme learns two sets of numeric weights from the attribute values, one set applying to positive ("is a keyphrase") examples and the other to negative ("is not a keyphrase") instances.

Ranking candidate phrases using the Naïve Bayes model

To rank each candidate phrase that has been found in a new document (one where there are no author keyphrases), Kea first calculates values for the three attributes described above. The Naïve Bayes model built in the training phase is then used to determine a probability score, as follows. When the model is used on a candidate phrase with feature values t (TF×IDF) and d (distance of first occurrence), two quantities are computed:

$$P[t,d,yes] = P_{TF*IDF}[t|yes] * P_{distance}[d|yes] * P[yes] \qquad (1)$$

and a similar expression for P[t,d,no], where $P_{TF*IDF}[t|yes]$ is the proportion of phrases with value t among all keyphrases, $P_{distance}[d|yes]$ is the proportion of phrases with value d among all keyphrases, and P[yes] is the proportion of keyphrases among all phrases (i.e. the prior probability that a candidate will be a keyphrase). The overall probability that the candidate phrase is a keyphrase can then be calculated:

$$p = P[yes] / (P[yes]+P[no]) \qquad (2)$$

Candidate phrases are ranked according to this value. When all of the document's phrases are ranked, the top twelve candidates are selected and written to a file for later indexing.

Further details of the extraction process and the Naïve Bayes scheme can be found in other articles (e.g. [16]).

Performance

We have evaluated the Kea algorithm in terms of the number of author-specified keyphrases that it correctly extracts from text. In our experiments[3], Kea's recall and precision were both approximately 0.23—therefore, it finds about one in four author-specified keyphrases. We also examined the question of how many keyphrases to extract, and considered both a rank cutoff and a probability cutoff. The experiments showed that selecting about twelve candidates usually includes all of the high-probability phrases (where "high" is empirically determined), without including too many poor phrases.

---

[3] In these experiments, we used a combined measure of recall and precision called the F-measure.

For this research, we have examined Kea's output for about twenty CSTR documents. We estimate that an average of nine of the twelve keyphrases extracted for each document could be considered useful to an end-user (some of these were ungrammatical, but still understandable), and about three of twelve were unusable. An example of the choices, using this article as input, is shown in Table 2.

Phrases chosen by the authors	Phrases chosen by Kea
browsing interfaces digital libraries text mining keyphrase extraction machine learning	1. digital libraries 2. browsing in digital libraries 3. keyphrase index 4. libraries with keyphrase 5. digital libraries with keyphrase 6. browsing tasks 7. keyphrase extraction 8. usability study 9. browsing interfaces 10. search engines 11. document collections 12. query engines

Table 2. Example output from Kea

The performance of our algorithm represents the current state of the art for machine-learning approaches (see [16]); however, recall and precision are still quite low. There are several reasons for this. First, only about 80% of the author's keyphrases actually appear in the text of the document, and this forms an upper bound on the number of phrases that can be extracted by any algorithm. Second, attempting to match author phrases complicates the machine-learning problem because the data is hugely skewed towards negative instances. For every author-specified key phrase in a document of the CSTR, there are about one thousand phrases that are not key phrases. Third, extracting author keyphrases is a more demanding task than simply extracting "good" keyphrases (although it is much easier to evaluate); even two human indexers will rarely come up with exactly the same index terms for a document [17].

The primary problem of evaluating our technique using author-specified keyphrases is that the scheme does not assess the quality of those keyphrases that do *not* match author phrases. We make the assumption that maximizing performance on extraction of author keyphrases will also maximize the quality of all extracted phrases. However, we are currently investigating other assessment techniques: in particular, we are constructing a corpus of documents where human indexers have marked (and ranked) *all* phrases in the text that could be used as valid keyphrases.

Kea processed the collection of 26,432 documents in about four days (on a Pentium 233-MHz system running Linux); more than half of this time was used by the part-of-speech tagger. The resulting raw keyphrase file contains 309,908 keyphrases, about 6 Mb of text. More than half of the raw phrases are duplicates from some other document; as described below, the list of unique phrases contains only 145,788 entries.

Indexing

Once keyphrases are extracted from each document in the collection, Keyphind converts the raw keyphrase file into indexes that are used by the query interface. This process, illustrated in Figure 3, produces four indexes: a phrase list, a word-to-phrase index, a phrase-to-document index, and a document-to-phrase index. The phrase list simply associates each unique phrase with a number that is used in the remaining indexes (for efficiency reasons); the other indexes are described below.

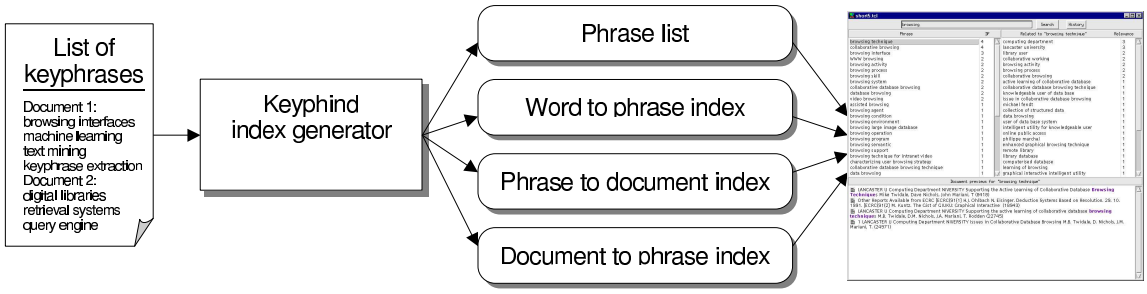

Figure 3. Generation of Keyphind indexes.

The *word-to-phrase index* lists all words in the document collection and indicates for each one the phrases that contain it. For example, "browsing" appears in "browsing technique," "collaborative browsing," "browsing interface," etc. This is the only index needed to process basic queries: the sets of phrases for each word in the query are found and then intersected to find the phrases containing *all* of the query terms.

The *phrase-to-document index* lists every phrase in the database and indicates which documents they were extracted from. Documents are represented by numbers, so the entry for "browsing technique" might show document numbers 113, 4024, 75376, and 165234. This index is used to retrieve the appropriate documents when a phrase is selected, and to calculate the co-occurring phrases for any selected phrase (see below).

The *document-to-phrase index* lists every document by number and indicates all phrases that were extracted from that document. There are twelve phrases for each document. The document-to-phrase index is used only in the calculation of co-occurring phrases (see below).

Keyphind's indexes are small, and quick to create. It took less than five minutes to build the indexes for the 26,000 document example collection (on a Pentium Pro 200-Mhz system running Linux); this speed, however, is dependent on being able to hold the entire phrase list in a hash table. More important than creation time, however, is index size, and the indexes are tiny in comparison to the source text. The file sizes for the four indexes created for the example collection are shown in Table 3. Since these indexes are derived from one gigabyte of source text, the keyphrase approach offers an index that is less than one percent of the size of the source. In comparison, standard full-text inverted index are typically larger than the source text, and even modern compressed indexes (e.g. [51]) are about one tenth the size of the source. Of course, the small size of the indexes is only achieved because we discard all but twelve phrases for each document—but we believe that for some browsing tasks, indexing twelve meaningful phrases may be as good as indexing every word.

Keyphind stores all of these indexes in memory. The word-to-phrase index is stored as a hash table for quick lookup, and the other three are stored as sequential arrays. The memory space

needed is about 20 Mb. The small size of these indexes, both in memory and on disk, means that keyphrase indexes can be downloaded to a networked client at the start of a browsing session, providing quick access and short response times.

Index	Number of items	File size
Phrase list	145,788 unique phrases	2.98 Mb
Word-to-phrase index	35,705 unique words	2.80 Mb
Phrase-to-document index	145,788 entries	1.84 Mb
Document-to-phrase index	26,432 documents	1.79 Mb
Total		9.41 Mb

Table 3. File sizes of Keyphind indexes.

Presentation and user interaction

Keyphind's query interface is shown in Figures 1 and 4. There are three activities that users can undertake. They can issue keyphrase queries, they can view and preview documents associated with a keyphrase, and they can work with co-occurring phrases related to their current selection. We describe each of these below.

Keyphrase queries

Being based on keyphrases rather than documents, Keyphind answers a slightly different question than does a full-text search engine. Instead of deciding which *documents* contain the user's query terms, Keyphind determines which *keyphrases* contain the query terms. When the user enters a word or topic and presses the "Search" button, the system consults the word-to-phrase index, retrieves those phrases that contain all of the query words, and displays them in the upper-left panel of the interface as the result of the query. In addition, the documents associated with each phrase in the result set are counted (using the phrase-to-document index) and the number is shown alongside the phrase. The result set can be sorted either by phrase or by number of documents. An example of a basic query on *extraction* is illustrated in Figure 4a.

Refining the query

The result set contains all keyphrases in the database that contain the query terms. Therefore, this list shows all possible ways that the user's query can be extended with additional words. Additional queries can be issued by double-clicking on phrases in the list.

Previewing and viewing documents

Once a keyphrase query has been made, the user can look at the documents associated with any of the phrases displayed. When the user selects one, Keyphind retrieves the first few lines of each associated document and displays these previews at the bottom of the interface (Figure 4b). If the phrase of interest occurs in the document preview, it is highlighted in colored text. To look more closely at a document, clicking on the preview presents the full document text in a secondary window (Figure 4c).

Figure 4. Four types of interaction with Keyphind. (a) Basic query results. (b) Document previews and co-occurring phrases. (c) Document inspection window. (d) Co-occurrence filtering of previews.

Related phrases and co-occurrence filtering

An additional capability for exploring the collection is based on the co-occurrence of keyphrases. When the user selects a phrase from the result list, additional phrases related to the user's selection are displayed in the upper-right panel of the interface. These related phrases can provide the user with new query ideas and new directions for exploring a topic. The scheme works as follows.

1. The user selects a phrase from the result list, such as *feature extraction* (Figure 4b).
2. *Feature extraction* is a keyphrase for 21 documents; however, since we originally extracted twelve keyphrases from each text, each of the 21 documents is also associated with eleven other keyphrases. These are called "co-occurring phrases."

3. Sometimes, a phrase will co-occur with *feature extraction* in more than one of the 21 documents, and Keyphind keeps track of this number.
4. The system puts the co-occurring phrases into a list and displays them in the top right panel of the interface (Figure 4b), along with the number of co-occurrences. For example, *pattern recognition* co-occurs with *feature extraction* in three documents, and *image acquisition* co-occurs twice.

The co-occurrence number is exactly the number of documents that would be returned if the user formed a query from both the target keyphrase AND the co-occurring phrase. Thus, related phrases can be used to filter the documents in the previews list. When the user selects a phrase in the co-occurrence list, the lower panel of the interface shows only those documents containing both keyphrases. This situation is illustrated in Figure 4d, where the user has selected *feature extraction* and then clicked on *pattern recognition* in the co-occurrence list; the documents shown in the preview panel are the two that contain both *feature extraction* and *pattern recognition* as keyphrases.

Summary

By using keyphrases as the basic unit of indexing and presentation, Keyphind allows users to interact with a collection at a higher level than individual documents. This approach supports browsing tasks in four ways.

1. *Topical orientation*. Queries return a list of keyphrases rather than documents, giving users an immediate idea of the range of topics that the collection covers. This is useful in both evaluating the collection and in exploring a subject area.
2. *Phrase-based clustering*. Each keyphrase represents several documents, so users can see and work with a much larger portion of the collection than they could with document-based searching. In addition, the reason why documents have been clustered together is easily understood—they share a keyphrase.
3. *Query refinement*. Since the returned phrases are almost always longer than the search string, the phrase list shows all possible ways that the query can be extended and still return a result. In addition, the co-occurring phrases provide a means for finding terms that are related to the user's query, and can be used to filter the query results, explore new directions, and help address the vocabulary problem (e.g. [7]).
4. *Results prediction*. By showing how many documents there are for each phrase, Keyphind indicates the collection's depth and also what will be returned when the query is extended.

Evaluation: a usability study of Keyphind

We hypothesized that using key phrases as the basis for indexing and clustering documents would support browsing in large collections better than conventional full-text search does. To test that hypothesis, and to determine the strengths and weaknesses of our approach, we conducted a small usability study in which people carried out browsing tasks with Keyphind and also with a traditional query interface. The goals of this study were to better understand how people use phrases in searching, to determine differences in people's search strategies when they used the two interfaces, and to make an initial assessment about whether working

with keyphrases makes browsing easier. Tasks involved either evaluating the coverage of a collection in an area, or looking for documents relevant to a particular topic. From our observations and from people's comparisons, it was clear that search strategies differed considerably, and that most participants found the tasks to be easier with the Keyphind interface. The next sections outline our methods and summarize our results.

Method

Ten computer science students and researchers from the University of Waikato participated in the study as unpaid volunteers. Five of the participants used search engines a moderate amount (1 – 10 sessions per week) and five searched more frequently (more than 15 sessions per week). Participants were given a short training session on the two interfaces, where the capabilities of each system was explained and where the participant carried out a practice task. Once the training session was complete, the participant began their tasks.

Tasks

Participants completed three tasks using two query interfaces. Two tasks involved assessing the coverage of a collection in a particular area, and one task involved exploring the collection to find documents relevant to a topic. A participant carried out each task first with one interface (Keyphind or traditional) and then repeated the task with the other interface. Order was counterbalanced, so that half the participants started with each interface.

Coverage evaluation tasks 1 and 2. Participants were given a task scenario that was intentionally underspecified, asking them to find subtopics of a topic area. For the first task, it was:

> "You are a researcher who has been asked to find on-line resources for doing computer science research in planning. As one part of your evaluation, please determine three kinds of planning that are adequately represented in the current collection. Adequate representation means that there should be at least three documents that have some relation to the type of planning you are investigating."

For the second task, the topic area was computer graphics, and the instructions were otherwise the same. None of the participants considered themselves to be experts in planning research or computer graphics, so they did not begin either task with a clear idea of what they were seeking. This task was designed to determine how easily people could reach a high-level assessment of a collection.

Exploration task. Participants were asked to name a subject area from their current research or studies. They were then asked to find two articles in the collection that were relevant to that topic. This task was designed to find out how people explored an area, and to determine how easily they could find something of use to their work.

Data collection

One experimenter recorded observations about strategy use during the tasks. To assist observation, participants were asked to "think aloud" as they carried out the task. After each task, participants were asked to compare their experiences with the two interfaces (see questions 1-3 of Table 4). We did not ask these questions after the exploration task, since that task was recognized to be highly variable. At the end of the session, the experimenter conducted a short interview. Four specific questions were asked (see questions 4-7 of Table 4), and then the experimenter asked additional questions to explore particular incidents observed during the tasks.

Questions asked after tasks:
1. Was it easier to carry out the task with one or the other of these systems?
2. If yes, which one?
3. If yes, was the task: slightly easier, somewhat easier, or much easier?
Interview questions:
4. What process did you use to find out about a topic with each system?
5. What were the major difference between the two systems?
6. Would you use a system like this one (Keyphind) in your work?
7. How would you use it?

Table 4. Questions asked after tasks and during interview.

Systems

Keyphind is illustrated above in Figures 1 and 4; the traditional interface that we compared it to is shown in Figure 5. This system is the standard WWW interface used with the New Zealand Digital Library (www.nzdl.org). Participants were allowed to change the search parameters in this interface (turning word stemming on or off, folding case, using boolean or ranked retrieval, phrase searching) as they wished. Search parameters in Keyphind, however, were fixed as described above.

Results

All but two participants were able to complete the tasks in both interfaces. In these two cases, technical problems prevented them from using the traditional interface. Our results are organized around two issues: first, which interface (if any) made the task easier; and second, how search strategies varied with the two interfaces.

Which interface made the tasks easier

We asked whether the first two tasks were easier with either interface, and if so, how much easier. We did not do a formal comparison for the third task, due to its open-ended nature. Participants' responses are shown in Table 5. Two participants were unable to compare the interfaces because of technical problems with the traditional interface.

Seven of eight people found the first task to be easier using the Keyphind interface, and three found the second task easier as well. No one found the tasks easier with the full-text interface, although several participants felt that there was no difference. When we considered the responses in terms of a participant's stated level of search-engine use, we did not find any clear distinctions between the moderate users (less than ten searches per week) and the more frequent users (more than 15 searches per week).

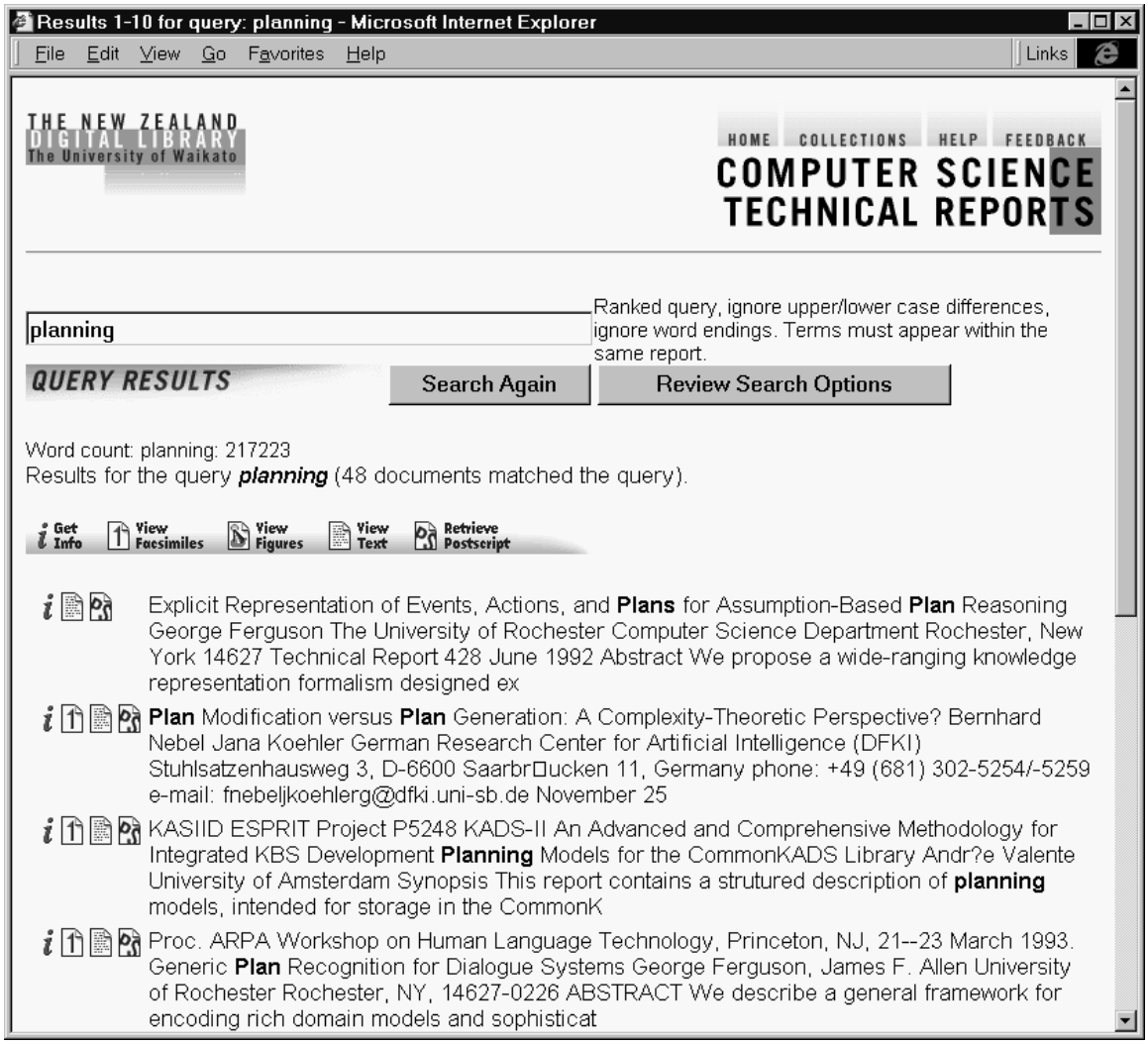

Figure 5. Traditional WWW query interface

Task	The task was easier with:			
	Keyphind	Traditional	No difference	No answer
1 (planning)	7	0	1	0
2 (computer graphics)	3	0	4	1

Task	How much easier:			
	Slightly	Moderately	Much	No answer
1 (planning)	2	0	4	1
2 (computer graphics)	1	0	2	0

Table 5. Responses to comparison questions (cells indicate number of people).

How the interfaces affected strategy use

Several differences were observed in people's strategies, differences that appear to relate to the way that the two interfaces present information. Most people began the tasks in the same

way—by entering a general query and then looking through the results—but from there the two groups diverged. In the full-text interface, people looked for information by scanning the document previews returned from the general query; in Keyphind's interface, they looked through the list of returned phrases. For example, the first task asked people to find types of planning covered by the collection. Several participants reported that in the full-text interface, they looked for phrases containing "planning" and kept track in their heads of how often they saw each phrase. Although "planning" was emphasized in the previews, this strategy could be laborious, since the previews were not formatted and showed only the first 200 characters of the file. In contrast, all the phrases in Keyphind's result list contained *planning*, and so people moved immediately to investigating those phrases that looked reasonable and were associated with several documents. The initial results returned for both interfaces are shown in Figure 6.

In the second collection-evaluation task, with the topic area *computer graphics*, strategies with the full-text interface were much the same as in the first. In Keyphind, however, the results of the initial query did not contain many phrases that were clearly subtopics of computer graphics, and the strategy of simply looking through the phrase list did not work as well. In addition, most of the documents were concentrated in one phrase—*computer graphics* itself (see Figure 7). Some participants dealt with this situation by showing the previews for this phrase, and then scanning the previews much as they did in the full-text interface. Four people, however, used the co-occurring phrases panel to look for possible subtopics of computer graphics; and there were several reasonable phrases, such as *ray tracing*, *volume rendering*, and *virtual reality* (see Figure 8).

The final task asked people to find two technical reports relevant to their area of research or study, and there were fewer strategy differences in this task. Participants began in the same way that they began the other tasks—by entering the research area as a query. In the Keyphind interface, people spent more time looking through document previews than in the other tasks, probably because the final task involved looking for documents rather than topics. Most participants used the following strategy: enter the research area as the initial query, choose a likely-looking phrase from the results list, and then scan through the document previews for that phrase.

With the full-text interface people once again carried out the task by looking through the document previews returned from the query. In this task, however, people were already familiar with the area, and so they already knew what subtopics they were looking for. That meant that when they did not find what they wanted, they (in most cases) knew of other search terms to try. Although people were on the whole successful in this task, there was some difficulty finding query terms that were appropriate to the ranked retrieval algorithm. For example, one participant entered *computer* and *vision* as query terms and was given a large set of documents where *computer* figured prominently, but that had nothing to do with computer vision. In most cases, changing the system's settings to do a string search for the phrase led to more relevant results.

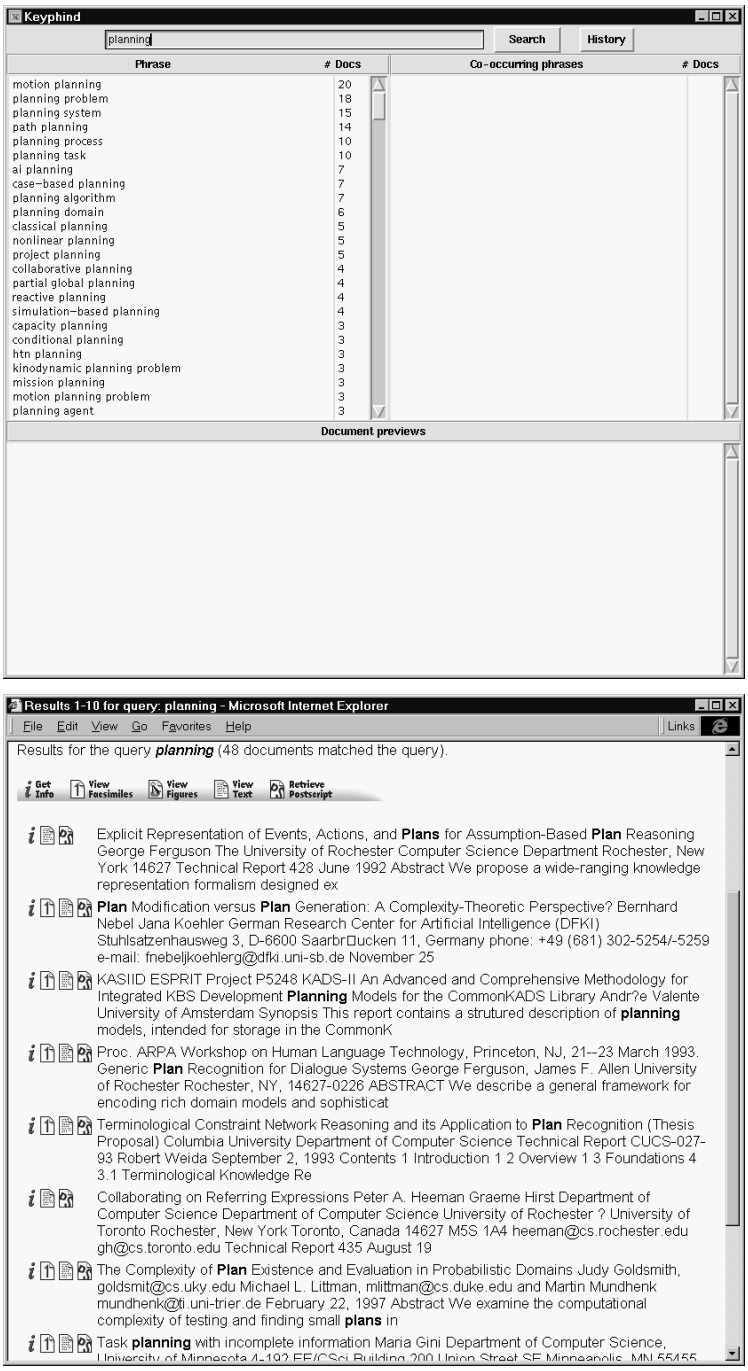

Figure 6. Results from the query *planning* in Keyphind (above) and traditional interface

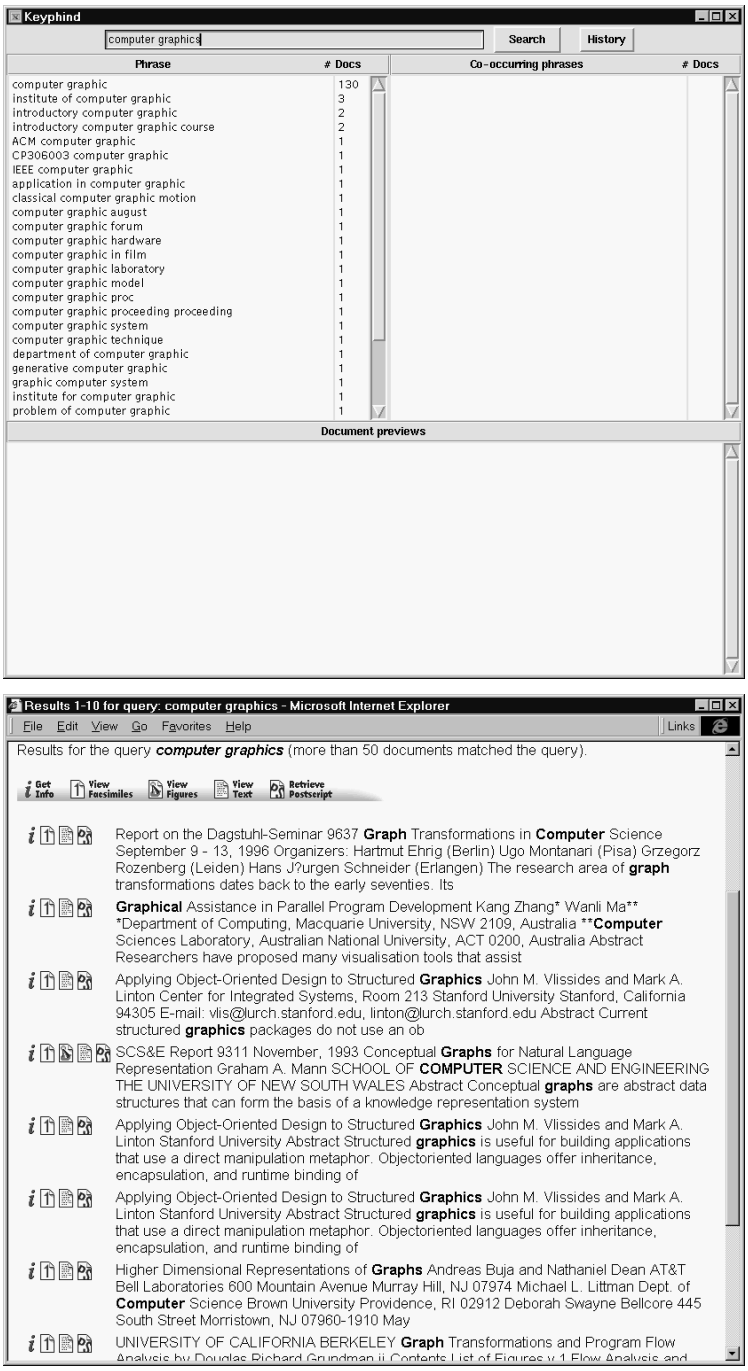

Figure 7. Results from the query *computer graphics* in Keyphind (above) and traditional interface (below)

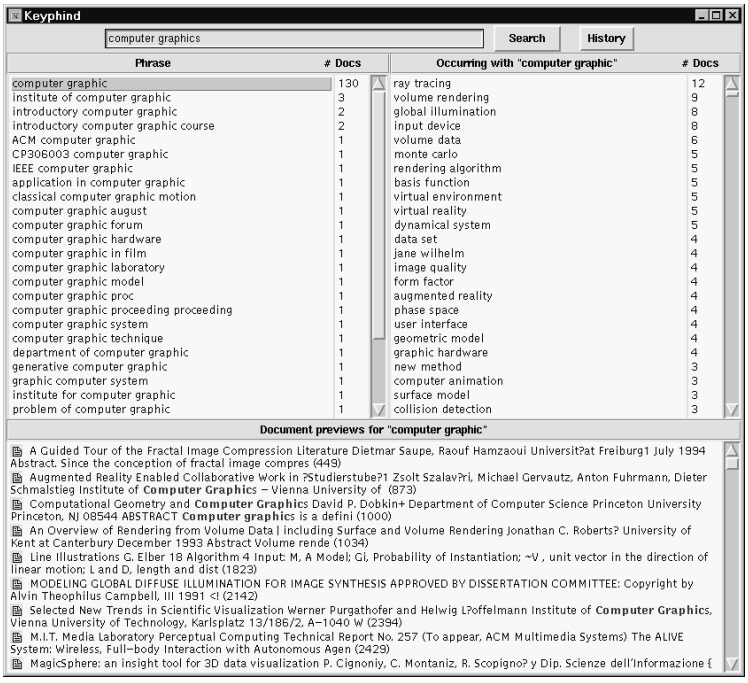

Figure 8. Most common phrases co-occurring with *computer graphics* in Keyphind (see upper right panel)

Other interview results

At the end of the session, we asked participants about the differences they saw between the two systems, and about how they might use a phrase-based system like Keyphind in the queries they performed for their research. The major differences seen by almost all of the participants were that Keyphind presented topics rather than documents, and that it was more difficult to explore an area by scanning through documents than by looking through a list of topics. However, three participants said that while Keyphind provided a faster and easier route to the subtopics of an area, they were less confident that the list of subtopics was complete than they were when they (laboriously) found subtopics by scanning document previews in the traditional interface. In addition, three participants said that they particularly liked being able to see how many documents would be returned from a further search.

When asked about how a phrase-based system could be used in their day-to-day searches, seven of the participants said that they would like to have a system like Keyphind available in addition to their normal search tools (although not as a replacement). Participants said that they would use such a system in situations where they were having difficulty with their traditional tools, or to "familiarize themselves with [an] area" when they first started to explore.

Discussion

The usability study provides evidence that a phrase-based query system can provide better support than traditional interfaces for browsing tasks, using an index that is far smaller than a full-text index. However, our experiences with Keyphind have suggested several issues in the keyphrase approach that must be considered in more detail. In this section, we look at two

main areas: first, limitations caused by the structure and format of a keyphrase approach, and second, issues involved with phrase quality and how users react to an imperfect index.

User queries and automatically-extracted keyphrases

Information retrieval literature suggests that using phrases assists some tasks and some queries, but does not greatly affect others (e.g. [31,33]). This seems to be the case with Keyphind as well, and the problem appears to lie in the fact that some kinds of user queries better fit the structure of the automatically-extracted phrase hierarchy. Borman [3] notes that one problem in traditional query systems is that successful searching requires that users know about the underlying IR techniques; and although we believe the keyphrase approach provides a much more transparent model, it does still require that users "think in keyphrases" to a certain extent.

For example, Keyphind was seen as more useful in the first task of the usability study; one likely reason for this is that people's initial queries in the first task returned a broader range of topics. An initial query on *planning* results in several phrases that can be considered sub-topics of planning (e.g. *motion planning*, *case-based planning*, *collaborative planning*); however, *computer graphics* does not produce such obvious good sub-topics. In short, this difference arises because many useful phrases fit the pattern *<something> planning*, but fewer fit *<something> computer graphics*. This situation is caused in part by the shallowness of the phrase hierarchy, which reduces the richness of the clusters when using a two-word phrase as a query. Note, however, that the case of *computer graphics* is exactly the kind of situation where the co-occurring phrases can provide an alternate set of clusters.

A second problem in matching user queries to the index is the vocabulary problem. Keyphind works best when initial queries are general and short, such as *planning*, the user can peruse the range of possible extensions to the general query and choose those that are most applicable. If users begin with a more specific query, however, they may choose the 'wrong' phrase and so miss related topics. For example, *path planning* returns only seven phrases, even though there are many more phrases under the closely-related *motion planning*. This problem could be addressed in part by using a thesaurus to automatically or interactively expand searches (although this would add another index) (e.g. [9]).

Finally, some of the queries that people make to systems like Keyphind are legitimate topics but are unlikely to appear as phrases. For example, there are undoubtedly reports relevant to *evaluation of browsing interfaces* in a 26,000-paper collection of computer science technical reports–but this phrase is unlikely to appear in the index because it is unlikely to be extracted as a keyphrase. Keyphind will therefore not return anything for this query. There are several possible ways to reduce the problem, such as automatically performing multiple searches on parts of the query, but the fact remains that keyphrases only approximate a true human-built subject index. Having several search tools that can make up for each others' shortfalls may be the only way to truly solve the problem.

Phrase quality and completeness

A user of Keyphind is in constant interaction with the phrases that have been automatically extracted from documents in the collection. For a phrase-based approach to be considered useful by its users, people must believe that the system has done a reasonable job of extracting important phrases from documents without including too much garbage.

Unfortunately, no fully-automatic system can come close to the quality of a human indexer, and so users are going to encounter both missing phrases and poor phrases.

In our view, there are three issues that affect the quality of the phrase index: phrases may be linguistically unusable (e.g. garbage strings), phrases may not be good descriptors of the document they were extracted from, or the "good" keyphrases contained in a document may have been missed by the extractor. Each of these problems reduces recall in the results set; however, in our experience with Keyphind, recall has not been a problem. In general, users seem willing to overlook a certain number of non-relevant documents (perhaps they have been well-trained in this regard by traditional search engines). Second, in large collections, recall may not as much a problem as precision: there are often so many documents that enough relevant items will be returned for the user's purposes, even though many have been missed by the indexing system. This assumes that users will be willing to work around the incorrect results, as long as they can find a reasonable set of documents in a reasonable time.

However, in our experience with the NZDL we have also found that some frequent users of traditional query systems prefer to carry out browsing tasks by looking over all of the documents and making up their own minds about the structure of the domain, even if this process is more laborious than having an automatically-extracted set of phrases. Since Keyphind cannot always provide a perfect set of keyphrases, it is essential to give people easy access to the documents themselves, so that they are not forced to trust the system but can draw their own conclusions from the raw data.

Comparison with previous work

Phrase-based systems

Phrases have been used in previous information-retrieval systems to improve indexing (e.g. [15,44]), to help refine queries (e.g. [1,38]), and to provide facilities for subject browsing (e.g. [10,29]). In addition, there are a variety of projects that have extracted different kinds of information from text, such as document summaries (e.g. [24]). In terms of the extraction of keyphrases, Keyphind is most similar to the Extractor system [47], which is fully automatic, and also approaches keyphrase extraction as a supervised learning problem (see [16] for a comparison with our extraction methods).

Keyphind differs from previous phrase-based systems in that it treats keyphrases as the fundamental unit of indexing and presentation, rather than an add-on to assist one function of a retrieval engine. This means that we use keyphrases as the only index rather than as a supplement to another index, and that we can provide retrieval, clustering, refinement, and results prediction with a single simple mechanism.

Document clustering

As described earlier, document clustering is often used in browsing interfaces as a means of providing a high-level view of a database. Keyphind differs from other clustering techniques in two ways. First, the mechanism for grouping documents together is the shared keyphrase, which is simple for users to understand—unlike a distance metric. It is easy to see why documents in Keyphind have been clustered together—they all share a common key phrase. Second, each document in the collection is always part of twelve clusters—one for each extracted keyphrase. This affords a greater degree of overlap than is generally possible with

other similarity measures. Since documents may often be about several different themes or topics, it is possible with keyphrase clustering to form an association between parts of a document, rather than demanding that two documents be similar in their entirety in order to be clustered together.

Basing clusters on keyphrases, however, does have some drawbacks not encountered in other methods. First, since documents can fall into more than one cluster, it can be difficult to determine how many documents are actually represented by the phrases returned from a query. Second, the sizes of Keyphind's clusters are variable: a keyphrase may represent hundreds of documents, or only one. Clusters that are too large or too small are less useful to the user; in particular, having many clusters of size 1 is no better than having a list of documents. Therefore, we are considering strategies for splitting large clusters and grouping small ones. For example, we could show some of the phrases within a large cluster (either the extensions of the phrase, or its set of co-occurring phrases); small clusters could be grouped by collapsing component phrases, by folding phrase variants more extensively, or by grouping synonyms based on a thesaurus.

Projects directly related to Keyphind

Finally, a note on Keyphind's lineage. Keyphind is one of several related projects dealing with the extraction and use of information from text. Kea, discussed earlier, is an ongoing project to extract key phrases from text. Keyphind is also descended from an earlier system called Phind [32] that builds phrase hierarchies from the full text of a document. These hierarchies were not based on key phrases, but on any repeating sequence in the text. The phrase indexes used in Keyphind have also given rise to a two other systems: Phrasier [21], a text editor that automatically creates links from phrases in a user's text to documents in a collection, and Kniles (www.nzdl.org/Kea/Kniles), a system that links keyphrases in returned documents to others in the collection.

Conclusion

In this article we described a search engine that supports browsing in large document collections and digital libraries. The Keyphind engine is based on a database of key phrases that can be automatically extracted from text, and uses keyphrases both as the basic unit of indexing and as a way to organize results in the query interface. Keyphind's indexes are much smaller than standard full-text indexes, are efficient, and are relatively easy to build. Keyphind supports browsing in three main ways: by presenting keyphrases that indicate the range of topics in a collection, by clustering documents to show a larger portion of the collection at once, and by explicitly showing all of the ways a query can be extended. A usability study of Keyphind gives evidence that the phrase-based approach provides better support than full-text engines for some kinds of browsing tasks—evaluation of collections in particular.

Acknowledgments

This research was supported by the New Zealand Foundation for Research, Science, and Technology. Our thanks to the anonymous reviewers for their comments and suggestions.

Software availability

Keyphind is available from www.cs.usask.ca/faculty/gutwin/Keyphind/. Kea, the system used to extract keyphrases for Keyphind, is available from www.nzdl.org/Kea/.

References

[1] P. Anick and S. Vaithyanathan, Exploiting Clustering and Phrases for Context-Based Information Retrieval, in: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 314-323 (ACM Press, 1997).

[2] N. A. Bennett, Q. He, C. Chang, B. R. Schatz, Concept Extraction in the Interspace Prototype, Technical Report, Digital Library Initiative Project, University of Illinois at Urbana-Champaign. Currently available from http://www.canis.uiuc.edu/interspace/technical/canis-report-0001.html

[3] C. L. Borman, Why Are Online Catalogs Still Hard to Use?, Journal of the American Society for Information Science, 47, No. 7, pp. 493-503 (1996).

[4] E. Brill, Some Advances in Rule-Based Part of Speech Tagging, in: Proceedings of the Twelfth National Conference on Artificial Intelligence, (AAAI Press, 1994).

[5] S.J. Chang and R.E. Rice, Browsing: a Multidimensional Framework, in: Annual Review of Information Science and Technology (ARIST), M. E. Williams ed., vol. 28, pp. 321-276 (1993).

[6] H. Chen and A. L. Houston, Internet Browsing and Searching: User Evaluations of Category Map and Concept Space Techniques, Journal of the American Society for Information Science, 49, No. 7, pp. 582-603, (1998).

[7] H. Chen and T. D. Ng, A Concept Space Approach to Addressing the Vocabulary Problem in Scientific Information Retrieval: An Experiment on the Worm Community System, Journal of the American Society for Information Science, 48, No. 1, pp. 17-31, (1997).

[8] H. Chen, G. Shankaranarayanan, L. She, A Machine Learning Approach to Inductive Query by Examples: An Experiment Using Relevance Feedback, ID3, Genetic Algorithms, and Simulated Annealing, Journal of the American Society for Information Science, 49, No. 8, pp. 693-705, (1998).

[9] H. Chen, J. Martinez, A. Kirchhoff, T. D. Ng, B. R. Schatz, Alleviating Search Uncertainty through Concept Associations: Automatic Indexing, Co-occurrence Analysis, and Parallel Computing, Journal of the American Society for Information Science, 49, No. 3, pp. 206-216, (1998).

[10] Y. Chung, W. M. Pottenger, B. R. Schatz, Automatic Subject Indexing Using an Associative Neural Network, in: Proceedings of the 3rd ACM International Conference on Digital Libraries (DL'98), pp. 59-68 (ACM Press, 1998).

[11] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections, in: Proceedings of the

Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 318-329 (ACM Press, 1992).

[12] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, Indexing by Latent Semantic Analysis, Journal of the American Society for Information Science, 41, No. 6, pp. 391-407 (1990).

[13] W. M. Detmer and E. H. Shortliffe, Using the Internet to Improve Knowledge Diffusion in Medicine, Communications of the ACM, 40, No. 8, pp. 101-108, (1997).

[14] K. Doan, C. Plaisant, B. Shneiderman, and T. Bruns, Query Previews for Networked Information Systems: a Case Study with NASA Environmental Data, SIGMOD Record, 26, No. 1 (1997).

[15] J. Fagan, Automatic Phrase Indexing for Document Retrieval: An Examination of Syntactic and Non-Syntactic Methods, in: Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.91-101 (ACM Press, 1987).

[16] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin and C. G. Nevill-Manning, Domain-Specific Keyphrase Extraction, to appear in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (Morgan Kaufmann, 1999). Also available from: http://www.cs.waikato.ac.nz/~eibe/pubs/Z507.ps.gz

[17] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, The Vocabulary Problem in Human-System Communication. Communications of the ACM, 30, No. 11, pp. 964-971, (1987).

[18] D. Harman, Automatic Indexing, in: Challenges in Indexing Electronic Text and Images, R. Fidel, T. Hahn, E. M. Rasmussen, and P. J. Smith eds., pp. 247-264 (ASIS Press, 1994).

[19] M. A. Hearst and C. Karadi, Cat-a-Cone: An Interactive Interface for Specifying Searches and Viewing Retrieval Results Using a Large Category Hierarchy, in: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 246-255 (ACM Press, 1997).

[20] S. Jones, Graphical Query Specification and Dynamic Result Previews for a Digital Library, in: Proceedings of the ACM Conference on User Interface Software and Technology (UIST'98), pp. 143-151 (ACM Press, 1998).

[21] S. Jones, Link as you Type: Using Key Phrases for Automated Dynamic Link Generation, Department of Computer Science Working Paper 98/16, University of Waikato, New Zealand (August 1998).

[22] E. Kandogan and B. Shneiderman, Elastic Windows: A Hierarchical Multi-Window World-Wide Web Browser, in: Proceedings of the ACM Conference on User Interface Software and Technology (UIST'97), pp. 169-177 (ACM Press, 1997).

[23] S. Kirsch, The Future of Internet Search: Infoseek's Experiences Searching the Internet, SIGIR Forum, 32, No. 2 (1998).

[24] J. Kupiec, J. Pedersen, and F. Chen, A Trainable Document Summarizer, in: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.68-73 (ACM Press, 1995).

[25] P. Langley, W. Iba, and K. Thompson, An Analysis of Bayesian Classifiers, in: Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 223-228 (AAAI Press, 1992).

[26] A. Leouski and W.B. Croft, An evaluation of techniques for clustering search results, Technical Report IR-76, Center for Intelligent Information Retrieval, University of Massachusetts (1996).

[27] A. Leouski and J. Allan, Evaluating a Visual Navigation System for a Digital Library, in: Proceedings of the Second European Conference on Research and Technology for Digital Libraries, pp. 535-554 (1998)

[28] C. H. Leung and W. K. Kan, A Statistical Learning Approach to Automatic Indexing of Controlled Index Terms. Journal of the American Society for Information Science, 48, No. 3 (1997).

[29] X. Lin, Graphical Table of Contents, in: Proceedings of the 1st ACM International Conference on Digital Libraries (DL'96), pp. 45-53 (ACM Press, 1996).

[30] G. Marchionini, Information Seeking in Electronic Environments (Cambridge University Press, 1995).

[31] M. Mitra, C. Buckley, A. Singhal, C. Cardie, An Analysis of Statistical and Syntactic Phrases, in: Proceedings of the 5th RIAO Conference on Computer-Assisted Information Searching on the Internet, sponsored by the Centre De Hautes Etudes Internationales D'informatique Documentaire (CID) (1997).

[32] C. Nevill-Manning, I.H. Witten, and G. Paynter, Browsing in Digital Libraries: a Phrase-Based Approach, in: Proceedings of the 2nd ACM International Conference on Digital Libraries (DL'97), pp. 230-236 (ACM Press, 1997).

[33] R. Papka and J. Allan, Document Classification using Multiword Features, in: Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM'98), (ACM Press, 1998).

[34] P. Pirolli, P. Schank, M. Hearst, and C. Diehl, Scatter/Gather Browsing Communicates the Topic Structure of a Very Large Text Collection, in: Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems, v.1 pp. 213-220 (ACM Press, 1996).

[35] M. Sahami, S. Yusufali, and M. Q. W. Baldonado, SONIA: A Service for Organizing Networked Information Autonomously, in: Proceedings of the 3rd ACM International Conference on Digital Libraries (DL'98), pp. 200-209 (ACM Press, 1998).

[36] G. Salton, Automatic Text Processing (Addison-Wesley, 1989).

[37] G. Salton, J. Allan, and C. Buckley, Automatic Structuring and Retrieval of Large Text Files, Communications of the ACM, 37, No. 2, pp. 97-108 (1994).

[38] B. R. Schatz, E. H. Johnson, P. A. Cochrane, H. Chen, Interactive Term Suggestion for Users of Digital Libraries: Using Subject Thesauri and Co-Occurrence Lists for Information Retrieval Information Retrieval, in: Proceedings of the 1st ACM International Conference on Digital Libraries, pp.126-133 (ACM Press, 1996).

[39] B. R. Schatz, Information Analysis in the Net: The Interspace of the 21st Century, White Paper for "America in the Age of Information: A Forum on Federal Information and Communications R&D", National Library of Medicine, (U.S. Committee on Information and Communications, 1995).

[40] B. Shneiderman, D. Byrd, and W. B. Croft, Sorting out Searching: A User Interface Framework for Text Searches, Communications of the ACM, 41, No. 4, pp. 65-98 (1998).

[41] A. F. Smeaton and F. Kelledy, User-chosen phrases in interactive query formulation for information retrieval, in: Proceedings of The 20th BCS Colloquium on Information Retrieval (IRSG'98) (Springer-Verlag, 1998).

[42] A.F. Smeaton, Information Retrieval: Still Butting Heads with Natural Language Processing?, in: Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, M.T Pazienza ed., Springer-Verlag Lecture Notes in Computer Science #1299, pp.115-138 (1997).

[43] T. Strzalkowski, J. Perez-Carballo, and M. Marinescu, Natural Language Information Retrieval in Digital Libraries, in: Proceedings of the 1st ACM International Conference on Digital Libraries (DL'96), pp. 117-125 (ACM Press, 1996).

[44] T. Strzalkowski and F. Lin, Natural Language Information Retrieval, in NIST Special Publication 500-240: The Sixth Text REtrieval Conference (TREC-6), (United States Department of Commerce, National Institute of Standards and Technology, 1997).

[45] R. C. Swan and J. Allan, Aspect Windows, 3-D Visualizations, and Indirect Comparisons of Information Retrieval Systems, in: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM Press, 1998).

[46] A. Tombros and M. Sanderson, Advantages of Query Biased Summaries in Information Retrieval, in: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM Press, 1998).

[47] P. Turney, Learning to Extract Keyphrases from Text, National Research Council Technical Report ERB-1057 (1999).

[48] B. Vélez, R. Weiss, M. A. Sheldon, and D. K. Gifford, Fast and Effective Query Refinement, in: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM Press, 1997).

[49] R. Weiss, B. Velez, M. A. Sheldon, C. Namprempre, P. Szilagyi, A. Duda, and D. K. Gifford, HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering, in: Proceedings of the Seventh ACM Conference on Hypertext (ACM Press, 1996).

[50] I.H. Witten, C. Nevill-Manning, R. McNab, and S. Cunningham, A Public Library Based on Full-Text Retrieval, Communications of the ACM, 41, No. 4 (April 1998).

[51] I.H. Witten, A. Moffat, and T. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images (Van Nostrand Reinhold, New York, 1994).