Combining BibTeX and PageRank to Classify WWW Pages

MIKE JOY, GRAEME COLE and JONATHAN LOUGHRAN
University of Warwick
IAN BURNETT
IBM United Kingdom
and
ANDREW TWIGG
Cambridge University

"Focused" search engines are those which limit themselves to a particular subset of the web, for example, pages from a certain web site, or pages about a particular subject. We present such an engine which uses a method of automatic classification using BibTeX bibliographies, along with an adaptation of Google's PageRank algorithm which complements this technique.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Search Process; E.5 [**Files**]: Sorting/searching; F.2.2 [**Non Numerical Algorithms and Problems**]: Sorting and Searching

General Terms: web search

Additional Key Words and Phrases: web search

1. INTRODUCTION

The construction of a web search engine which can accurately and efficiently return the web pages most relevant to a user's query is a complex task, which every day gets harder as more documents are published on the web. Popular search engines such as Google and Yahoo! store information about documents from all over the web, corresponding to many different topics. *Focused* search engines, instead of indexing pages from the whole web, limit themselves to a small subset of them [Chakrabarti et al. 1999].

Google uses a system of document importance measurement known as PageRank [Brin and Page 1998]. It views the web as a directed graph of interconnected nodes, where each edge corresponds to a hyperlink. A page with many links pointing to it is inferred to be a useful page, and so is placed higher in search results. We have developed a search engine that uses an adaptation of PageRank, which

Author's address: M.S. Joy, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK.

weights a document's rank by its overall relevance to a particular topic. Haveliwala [Haveliwala 2003] describes a system of topic-sensitive PageRank; we present a similar system for our focused search engine.

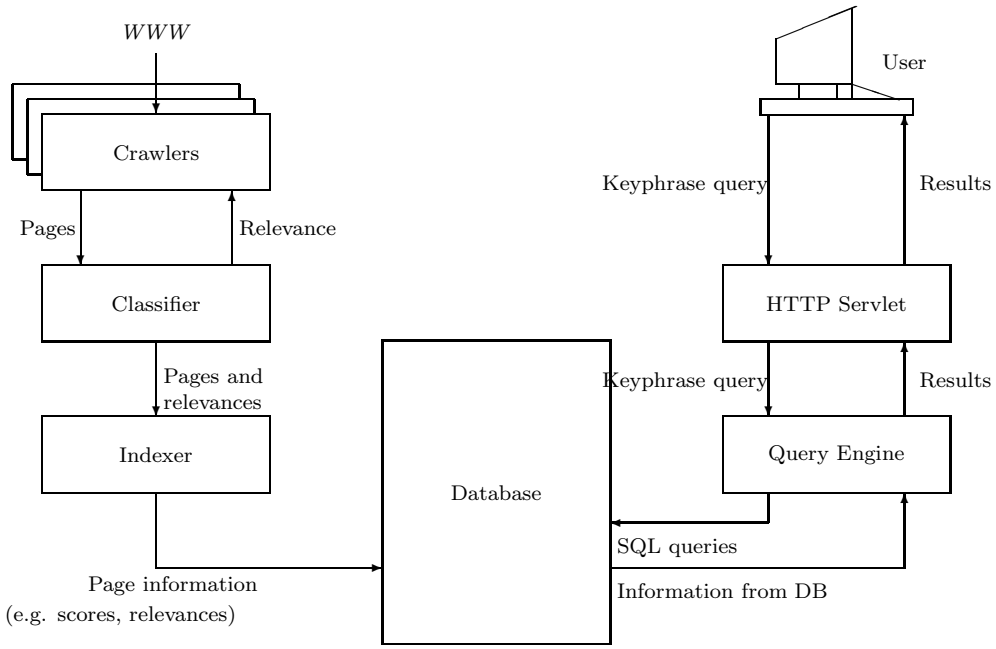

Fig. 1. Architecture of our search engine

Figure 1 shows the architecture of our search engine. *Crawlers* follow links on the web, and pass the crawled resources to the classifier, described below. If the classifier considers the document relevant to computer science, which is our search engine's area of interest, then details about that web resource are indexed into the database by the *indexer*. The crawler is a focused one, meaning it will only follow links from those pages which were determined as "relevant" by the classifier. Details stored in the database of a relevant page include an index of words in that document, together with the attributes of the occurrences of the words, for example, the position of the word in the document, and any formatting applied to the word.

The *classifier* takes documents downloaded by the crawler and uses a *classification algorithm* to determine the *overall relevance* of the document to the search engine's field of interest. It also determines the document's relevance to several subcategories of the field.

For our purposes, a *keyword phrase* is an unordered set of keywords. The job of the query engine is to take a keyword phrase and make a judgement about which subset of pages are most relevant (in some precise sense defined later) to

the keyword phrase entered. It then returns a list of relevant documents, sorted in descending order of *overall score*, the algorithms for calculating which are described later.

2. CLASSIFIER

The classification algorithm is based upon the use of bibliographies as a way of forming a set of training documents for the search engine to work with. These bibliographies are in the form of a BibTeX file, an electronic resource that stores entries about papers and reports which pertain to a particular topic. This means that all of the reports stored within the BibTeX file are all about the same topic, and so their use is perfect for this kind of situation. Figure 2 is an example of an entry taken from a BibTeX file (bibliography) containing UNIX papers.

```
@Article{Ellis:1980:LS,
  author = "J. R. Ellis",
  title = "A {LISP} shell",
  journal = "j-SIGPLAN",
  volume = "15",
  number = "5",
  pages = "24--34",
  month = "may",
  year = "1980",
  CODEN = "SINODQ",
  ISSN = "0362-1340",
  bibdate = "Sat Apr 25 11:46:37 MDT 1998",
  acknowledgement = "ack-nhfb",
  classification = "C6140D (High level languages)",
  corpsource = "Computer Sci. Dept., Yale Univ., New Haven, CT, USA",
  keywords = "INTERLISP; LISP; shell system; UNIX system",
  treatment = "P Practical",
}
```

Fig. 2. Example of an entry contained within a BibTeX file

A major advantage of using BibTeX entries as the training set is that they are generally of high quality, accurate and precise, since they are written by the authors themselves.

Using the fact that each entry stored within the BibTeX file contains a list of keywords to be associated with that publication, each document indexed can be assigned a relevance based on occurrences of those keywords in the document being classified. For example, the above entry is in a UNIX bibliography. If keywords such as “shell system” or “LISP” appear in a document fetched by the crawler, then it is reasonable for the classifier to assume that the document is relevant to the UNIX category. However, before this comparison can be performed, the BibTeX files are converted into a more useful form we call a classification graph.

2.1 Classification Graph

A *classification graph* is a layered graph built up by parsing BibTeX files. The graph is used to store the information gathered from the training documents (the

BibTeX files) in a way that is convenient for the classification algorithm to use. The classification graph is a directed acyclic graph that has four different types of nodes divided into levels, as in Figure 3.

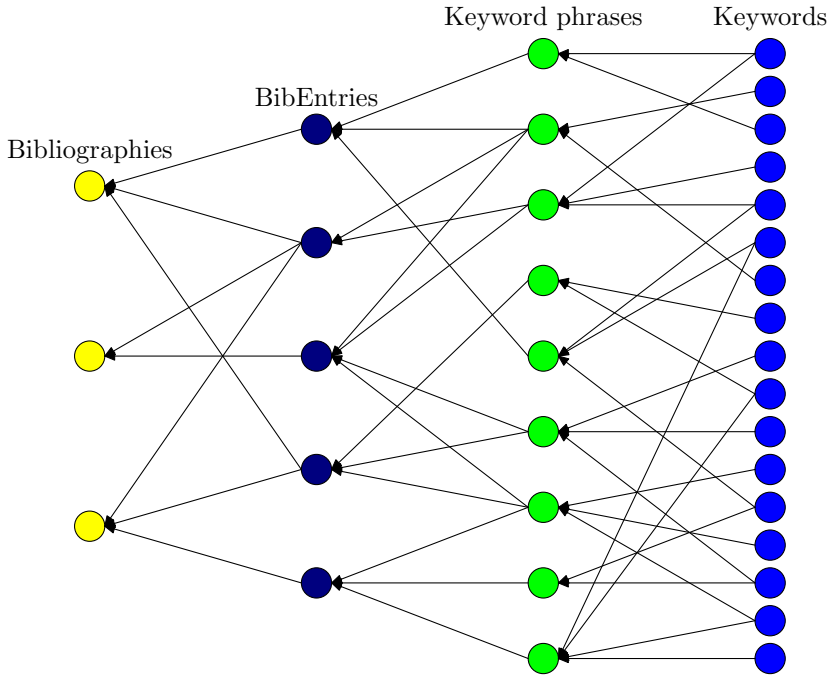

Fig. 3. Classification Graph

Level 1 (bibliography) nodes represent categories, level 2 (bibentry) nodes represent documents, level 3 nodes represent keyword phrases and level 4 nodes represent individual keywords. The edges in the graph represent the natural subset relation between each of the levels, for example, all of the keyword phrases containing a certain keyword.

The scope of a crawler is the union of all the classification graphs derived from a set of BibTeX documents, and the scope of a search can be extended by incorporating extra bibliographies. For example, given a bibliography containing papers on "optimizing Linux performance", that search scope be extended by including a classification graph of papers on "optimizing UNIX performance", and extended further by including a classification graph of papers dealing with "optimizing operating system performance".

In terms of the BibTeX files themselves, each level 1 node is a single bibliography file, each level 2 node is a publication's entry (which we will refer to as a *BibEntry*), each level 3 node is a phrase taken from the list of keywords for a document, and each level 4 node is from a phrase split up into individual words.

The motivation behind splitting all of the information up in this manner is to allow a direct comparison with all the keywords taken from the document being

classified, with the keywords taken from each the training documents, to allow a relevance for each category to be calculated.

2.2 Classification Algorithm

The aim of the classification algorithm is to calculate the relevance of a given document to the previously specified categories. These categories are created using the BibTeX training documents (see figure 2). The actual input that the algorithm takes is a list of all the words stored within the document with all common words removed, and the classification graph `bibGraph`. This process is performed to help save space when storing the results, and to cut out any words that are not specific enough to search for during any query. Each of the keywords obtained from the document are assigned a score depending on the attributes of each occurrence of the word in the document (see section 4.1.1).

The keywords are stored in a data structure `bHits`, which for each keyword from the document stores *count* (the number of times that a word appears in the document, and *score* (the score of the word for each occurence). Bucketed hits are explained in greater detail in section 4.1.2, and the classification algorithm is shown in pseudocode in appendix A. The algorithm consists of five sequenced components:

(1) Filter the keywords from the document so that only keywords that correspond to keywords from the classification graph are left. This action is performed so that only the keywords pertaining to the topic currently being used will be compared.

(2) Filter out all of the keyword phrases that do not match the keyword phrases from the classification graph using an algorithm similar to the Phrase Matching Algorithm used by the query engine, described in section 4.1.2. The score for the phrase is equal to that generated by the Phrase Matching Algorithm.

(3) Sum the scores of all the keywords contained in the document, for each BibEntry containing them. The score for each keyword is equal to the keyword score as previously defined multiplied by the degree of that node. This means that the more keyword phrases that a keyword appears in the highier the score of that keyword. The idea is that if a keyword is contained in more phrases, then it is more likely to be important to the category being focused upon.

(4) Sum the scores for each BibEntry within a category to form a score for that Bibliography.

(5) Sum the Bibliography scores to form an overall document relevance score.

3. AN EXTENSION TO PAGERANK

Google's PageRank algorithm [Brin and Page 1998] was modified to complement the classifier design detailed earlier. PageRank, rather than determining the *relevance* of a document to a particular search phrase, gives an indication of the document's *importance*, working on the principle that "a link from a page p to a page q can be viewed as an endorsement of q by p" [Dhyani et al. 2003]. To describe our algorithm, it is necessary to give a short summary of PageRank.

3.1 PageRank

The PageRank algorithm is a mathematical model of a "random surfer" who starts on one (random) page of the crawled webspace, and randomly follows links, never moving backwards. Occasionally, the random surfer stops following links and starts again on a randomly chosen page. A page's PageRank is simply a measure of the probability that this "random surfer" will, in an infinitely long walk, be at that page. The pagerank of a page p is defined as the sum of the pageranks of the pages that link to it, weighted by the outdegrees of the linking pages. Resolving this definition is done by considering the principal eigenvector of the link matrix.

To calculate the PageRanks, a directed graph is built to represent the web, in which nodes represent documents, and edges represent hyperlinks between pages. The graph has associated an $n \times n$ transition matrix $\mathbf{T}$, where n is the number of URLs crawled, and $\mathbf{T}_{ji}$ is the probability of the "random surfer" moving to page j given that it is currently on page i, readily computed by the number of links on page i[1]. Power iteration is then used to calculate an approximation to the principal eigenvector of this matrix, which corresponds to the PageRanks of the documents concerned.

3.2 How the extension differs

Our adaptation of PageRank maintains the main principle that a page's backward links (that is, the links pointing *to* the page) are most significant in determining the page's importance. However, in this extension, forward links and overall relevances (as determined by the classifier at the crawling stage) are taken into account as well.

3.2.1 *Allowing the random surfer to follow backward links.* It is not just the links *to* a page (its *backlinks*) that are considered when calculating the transition matrix, but also the links *from* the page (its *forward* links). Hence, the random surfer is given a "back button", as found on web browsers. Say a page a links to another page b, but not vice versa. Using the normal PageRank algorithm, it is b's backlinks that determine its importance, i.e. there is a probability that the random surfer may traverse from a to b. However, we model the probability that it may jump from b back to a.

Backward links are of course still considered more important than forward links (which actually has the effect that the random surfer is more likely to follow forward links than backward links, as is intuitive for a web surfer); if forward links were considered more important, then a website owner need only insert a large number of links to popular sites to artificially inflate their rank.

3.2.2 *Allowing the random surfer to remain on the same page.* We alter the graph of the web slightly by assuming that every page has a link to and from itself. This is again modifying the behaviour of the random surfer. The transition matrix $\mathbf{T}$, which defines the random surfer's behaviour, represents a Markov chain; when it is time for the random surfer to make a transition, it moves from one page to

[1]more precisely, a *damping factor* is used so that there is a small probability of the random surfer moving to a random node probably unconnected with i, so $\mathbf{T}_{ji}$ (if there is an edge from i to j) is not simply the reciprocal of the out-degree of i. For more information, see [Brin and Page 1998].

another according to the probabilities in the appropriate column of the transition matrix. However, if the page is a "good" page, a user could reasonably be expected to read it for longer before clicking a link to go somewhere else. We argue that this models more accurately the behaviour of a human surfer looking for specific content (as opposed to surfing at random).

3.2.3 *Considering overall relevance.* In the previous paragraph, how do we define a "good" page? We cannot simply calculate the page's score with respect to some arbitrary search terms, as the ranks are calculated just after the crawling stage, before any queries are submitted to the search engine. So, we must use the *overall relevance*, determined by the classifier. While in PageRank, the random surfer has an equal probability of following any link it sees, in our adaptation, these probabilities are weighted by each page's overall relevance.

3.2.4 *The Transition Matrix.* If we consider the webspace crawled as a graph G with a set of edges E, then n (the number of rows and columns in the transition matrix $\mathbf{T}$) is the number of nodes in G, denoted $|G|$. We assume that the random surfer has an interest in the subject of our focused crawler, hence, links to pages that were deemed relevant overall are more likely to be followed. Let the relevance of a page p be denoted $C(p)$; we then use the *normalised* relevance, which expresses the relevance score as a figure between zero and one:

$$C_{norm}(p) = \frac{C(p)}{C(p)+1} \tag{1}$$

This has the desirable effect that the ranks do not rely too much on the classifier's results; they rely more on the link structure of the web.

As stated earlier, our "random surfer" can follow backward links as well as forward links. It is more likely to follow forward links than backward links (thus making a page's *backward* links more important when determining that page's rank, as the random surfer is more likely to come from the edges pointing *into* that node), and this preference for following forward links, k, is set at 0.7 in our implementation. The exact value of k appears not to be critical.

Let us also introduce some auxiliary functions. F_i denotes the set of all pages which are linked to from page i (i's forward links) and B_i denotes the set of all pages which link to page i (i's backward links). That is, for a graph (webspace) G with a set of edges E,

$$F_i = \{p \in G \mid (i,p) \in E\} \qquad B_i = \{p \in G \mid (p,i) \in E\}$$

For any two pages i and j, $(i,j) \in E$ if and only if there is a link on page i which points to j, **or if** $i = j$. Recall that we consider every page to have a link to and from itself. Finally, $A(i,j)$ indicates the presence of a link between two pages, as follows.

$$A(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

PageRank uses a *damping factor*, which represents the probability of the random surfer not following a link, but starting again on another randomly chosen page. Our system also uses a damping factor, but for a different reason; it avoids infinite values

when dealing with nodes which were given zero relevance. In our implementation, the damping factor d is 0.1, the same as in PageRank, and like the value of k above, its value is not critical.

So, we can now define the transition matrix $\mathbf{T}$ as:

$$\mathbf{T}_{ji} = k\left(\frac{A(i,j).(d + C_{norm}(j))}{\sum_{\forall p \in F_i}(C_{norm}(p) + d)}\right) + (1-k)\left(\frac{A(j,i).(d + C_{norm}(j))}{\sum_{\forall p \in B_i}(C_{norm}(p) + d)}\right) \quad (2)$$

The sum of each column of $\mathbf{T}$ must be 1, that is, the random surfer must always have a defined "next move", even if that move is to follow the current page's loop-link to itself.

3.2.5 *Rank calculation.* The ranks, as with Google's PageRank, correspond to the elements of the principal eigenvector of the transition matrix $\mathbf{T}$, defined above. This can be found by calculating the steady-state probability vector $\mathbf{r}$ (which is a vector of ranks) such that $\mathbf{Tr} = \mathbf{r}$. This is accomplished by starting with an initial arbitrary n-dimensional probability vector $\mathbf{p_0}$, and repeatedly premultiplying this vector by $\mathbf{T}$. Eventually, irrespective of $\mathbf{p_0}$, the vector so obtained will tend towards $\mathbf{r}$ with successive multiplications. So, by the equation and the ergodic theorem:

$$\lim_{i->\infty} T^i p_0 = r$$

we take a suitable value for infinity (10 usually gives adequate results) and produce a sufficiently accurate approximation to $\mathbf{r}$. The sum of all the elements in $\mathbf{r}$ should be one (as it is a probability vector). However, we, as do Google, multiply each element of $\mathbf{r}$ by $|G|$, so that the *mean* of all the ranks is one. This means that the magnitudes of the ranks are independent of the number of pages crawled.

3.3 Subject-based ranking

Recall that the classifier, as well as assigning each document an overall relevance score based on the document's relevance to the classifier's field of interest, also assigns several "category relevances" based on the document's relevance to subcategories of the classifier's field of interest, defined by the individual BibTeX bibliographies that comprise the classifier's training set of documents. Using these category relevances, we can calculate not only the rank of a document, but the individual "category ranks" calculated by considering only those pages which were considered relevant to the category in question, and using the category relevance to weight the probabilities instead of the overall relevance. This gives us a topic-sensitive PageRank system, similar to that discussed in [Haveliwala 2003].

The subject-based ranking technique brings an element of relevance (to the user's search query) into the rank, which is normally a measure only of importance.

The transition matrix for subject-based ranking is calculated in exactly the same way as in equation 2, with the following exceptions:

—The graph G contains only those nodes which were given a non-zero relevance to the category in question. The set of edges E excludes any links in which either end points to a node not in G.

—Instead of the overall relevance for a document being used to weight the probabilities, C_{norm} refers to the relevance assigned to that document for the category, rather than the overall relevance. The category relevance is normalised in the same way as in equation 1.

In this way, we can produce a rank vector for each individual category, and use these ranks instead of the overall ranks should the user select a particular category in which to focus the search.

4. USING THE RANKS IN PLACING SEARCH RESULTS

When a query is submitted to the search engine, the query is processed by the query engine, which uses the information stored in the database by the crawler (for example, word frequencies for each document) to decide which documents should appear at the top of the list returned to the user. Rank alone is not an adequate means of sorting this list, as rank is a measure of importance and not of relevance; it is usually helpful if the pages returned are popular, important ones, however this is not the case if those pages bear no relevance to the topic of interest (determined by the classification graph used as the training set). On the other hand, it is not always the case that simple analysis of a page's content can determine the usefulness of the page with respect to the user's query; pages further up the hierarchy of a relevant website tend to be more useful than a page deep within the filesystem which may nonetheless contain more occurrences of the search words. Hence, a means of blending relevance and importance is required.

4.1 Word and phrase counts

4.1.1 *Keyword Scores.* The crawler side of our search engine associates each distinct word in a document with a score, based on the prominence of that word in the document. Thus, there can be considered to be a function:

$$keywordscores : document \rightarrow (keyword \rightarrow score)$$

If a word w does not appear in the document d, then its score in that document is zero, that is, $(keywordscores\ d)\ w = 0$. This score is calculated at the crawling stage by assigning "weightings" to certain HTML tags as follows:

Tag	Weight
<body>	2
<title>	5
<bold>,<b>,	3
<h1>	5
<h2>	4
<h3>	3
<h4>	2
<meta-keywords>	4
<meta-description>	3

Table I. Hit weights for HTML tags

Every word in the document then has a weighting according to the HTML tags in which it is enclosed. In the case of nested tags, the respective weightings are

summed; for example, if a word is in a `<bold>` tag which is itself contained within the `<body>` tag, then that word would be given a weighting of 5. The score of a word w in a document d is then defined as the sum of the weights of all the occurrences of w in d.

So, given a document d and a search query q, we can calculate the Keyword Score Sum (KSS) as the sum of all the keyword scores of the words in q for the document d, i.e.

$$KSS(d,q) = \sum_{i=0}^{n-1} keywordscores\ d\ q_i$$

However, we will see later that *normalising* this value produces better results. So, the normalised keyword score sum, KSS_{norm}, is defined:

$$KSS_{norm}(d,q) = \frac{1}{n} \sum_{i=0}^{n-1} \frac{keywordscores\ d\ q_i}{keywordscores\ d\ q_i + 1} \tag{3}$$

4.1.2 *Phrase Matching.* The crawler also stores information about the position of words in the document. A particular occurrence of a word in a document is called a *hit* and has associated with it the position of the word in the document and its weight. The average weight of a word is simply the weighting of that occurrence of the word divided by the total number of occurrences of that word in the document.

A collection of these hits, all of which pertain to a particular word in a document, is called a *bucket* of hits. Every page is associated with some buckets of hits, each bucket corresponding to a word. We used an algorithm that calculates both exact and approximate phrase matches within a document, as described below.

4.1.2.1 *Exact Matches.* Let $q = (q_1, \ldots, q_n)$ be a search keyword phrase with keywords $q_1, q_2, \ldots$, and the subscripts to d have an analogous meaning — d_1 means the first word of the document d. Let us also define a set $wordpos(d,q)$, which is the set of all the word positions (natural numbers) at which the phrase q appears in the document d, where word d_i is equal to q_i. That is:

$$wordpos(d,q) = \{i : \mathcal{N} \mid (\forall pos : \mathcal{N} \mid i \leq pos < n+i \bullet pos \in \mathrm{dom}(bucketedhits\ d\ q_{pos-i}))\}$$

So, $wordpos(d,q)$ is the set of all word positions i at which, for all word positions pos between i and $i+n-1$ inclusive, it is the case that pos is an element of the domain of the set $bucketedhits\ d\ q_{pos-i}$ — that is, the word q_{pos-i} exists at position pos in the document d.

The "exact phrase" component of the overall phrase matcher score is the sum of the average weights of all words in the document which are part of an exact phrase match.

$$PM_{phr}(d,q) = \sum_{\forall i \in wordpos(d,q)} \sum_{j=i}^{i+n-1} bucketedhits\ d\ q_{j-i}\ j$$

4.1.2.2 *Close Matches.* A "close match" of a phrase is defined as the occurrence of the phrase in the document with no more than one additional word inserted between each of the phrase words. We can define this mathematically with the help of

some auxiliary functions. Let $phrase(d, p)$ take a document d and a set of positions (natural numbers) p. It returns a phrase (string) s which is the concatenation of all the words in d whose positions are included in the set p, in ascending order of position number. Let $min(p)$ return the lowest numbered element in the set of naturals p. We can now define a function *closepos*, analogous to the *wordpos* function in the exact match instance, as follows.

$$closepos(d, q) = \{p : \mathcal{P}(\mathcal{N}) \mid phrase(d, p) = q \ \wedge$$
$$(\forall i \in p \ \bullet \ (\exists j \in p \mid j < i) \ \Rightarrow \ \exists k \in p \mid k < i \ \wedge \ i - k \leq 2) \ \wedge$$
$$min(p) \notin wordpos(d, q)\}$$

So, our function $closepos(d, q)$ returns the set of all sets of positions which represent in d the phrase q, for which each position, if it is not the lowest in the set, has another position number at most two words before it. We also do not want exact phrase matches in this set, so we stipulate that the lowest position number in any of the sets in $closepos(d, q)$ cannot be an element of $wordpos(d, q)$. We now express the close match component of the overall phrase matcher score as half the sum of the average weights of all those words in the document which are part of a close match. Let d_i denote the word at position i of document d.

$$PM_{clo}(d, q) = \frac{1}{2} \sum_{\forall p \in closepos(d,q)} \sum_{\forall i \in p} bucketedhits \ d \ d_i \ i$$

4.1.2.3 *Total Phrase Matcher Score.* The phrase matcher score involves the phrase match hit weights and the close match hit weights; the number of phrase and close matches, which are equal, respectively, to the number of elements in $wordpos(d, q)$ and $closepos(d, q)$, and are denoted below as M_{phr} and M_{clo}; and a pair of constants C_{phr} and C_{clo}, which in out implementation were 8 and 4 respectively. The formula for calculating the phrase matcher score is as follows.

$$PM(d, q) = (PM_{phr}(d, q) \ + \ PM_{clo}(d, q))(C_{phr}M_{phr}(d, q) \ + \ C_{clo}M_{clo}(d, q))$$

5. EXPERIMENTAL RESULTS

5.1 Method

In order to test our algorithms performed well (that is, returned relevant results) a comparison method was devised. For each search term supplied to the engine, the first ten results returned were examined, and the results' relevance to the search terms evaluated. Each result was then given a score of 1 if it was relevant, 0 for somewhat relevant and -1 for irrelevant. This is analogous to the method suggested in [Li and Shang 2000]; we decided to use -1 for an irrelevant result to penalise algorithms which gave many irrelevant results. While a simple three-point scoring system might seem vague, using a finer score range might introduce an element of subjectivity into the experimenter's judgement. The three-point scoring system provides a trade-off between providing an adequate range of results (as opposed to "relevant" or "not relevant") while limiting the degree of subjectivity that might inevitably be introduced by a human experimenter. Note that the experiments must be undertaken by a human being, because a computer program cannot decide the relevance of a search result to a query any more accurately and objectively than

a human; if it could, then we could simply use this program as our query engine. It is worth remarking that the word "relevant" is a subjective term, which depends on the experimenter, and we have not offered a specific formal definition for the term. When we evaluated the "relevance" of results to a search term, the specific question we asked was: "if I was searching for general information about the given subject, would the page have been useful?" This was a question which, for almost all cases, the experimenters were able to agree on the answer.

5.2 Implementation

For each version of the algorithm we tested, eleven computer science-related search terms were used as test data, for which to examine and score the results. The first ten results for each search term were examined, and the judged scores for each search term were summed, hence each tested algorithm was assigned a judgement between -110 and 110. Although there is no *guarantee* that all pages in a given site will have been crawled (since the crawler chooses which links to follow), it is unlikely than an important or relevant page will have been omitted. All pages in the sites used for the experimental results presented here were crawled.

The test searches were performed on a database containing data crawled from our computer science department's website — a total of 6,489 documents.

Table II shows the scores for each algorithm. The quantities on which the results were sorted in each algorithm is shown in the left-hand column, where KSS denotes Keyword Score Sum, explained in section 4.1.1, PM denotes the phrase matcher score, and R denotes rank. The subscript $_{norm}$ indicates that the quantity was normalised in the same way as in equation 1, so $R_{norm} = \frac{R}{R+1}$. For the "category" algorithm, an appropriate category was selected for each search term (if one was available) and the search limited to that category, using the appropriate category ranks instead of the overall ranks for R. The exception to this rule is KSS_{norm}, which is defined as in equation 3.

Algorithm	Score
$KSS + PM$	40
$(KSS + PM) \times R$	32
$KSS_{norm} + PM_{norm} + R_{norm}$ (overall)	57
$KSS_{norm} + PM_{norm} + R_{norm}$ (category)	62
Google	75
Yahoo!	69
Atomz	9

Table II. Relevance judgements for the tested algorithms, alongside judgements for performing the same experiment on other search engines for comparison purposes.

5.3 Comparison

The same experiments were performed on Google, Yahoo! and the search engine used by the university, provided by Atomz. The same eleven search terms were supplied to these search engines, limited only to those pages within the department's webspace. The top ten PostScript or HTML files were examined and assigned a relevance judgement as before by the experimenter; the reason for the limited range

of file types was to enable a meaningful comparison with our search engine, which currently only has plugins for processing those file types, although our software allows plugins for other file types to be added at a later date.

Our experimentation showed that our search engine gives reasonably relevant results. While the scores attained are not as high as the most popular search engines, they are comparable with them, and much higher than the commercially available search engine used by the university.

5.4 Summary

The normalisation of the individual quantities which comprise the overall score by which the query engine orders the presented results was seen to give a better performance overall than using the raw values. From examining the results returned by each algorithm for the search terms, it became apparent that the results a user would consider most useful had reasonable keyword score sum scores for each of the search words, rather than a very high score for one of the words and low scores in the others, which is why the normalised keyword score sum (which does not allow the score of any one word to dominate over the others) was used. Additionally, normalising the phrase matcher score and the ranks had the desirable effect that very highly ranked documents were not unjustly placed above lower-ranked, but more relevant, results.

6. CONCLUSION

We have presented a method of automatic classification which classifies crawled documents based on the contents of BibTeX bibliography files of known category. Using an adaptation of Google's PageRank, we have constructed a focused search engine. Our adaptation of PageRank was further extended to exploit the classification method of our engine, using subject-based ranking. Experiments undertaken on our search engine using some sample search terms on the department's webspace of 6,489 URLs showed that the system performs comparably to some of the popular search engines.

6.1 Acknowledgements

This study is based on work originally undertaken by Ian Burnett, Hristo Djidjev, Mike Joy and Andrew Twigg, who conceived the original designs for the classification and ranking algorithms presented in this paper.

A. CLASSIFICATION ALGORITHM

The following constants are used to refine the output of the classification by changing the threshold levels; if, for example, the score given to an entry is lower than $Min\_BibEntry\_Score$, then that entry is not considered further.

—$Min\_Keyword\_Score$
—$Min\_BibEntry\_Score$
—$Min\_Bibliography\_Score$
—$Min\_Document\_Score$

`Map L`

```
For each Keyword k in bHits
  Find the corresponding Bibkeyword bibKey in bibGraph
  If bibKey = null, skip to the next k
    For each neighbour of bibKey (ie all the phrases containing
bibKey)
      count = L.get(phrase)
      If count = null
        L.put(phrase,1)
      Else
        L.put(phrase,count+1)
    END
  END
END

Set all of the scores and counts for every element in bibGraph to 0.

Set K,B

For each Phrase k in L
  count = Count of k
  If count >= Number of words in k
    Use Phrase Matching Algorithm (see section)
    k.score = score from Phrase Matching
  If Number of Phrase Matches >= Min_Keyword_Score
    Add k to K
  END
END

For each Keyword k in K
  For each BibEntry b containing k
    b.score = b.score + (k.score * k.degree)
(The degree of k is the number of neighbours that the node has)
    b.count = b.count + 1
    If b.count >= Min_BibEntry_Score
      Add b to B
  END
END

docScore = 0
For each BibEntry b in B
  For each bibliography bb containing b
    bb.score = bb.score + b.score
    docScore = docScore + b.score
    bb.count = bb.count + 1
    if bb.count >= Min_Bibliography_Score
```

```
      Add bb to results
  END
END

Boolean isRelevant = (docScore >= Min_Document_Score)
```

The document's scores for each of the categories are stored inside `bb.score` for each of the categories, and the overall document score which is a sum of all of the bibliography scores is stored in `docScore`. These values are important when using the Page Ranking Algorithm and describe the relevance that a document has to each of the categories.

REFERENCES

Achilles, A.-C. The Collection of Computer Science Bibliographies. http://liinwww.ira.uka.de/bibliography/.

Brin, S. and Page, L. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World-Wide Web Conference*.

Chakrabarti, S., van den Berg, M., and Dom, B. 1999. Focused crawling: A new approach to topic-specific web resource discovery. In *WWW8*.

Dhyani, D., Ng, W. K., and Bhowmick, S. S. 2003. A survey of Web metrics. *ACM Computing Surveys 34,* 4 (Dec.), 469–503.

Haveliwala, T. H. 2003. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *IEEETKDE: IEEE Transactions on Knowledge and Data Engineering 15*.

Kumar, S. R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tomkins, A., and Upfal, E. 2000. The web as a graph. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-00)*. ACM Press, N. Y., 1–10.

Li, L. and Shang, Y. 2000. A new method for automatic performance comparison of search engines. *Kluwer's World Wide Web Journal 3,* 4, 241–247.

McCallum, A. and Nigam, K. 1998. A comparison of event models for naive Bayes text classification. In *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*. 41–48.

Yang, Y. 1999. An evaluation of statistical approaches to text categorization. *Information Retrieval 1,* 1–2, 69–90.