Analysis of a Small Search Engine

Wes Dyer

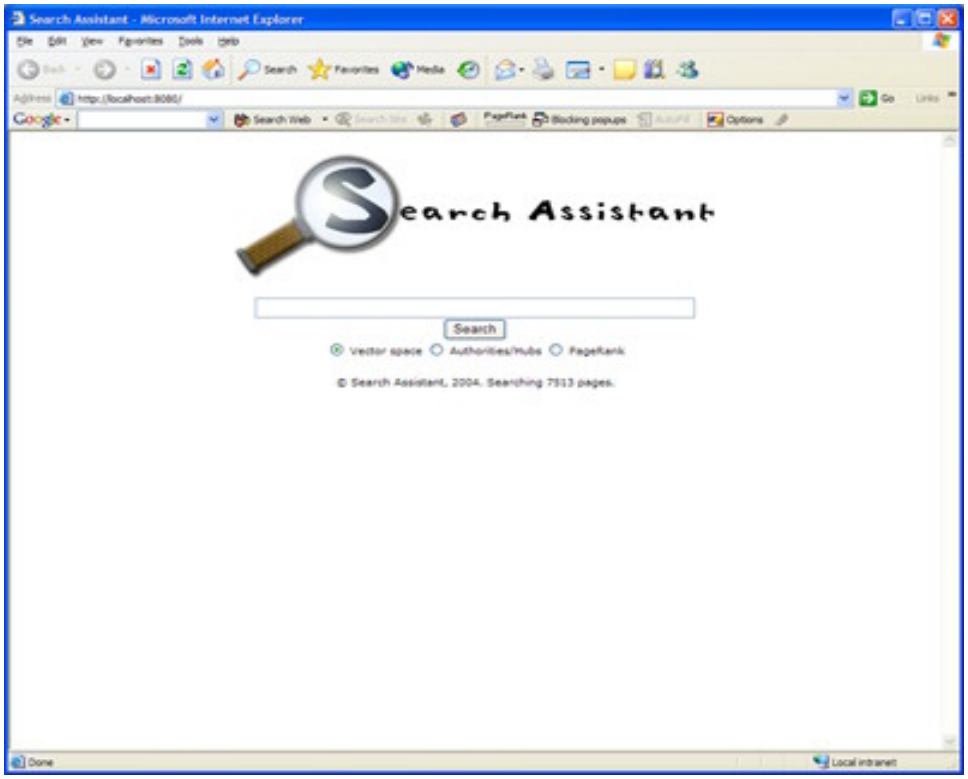
wesdyer@asu.edu

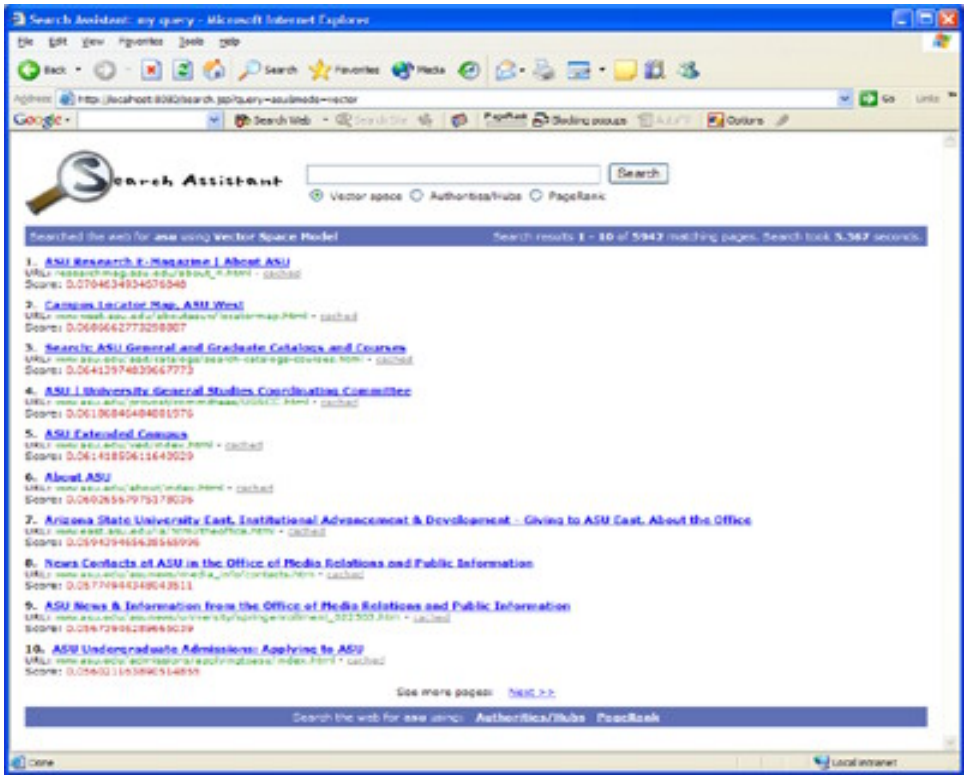
CSE 494

Project B

Introduction

Search Assistant is a small search engine that is written in Java and uses Java Server Pages (JSP) to provide a user interface. Users enter a query as a set of keywords and choose an algorithm to use to compute document scores against the given query. The selected algorithm is subsequently run and the relevant documents are sorted and returned to the user. The three algorithms that are implemented in Search Assistant are: Vector Space Model, Authorities/Hubs, and PageRank.





Search Assistant runs using a document crawl of a portion of the sites related to ASU. Although, only 7513 documents were indexed, over 9500 were crawled. Links were extracted from the latter category. Therefore, algorithms that use the link structure of the web have access to approximately 2000 more documents than those that rely upon the inverted index. The reason that this particular search engine is termed "small" is because of the size of the corpus which therefore influenced the complexity of the implementation of the various algorithms.

After the algorithms were implemented, they were extensively tested for correctness using small datasets and they were tested for performance on large datasets. This led to an iterative refinement of the implementation that has proven the correctness of the approach as well as the feasibility. An analysis of each method will be hereafter described in detail.

Due to the ease of comparing the implementations of the various algorithms, it is intriguing to analyze their comparative strengths and weaknesses against the corpus. Some tantalizing results were produced by this analysis. Finally, suggestions for future work and directions are made based upon the results of this implementation.

Vector Space Model

The Vector Space Model is based upon the previous project and a detailed analysis of its implementation and analysis has been made. A brief refresher containing the important facts from this analysis will be restated here because the vector space model is the foundation of the other two algorithms.

The Vector Space Model uses the inverted index provided by the Lucene API. Documents are represented as vectors in the space of terms. It uses the cosine theta metric to find the similarity between two documents. Recall that a query can be considered to be a document for the purposes of a similarity; therefore, query and document will be used interchangeable hereafter. Thus similarity is defined as:

$$sim(d_1, d_2) = \frac{d_1 \bullet d_2}{|d_1||d_2|}$$

Each term element in the document is computed using *tf-idf* weights. *N* is the number of documents in the corpus. N_t is the number of documents in which term *t* appears. *freq* is the frequency of term *t* in document *d*. *maxtf* is the maximum frequency of any term in document *d*. We can now define *tf-idf*.

$$tf = \frac{freq}{\max tf}$$

$$idf = \log \frac{N}{N_d}$$

$$tfidf = tf * idf$$

It was also shown that the algorithm for computing vector space similarity consisted of two components: an offline component and an online component. The offline component computed document normalization factors and max term frequencies. Previously, this computation was done when the query application was started; however, Search Assistant precomputes these values and stores them in a file that is then loaded when Search Assistant starts. The online component ran the vector space similarity algorithm and then sorted the results. It was shown that this algorithm runs in *O(N log N)* time. This is because the time it takes to sort the results is asymptotically larger than the time it takes to determine the similarity of each of the documents with the query. If the sorting is not performed then the time complexity is *O(QM)* where *Q* is the number of terms in the query and *M* is the average number of documents in which a query term appears.

The results that the vector space model returns make sense because it finds documents that are similar to the query. It has the powerful ability to find important words both in the query and in the documents and relate the two together; however, one drawback is that it would be easy to manipulate since it relies entirely upon a page's description of itself. The nature of vector space model seems to capture relevancy but has a more difficult time capturing importance or value of a document.

As previously stated, the implementation of the vector space model in Search Assistant is identical to the previous implementation except that it does not rerun the offline component each time it starts.

Authorities/Hubs

The authorities/hubs algorithm uses the link structure of the crawled hypertext pages to calculate link based statistics for each page. The algorithm starts by running the vector space algorithm with the given query. Then the top ten documents are chosen as seeds for the root set *R*. This value was chosen by experiment because it chooses a variety of pages that have a high probability of being relevant for the query and because it does not include too many documents in the root set. After the root set is chosen, it is expanded into a base set of documents *B* by the following procedure. It adds all documents that are linked to by at least one page in *R* and for each page in the *R* it adds up to fifty pages that link to that page. The reason that this is limited to fifty is that a given page can have a very high number of back links and for the given crawl any number of back links above fifty will not substantially alter the results because fifty is several standard deviations above the mean.

Now an adjacency matrix *A* is constructed from the pages in *B*. Each element of *A*, $a_{i,j}$ is set to 1 if there exists a link from page *i* to page *j* and 0 if there does not exist a link from page *i* to page *j*. Now the authority and hub vectors are initialized to $|B| \times 1$ vectors with all elements set to 1. The authority score of a page is the sum of the hub scores of the pages that link to it. The hub score of a page is the sum of the authority scores of the pages to which it links. Therefore, the authority and hub scores are then calculated thus:

$$a_i = (A^T A)^i a_0$$
$$h_i = (AA^T)^i h_0$$

Because A^TA and AA^T are symmetric and square, the equations converge to the normalized principle eigenvalue vectors Λ_1 and Λ_2 respectively. Thus we can describe the converged authorities and hub scores as the following:

$$a = \Lambda_1$$
$$h = \Lambda_2$$

Note that an eigenvector v of a matrix M is a vector such that:

$$Mv = \lambda v$$

Or in other words, multiply the matrix M by one of its eigenvectors will yield a constant multiple of the eigenvector. Note that we can find the eigenvector by iteratively finding a_i and h_i. These facts motivate the algorithm that is detailed below:

```
AuthorityHub(query q) {
    Find similar documents using vector space model
    Keep top k documents in root set R
    Let B = R
    For each page p in R {
        For each forward link from p to q {
            if q is not in B then add q to B and indicate in adjacency
            matrix
        }
        Set x to 0
        While x < 50 and there exists back links to p {
            Select a random link from q to p
            if q is not in B then {
                add q to B
                indicate in adjacency matrix
                set x = x + 1
            }
        }
    }
    Initialize all authority and hub scores to be 1/|B|
    For 50 iterations {
        For each page p in B {
            Set authority score of p to be 0
            For each page q with a link in adjacency matrix to p {
```

```
                        authority score of p += hub score of q
                    }
                }
                For each page p in B {
                    Set hub score of p to be 0
                    For each page q with a link to in adjacency matrix by p {
                        hub score of p += authority score of q
                    }
                }
                Normalize authority scores
                Normalize hub scores
            }
            Sort the authority and hub scores
        }
```

For our analysis, we will assume that we have N documents in the corpus, that each document has an average of L_f forward links and L_b back links. Then it has already been shown that the time complexity of running the vector space model is $O(N \log N)$. The time necessary for expanding the root set is:

$$O(k(L_f + L_b))$$
$$= O(L_f + L_b)$$

Thus the time complexity is linearly bounded by the sum of the number of forward links and back links to and from pages in the root set. The limit is included because although a page will not contain a very large number of forward links, this could easily happen with back links. Next, to compute the authority and hub scores the following time complexity is introduced:

$$O(50((k + kL_f + kL_b)(L_f + L_b)))$$
$$= O({L_f}^2 + {L_b}^2 + 2L_f L_b + L_f + L_b)$$
$$= O({L_f}^2 + {L_b}^2)$$

The time complexity to sort the authority and hub scores will be:

$$O((L_f + L_b)\log(L_f + L_b))$$

Therefore, the total time for the algorithm is described below:

$$O(N \log N + L_f + L_b + {L_f}^2 + {L_b}^2 + (L_f + L_b)\log(L_f + L_b))$$
$$= O(N \log N + {L_f}^2 + {L_b}^2)$$

In a very large corpus where $N>>(L_f+L_b)$ this becomes:

$$O(N \log N)$$

In a small highly connected corpus this becomes:

$$O(L_f{}^2 + L_b{}^2)$$

This demonstrates the fact that while the authority/hubs computation is intensive, the nature of selecting the root set by picking the top *k* documents fixes the time complexity to some extent. Therefore, this algorithm scales well. In fact, the algorithm proportionally takes much longer in a small corpus than in a larger one. In a very large corpus the authority/hubs score will be bounded by the time taken to run the vector space model prior to the authority/hubs computation.

The authority/hub computation captures some of the importance of documents. It relies upon the assumption that forward links and back links from relevant pages will be relevant as well. In a normally distributed population where the graph of the pages is loosely connected overall this will hold; however, in a small strongly connected graph many of the pages which are linked to and linked from the root set will not be relevant to the query. Another problem with the authority/hub scores is that they are not stable in a changing corpus.

PageRank

PageRank is something of the best of both the vector space model and the authority/hub scores. Note that from the previous two types of query evaluation we have seen that vector space model captures relevancy whereas the authority/hub scores capture importance. We have also noted that the two notions are not always related and this accounts from the principle weakness of each approach. PageRank addresses these issues. It scores documents based on a weighted combination of their vector space score and their PageRank. PageRank is essentially the normalized probability that a web surfer would be at a particular page and a given instant. This model of PageRank provides the basis for its computation. It begins by defining a large square stochastic matrix where each entry $a_{i,j}$ represents the probability that from page j a surfer will move to page i. Note that each column must therefore sum to 1. To avoid rank sinks which are pages with back links but no forward links it is assumed that the web surfer will not only follow links but also will become bored and jump to a new random page at any given time. Thus the matrix can be modeled thus:

$$M^* = \alpha(M + Z) + (1-\alpha)K$$

In this equation, *M* represents the stochastic matrix where each element $m_{i,j}$ is the number of links to page *i* from page *j* divided by the number of forward links from page *j*. Each column *j* of matrix *Z* is set to 0 if page *j* has any forward links otherwise all elements in that column are set to 1 divided by number of pages. The matrix K has each element set to 1 divided by the number of pages. Finally, α is defined as a number between 0 and 1 inclusively. It is therefore, the probability that a web surfer will follow a link on the current page as opposed to jumping to a new page. Now we define the PageRank of each page to be the corresponding element in the principle eigenvector of the M^* matrix. Note that this can be computed by iteratively calculating the PageRank until it converges. It has been shown that this convergence happens quickly. The total score of a document is a weight combination of the PageRank of a page and the vector space score, where the weighted coefficients sum to 1.

Now the algorithm can be defined. Note that the PageRank algorithm consists of two components: an offline component and an online component. First the PageRank of each page is computed offline and stored for query evaluation. Second, queries are evaluated with the vector space model and the weighted combination of the vector space score and the PageRank are used to sort the documents. The input, α, is used to determine to probability that a user will follow a link on the page rather than jumping.

```
PageRankCompuation(α) {
    Construct a NxN matrix, M, where N is the number of pages
    Set all elements in M to 0
    For each page p in the corpus {
        if p has no forward links {
            Set all entries in column p of M to 1/|N|
        }
        else {
            For each page q that is linked to from p {
                Set element q, p in M to 1 divided by number of
                forward links from p
            }
        }
    }
```

```
For each element in matrix M {
        Set the element to α(element) + (1 − α)(1 / |N|)
}
Set PageRank of each page to 1/|N|
Iterate 20 times {
        Foreach page p {
                Set New PageRank p to 0
                Foreach page q {
                        New PageRank p += PageRank q * element q, p
                }
        }
        Normalize PageRank
}
Write results
}
```

Therefore, the time complexity is:

$$O(N^2 + NL_f + N^2 + 20N^2 + N)$$
$$= O(N^2)$$

Even for a crawl of only 7513 pages this represents an enormous cost. Fortunately as it has already been noted, the PageRank scores are computed offline. The algorithm for computing the weighted combined score is listed below. The inputs are the query and β which is the weight given to the vector score rather than the page rank.

```
PageRankScore(query q, β) {
        VectorScore of each page = VectorSpaceModel(q)
        For each page p {
                Score of p = (1 - β) (VectorScore of p) + β (PageRank of p)
        }
        Sort pages by score
}
```

Thus, the time complexity can be modeled by:

$$O(N \log N) + O(N) + O(N \log N) = O(N \log N)$$

For Search Assistant, α was assigned the value .8 and β was assigned the value .9. This means that for our model a user will follow a link with 80% probability and will jump to a random page with 20% probability. Also the total score is 30% of

the vector space score and 70% of the page rank score. Both of these values were determined by experimentation. The values were varied and the precision of the top ten results was analyzed and refined. It should be noted that the PageRank scores tend to be an order of magnitude smaller than the vector space score so the large difference in percentage contribution helps to balance the scores. PageRank is a powerful concept. It directly includes the relevancy of the terms and provides a powerful model for favoring important documents.

Conclusion

In order to compare the three methods of evaluating a query, several queries have been selected and the results of each method are listed. The results from these queries are attached to this paper. From these results an assortment of conclusions can be drawn about the data set and the algorithms.

First, note that the vector space algorithm is fairly accurate; however, it does favor shorter more focused documents. It should also be noted that although this algorithm is more easily manipulated than the other two this was not exploited in the data set that was given.

Second, the authority and hubs scores seem to give results that are not very accurate. This is due to the nature of the data set. Because the authority and hub scores are highly dependent on common links between pages in the base set and because the data set is a small highly connected graph, most of the common links between pages are administrative pages. In a larger data set the common links between relevant pages will more likely be relevant as well. Of the two metrics, the authority score seems to be more reliable but again it favors administrative pages in this data set.

Third, the PageRank scores in general are better than the vector space scores. This is because the PageRank scores tend to weed out pages that are relevant but are not the important pages that were desired. It boosts the scores of popular pages; however, PageRank also suffers from a highly connected graph.