Comparison of Text Search Ranking Algorithms

by

Dan Han and Bradford G. Nickerson

TR11-209, September 6, 2011

Faculty of Computer Science
large University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada
Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
http://www.cs.unb.ca

1 Introduction

It is challenging when we try to retrieve information from the web since it has features that make efficient search hard. First, the web has a very large set of web pages with little systematically organization. Current estimates are that there are at least 14×10^9 web pages pages with a life of less than one year [12][6]. In addition, anyone can add a document to the web at any time without permission. This also leads to extreme diversity of web pages. The content of web pages can represent any text, image, music, video, information source or any combination of them.

Most search engines, including Google, run programs that retrieve pages from the web, index the words in each document, and store this information in an efficient format [8]. With the large number of web pages, for most searches, there will be a huge number of pages containing the words in a search phrase. Thus, it is important to have a good ranking mechanism that fits the search criteria so that the resulting pages can be sorted based on their importance.

2 PageRank algorithm

PageRank is used by Google together with a number of different factors, including standard information retrieval (IR) measures, proximity, and anchor text (text of links pointing to Web pages) in order to find most relevant answers to a given query [9]. Other than document collections, web pages on the web are hypertext and provide plenty of auxiliary information in the metadata of the web pages, such as link structure and link text [12]. The PageRank algorithm makes use of these features. The algorithm is based on the directed graph created by treating web pages as nodes and hyperlinks as edges [5]. Google's PageRank algorithm assesses the importance of web pages without human evaluation of the content. Google claims, "the heart of our software is PageRank" [8].

The basic idea behind PageRank is that a page is ranked higher if there are more links to it. More specifically, PageRank is a probability distribution which is created to represent the likelihood that a person randomly clicking on links will arrive at any particular page [5]. A probability is expressed as a numeric value between 0 and 1. A PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank [8]. In this way, web pages with the highest PageRank value will apear on the top of the search results.

2.1 Basic algorithm

Suppose the web of interest has n pages, each page represented by P_k where k is an integer and $1 \leq k \leq n$. In this report, we represent the PageRank of a web page P_k

by $W_{pr}(P_k) \in [0,1]$, which is a measure of its importance. $W_{pr}(P_k)$ is non-negative and $W_{pr}(P_k) > W_{pr}(P_j)$ indicates that page P_k is more important than page P_j. $W_{pr}(P_k) = 0$ implies that page P_k has the smallest ranking value. Assume page P_j contains L_j outgoing links. If one of those links is to page P_i, then P_j will pass on $\frac{1}{L_j}$ of its importance to P_i. The rank of P_i is the sum of all the contributions made by pages linking to it. The links to a given page are called the backlinks for that page [11]. In this web of n pages, let $S_i \subset \{1, 2, ..., n\}$ denote the set of pages with a link to page i, that is, S_i is the set of page i's backlinks. For each i, we require

$$W_{pr}(P_i) = \sum_{P_j \in S_i} \frac{W_{pr}(P_j)}{L_j} \tag{1}$$

where $L_j \in \mathbb{N}^+$ since if $P_j \in S_i$ then page P_j links to at least page P_i [11].

We then create a matrix, called the hyperlink matrix $H = [H_{ij}]$ of size (n, n) in which the entry in the i^{th} row and j^{th} column is as follows:

$$\text{if } P_j \in S_i, H_{ij} = 1/L_j; \text{ otherwise, } H_{ij} = 0 \tag{2}$$

This matrix has some special properties [8] as follows:

1. Its entries are all nonnegative.
2. The sum of the entries in a column is one unless the page corresponding to that column has no links, in which case the sum of entries = 0.

Matrices with these properties are called stochastic. We also form a vector $\mathbf{r} = [W_{pr}(P_i)]$ whose elements are PageRanks — the importance rankings of all pages.

Equations (1) and (2) above defining PageRank can be expressed as (see e.g. [8], [11]):

$$\mathbf{Hr} = \mathbf{r} \tag{3}$$

So the vector $\mathbf{r}$ is an eigenvector of the matrix $\mathbf{H}$ with eigenvalue 1. As an example, figure 1 shows a small web of 4 pages with 8 links represented by arrows.

The corresponding matrix is:

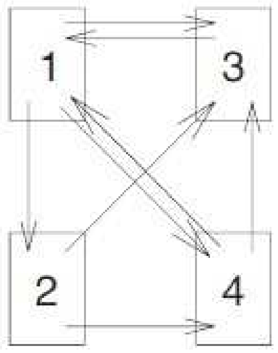


Figure 1: A simple web with only four pages. An arrow from page A to page B indicates a link from page A to page B (from [11]).

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \text{ with eigenvector } \mathbf{r} = \begin{bmatrix} 0.387 \\ 0.129 \\ 0.290 \\ 0.194 \end{bmatrix}. \tag{4}$$

The values L_j for the example in Figure 1 are $[3, 2, 1, 2]$ corresponding to the outdegree of each vertex in the graph.

2.2 Algorithm with damping factor

Two difficulties arise with using the simplified ranking function in equation (3) to rank websites. There are webs with non-unique rankings and webs with dangling nodes (nodes with outdegree = 0) [11]. To overcome these problems, the way a random surfer moves through the web is modified, and the damping factor is introduced. The PageRank theory holds that even an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue clicking is a damping factor α [5]. With probability α the person is guided by matrix $\mathbf{H}$. With probability $1 - \alpha$ the person chooses the next page at random [8]. Various studies have tested different damping factors, but it is generally assumed that a damping factor of around 0.85 works well [10].

Equation (1) is modified as follows:

$$W_{pr}(P_i) = \frac{1-\alpha}{n} + \alpha \sum_{P_j \in S_i} \frac{W_{pr}(P_j)}{L_j} \tag{5}$$

where $P_1, P_2, ...P_n$ are pages in the web of interest, S_i is the set of pages that link to P_i, L_j is the number of outbound links on page P_j, and n is the total number of pages in the web. Let $\mathbf{U}$ be a (positive) uniform transition probability matrix with $u_{ij} = 1$ for all i and j. We can replace matrix $\mathbf{H}$ with the matrix $\mathbf{M}$, as follows [11]:

$$\mathbf{M} = \frac{1-\alpha}{n}\mathbf{U} + \alpha\mathbf{H} \tag{6}$$

The PageRank values are the entries of the dominant eigenvector of the modified matrix $\mathbf{M}$ in equation (6). The eigenvector is

$$\mathbf{r} = \begin{bmatrix} W_{pr}(P_1) \\ W_{pr}(P_2) \\ \vdots \\ W_{pr}(P_n) \end{bmatrix} \tag{7}$$

Then $\mathbf{r}$ is the solution of equation (8) shown below:

$$\mathbf{r} = \frac{1-\alpha}{n}\mathbf{U} + \alpha\mathbf{H}\mathbf{r} = \frac{1-\alpha}{n}\mathbf{U} + \alpha \begin{bmatrix} H(P_1,P_1) & H(P_1,P_2) & \cdots & H(P_1,P_n) \\ H(P_1,P_1) & \ddots & & \vdots \\ \vdots & & H(P_i,P_j) & \\ H(P_n,P_1) & \cdots & & H(P_n,P_n) \end{bmatrix} \mathbf{r} \tag{8}$$

where the adjacency function $H(P_i, P_j)$ is 0 if page P_j does not link to P_i, and normalized such that, for each j,

$$\sum_{i=1}^{n} H(P_i, P_j) = 1 \tag{9}$$

i.e. the elements of each column sum up to 1. The matrix $\mathbf{H}$ is a stochastic matrix [5], as already mentioned in equation (2).

Equation (6) is based on a model of a random surfer. The random surfer simply keeps clicking on successive links at random. However, if a real Web surfer ever gets into a small loop of web pages, it is unlikely that the surfer will continue in the loop forever. Instead, the surfer will jump to some other page [12]. If a page has no links to other pages, it becomes a sink and therefore terminates the random surfing process. If the random surfer arrives at a sink page, it picks another URL at random and continues surfing again [5].

For the simple web of four pages in Figure 1 with matrix $\mathbf{H}$ given by (4), the new formula gives (with $\alpha = 0.85$)

$$\mathbf{M} = \begin{bmatrix} 0.0375 & 0.0375 & 0.8875 & 0.4625 \\ 0.32083 & 0.0375 & 0.0375 & 0.0375 \\ 0.32083 & 0.4625 & 0.0375 & 0.4625 \\ 0.32083 & 0.4625 & 0.0375 & 0.0375 \end{bmatrix} \text{with eigenvector } \mathbf{r} = \begin{bmatrix} 0.368 \\ 0.142 \\ 0.288 \\ 0.202 \end{bmatrix} \tag{10}$$

This yields the same overall ranking of pages as equation (4), but the ranks are slightly different [11]. Equation (8) is always used to compute the PageRank vector $\mathbf{r}$ as it handles the dangling node and non-unique rank issues.

PageRank or $\mathbf{r}$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web (see e.g. [10] and [5]).

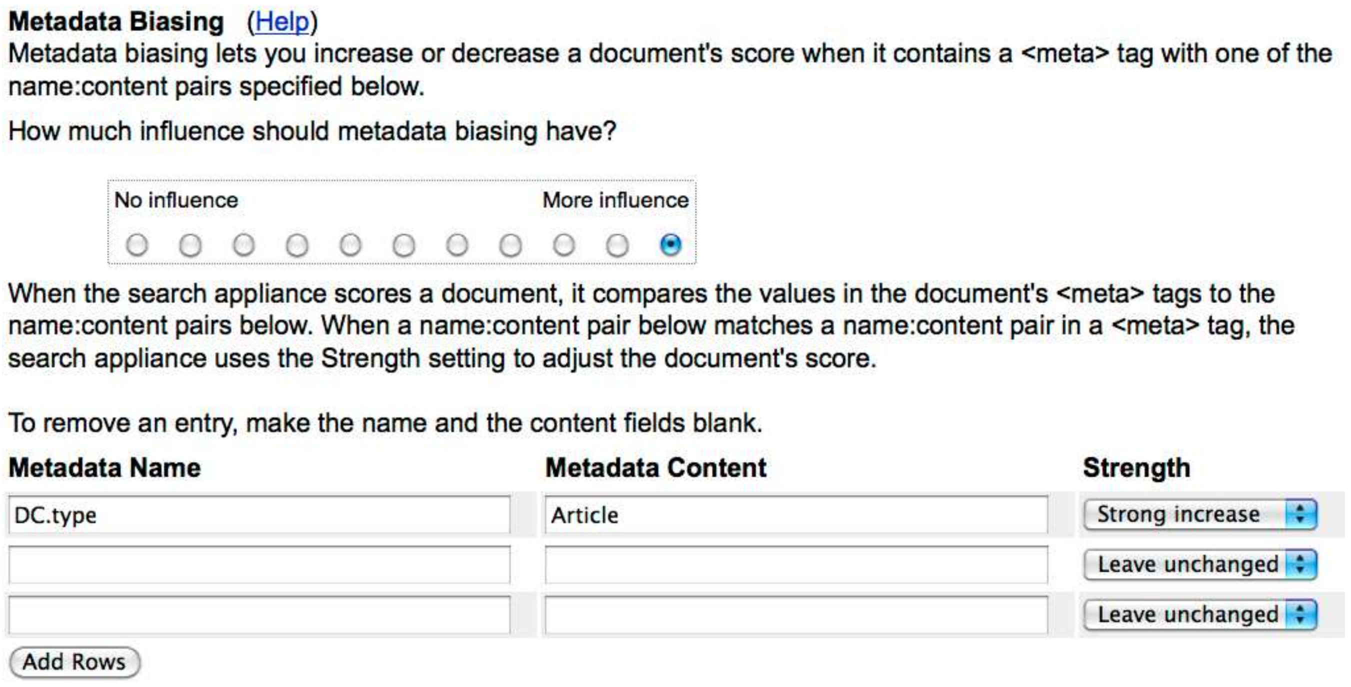
3 Result biasing

The Google Search Appliance (GSA) provides multiple ways to exert influence on the search results. This includes Source Biasing, Date Biasing and Metadata Biasing. Metadata Biasing enables us to influence the order of documents based on metadata associated with a document. For example, we might use metadata biasing to increase the score of documents whose document type is “Article”. We can create a biasing scheme with the user interface depicted in Figure 2.

To use Metadata Biasing, we can first change the influence setting from “No influence” to a stronger setting. We can make a specific adjustment by compiling a list of metadata tags. Documents are modified to include metadata tags corresponding to the metadata attributes described in the Metadata Biasing entries. As explained in [1], a metadata tag contains a name-value pair. An example name-value pair is as follows:

```
<meta name="DC.type" content="Article" xml:lang="en">        (11)
```

We have seven choices on the influence strength of each tag: Strong decrease, Medium decrease, Weak decrease, Leave unchanged, Weak increase, Medium increase and Strong increase. We assume these tags have values $-d_3, -d_2, -d_1, 0, d_1, d_2, d_3$, respectively, where d_1, d_2 and d_3 are $\in \mathbf{R}^+$, and $d_1 < d_2 < d_3$. When the search appliance ranks search results,


Metadata Biasing (Help)

Metadata biasing lets you increase or decrease a document's score when it contains a <meta> tag with one of the name:content pairs specified below.

How much influence should metadata biasing have?

No influence ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ [✓ ●] More influence

When the search appliance scores a document, it compares the values in the document's <meta> tags to the name:content pairs below. When a name:content pair below matches a name:content pair in a <meta> tag, the search appliance uses the Strength setting to adjust the document's score.

To remove an entry, make the name and the content fields blank.

Metadata Name	Metadata Content	Strength
DC.type	Article	Strong increase
		Leave unchanged
		Leave unchanged

Figure 2: The Metadata biasing user interface from a GSA model GB-7007-1M running version 6.8.0.G.30 of the GSA software.

it compares metadata tags with each pattern in the list. For each document, the search appliance traverses the list in the order we specify the metadata tags from top to bottom, and compares the tags with the document's metadata. The search appliance makes only one score adjustment for each document. Once a tag matches a document, the score of the document is modified, and the search appliance continues with the next document to be rescored, to see if the document matches any metadata tag [1].

Assume for a specific web page P, the PageRank value is $W_{pr}(P)$. The general degree of influence that Metadata Biasing has is represented by f, which is one of 11 nonnegative numbers $\in [0, f_{max}]$. The score of the first matched tag in the document's metadata is represented by W_{mb}. As there are seven possible degrees of strength for each tag, the domain of W_{mb} should be seven numbers corresponding to seven degrees as described above. $W_{mb} = 0$ corresponds to the strength of a tag setting to "Leave unchanged". Based on the documentation we can find, we estimate that the final rank R_i of web page P_i is as follows:

$$R_i = W_{pr}(P_i) + fW_{mb}(P_i) \tag{12}$$

For the example in Figure 1 and equation (10), the PageRanks for page 1, 2, 3 and 4 are $W_{pr}(1) \approx 0.368$, $W_{pr}(2) \approx 0.142$, $W_{pr}(3) \approx 0.288$ and $W_{pr}(4) \approx 0.202$, respectively.

The ranking of these pages is 1, 3, 4, 2 from top to bottom. Now assume we have the name:content pair in the Metadata Biasing list, which is meta name = "DC.type" and meta content = "Article". The strength is set to "Strong increase". In the four pages, we assume only pages 2 and 3 have metadata tags that agree with this name:content pattern. If our equation (12) is correct, the search appliance will make score adjustments for pages 2 and 3 as follows:

$R_2 = W_{pr}(2) + fW_{mb}(2) = 0.142 + fW_{mb}(2)$
$R_3 = W_{pr}(3) + fW_{mb}(3) = 0.288 + fW_{mb}(3)$

Assuming there is no other metadata tags in the list in Metadata Biasing scheme, then the search appliance will not make changes to the scores of pages 1 and 4. The final scores for these two pages are as follows:

$R_1 = W_{pr}(1) = 0.368$
$R_4 = W_{pr}(4) = 0.202$

Consider a Metadata Biasing scheme in which the general degree of influence is set to the strongest degree. Assuming that $f \in [0, 1]$, then in this situation, $f = 1$. We assume the values $(d_1, d_2, d_3) = (0.05, 0.10, 0.15)$. With a W_{mb} value of "Strong increase" $= d_3$ for pages 2 and 4, we have the following four final ranks:

$R_1 = 0.368$
$R_2 = 0.142 + 1 * 0.15 = 0.292$
$R_3 = 0.288 + 1 * 0.15 = 0.438$
$R_4 = 0.202$

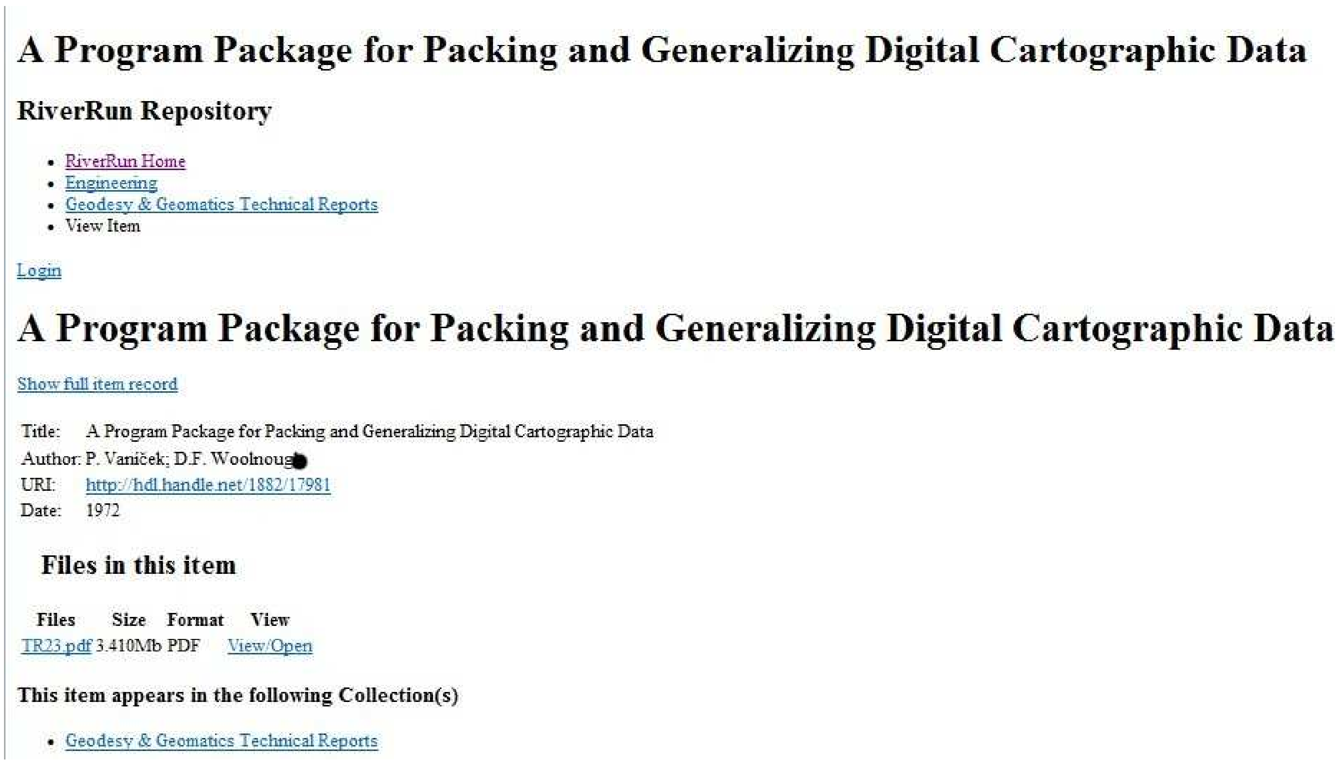
Thus, the final ranking of these four pages is 3, 1, 2, 4 from top to bottom. We can see that after Metadata Biasing, the scores of pages 2 and 3 are boosted, and the rank of every page has changed from that arising from equation (10) (i.e. rankings of 1, 3, 4, 2).

4 Experimental results and discussion

We performed an experiment to see if equation (12) is the way GSA actually ranks documents. We first built a web page collection of 100 web pages, all of which come from http://dspace.hil.unb.ca:8080/handle/1882/11037, which is a collection of Geodesy and Geomatics Technical Reports at UNB. One example web page from this collection is shown in Figure 3. These pages are indexed by 1, 2,..., 100.

This collection ($n = 100$) has no links among them (there are links to web pages outside this collection). Thus, we except $\mathbf{r} = [c, c, ..., c]$, i.e. all PageRanks computed using equation (8) are the same value (here indicated as c). In the metadata of each page, there is a name-value pair as follows:

```
<meta name="DC.type" content="Technical Report" xml:lang="en" />    (13)
```


A Program Package for Packing and Generalizing Digital Cartographic Data

RiverRun Repository

- RiverRun Home
- Engineering
- Geodesy & Geomatics Technical Reports
- View Item

Login

A Program Package for Packing and Generalizing Digital Cartographic Data

Show full item record

Title:	A Program Package for Packing and Generalizing Digital Cartographic Data
Author:	P. Vaníček; D.F. Woolnough
URI:	http://hdl.handle.net/1882/17981
Date:	1972

Files in this item

Files	Size	Format	View
TR23.pdf	3.410Mb	PDF	View/Open

This item appears in the following Collection(s)

- Geodesy & Geomatics Technical Reports

Figure 3: An example web page from the web page collection used in the experiment.

We modified the first 10 pages to change the name-value pairs to that shown in (11). A Metadata Biasing scheme was created, in which the overall strength of influence is set to have the stongest influence. An entry was added to the Metadata Biasing scheme, which has Metadata Name = "DC.type" and Metadata Content = "Article", as shown in Figure 2. We then searched for the string "and" within this web page collection with the strength of the Metadata Biasing tag set to seven different degrees (Strong Increase down to Strong Decrease). All the 100 web pages in the collection appeared in the search results and we checked if there is any influence on the rankings of the first ten pages.

The results are shown in Table 1.

As we can see from Table 1, if the strength of the tag in Metadata Biasing is set to Strong increase, Medium increase or Weak increase, the final ranking of the first 10 pages will be boosted. If the strength is set to Strong decrease, Medium decrease or Weak decrease, the final ranking is reduced. The stronger the influence, the more the final ranking will be affected. Even if the tag is set to strong decrease or strong increase, the score of Metadata Biasing W_{mb} will still not be the dominant part in the final score. This can be known from the fact that not all 10 pages with name-value pairs as in (11) appear in the top 10 or bottom 10 in the final ranking with Metadata biasing of strong increase or strong decrease.

Table 1: Experimental results of GSA ranking using search string "and" with Metadata Biasing.

Index of Page	**Ranking**						
	Strong Increase	*Medium Increase*	*Weak Increase*	*Leave Unchanged*	*Weak Decrease*	*Medium Decrease*	*Strong Decrease*
6	1	1	1	2	7	25	25
9	2	2	2	7	8	26	26
1	3	3	8	13	27	27	27
10	18	28	28	34	94	94	94
8	19	29	29	54	95	95	95
7	20	30	30	57	96	96	96
5	21	31	31	61	97	97	97
3	22	32	32	69	98	98	98
4	23	33	33	80	99	99	99
2	24	34	34	97	100	100	100

Our equation (12) holds for the example data explored here. The ranking provided by the GSA as shown in Table 1 clearly consider additional factors, beyond what we consider here. If all 100 PageRanks from equation (8) are the same, then the first 10 pages should all be ranked in the top 10 for any positive strength (i.e. Strong, Medium or Weak increase). The rank of each page in the collection is clearly being computed using a different equation.

To test if the search string "and" was a member of the stop word set, and being ignored, we searched using the search string "Geodesy and Geomatics", which is present in all 100 web pages. The results are shown in Table 2.

As we can see from Table 2, if the strength of the tag in Metadata Biasing is set to Strong increase, Medium increase or Weak increase, the final rankings of the first 10 pages are in the top 10. If the strength is set to Strong decrease, Medium decrease or Weak decrease, the final rankings are in the bottom 10. This result is obviously different from the results shown in Table 1, and corresponds with equations (8) and (12). Thus, we can know that the string "and" is a member of the stop word set and was treated differently by the search appliance.

5 Lucene Ranking Algorithm

Appach Lucene[2] is an open source information retrieval (IR) software library, originally created by Doug Cutting [3]. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform [2]. The Lucene ranking algorithm ranks

Table 2: Experimental results of GSA ranking using search string "Geodesy and Geomatics" with Metadata Biasing.

Index of Page	**Ranking**						
	Strong Increase	*Medium Increase*	*Weak Increase*	*Leave Unchanged*	*Weak Decrease*	*Medium Decrease*	*Strong Decrease*
10	1	1	1	9	91	91	91
8	2	2	2	34	92	92	92
7	3	3	3	37	93	93	93
1	4	4	4	38	94	94	94
6	5	5	5	40	95	95	95
5	6	6	6	44	96	96	96
3	7	7	7	56	97	97	97
9	8	8	8	69	98	98	98
4	9	9	9	71	99	99	99
2	10	10	10	98	100	100	100

documents resulting from a search query, based on their content. It is a good example of vector space model widely used by TREC (Text REtrieval Conference) systems (e.g. , [13], [15]). At the core of Lucene's logical architecture is the idea of a document containing fields of text. This flexibility allows Lucene's API to be independent of the file format. Text from PDFs, HTML, Microsoft Word, and OpenDocument documents, as well as many others, can all be indexed as long as their textual information can be extracted [3].

Lucene provides a scoring algorithm to find the best matches to document queries. The default scoring algorithm considers such factors as the frequency of a particular query term with individual documents and the frequency of the term in the total population of documents. The Lucene scoring algorithm considers the rarity of a matched term within the global space of all terms for a given field. In other words, if you match a term that is not very common in the data then this match is given a higher score [7].

Terms can be considered as single words in most IR systems. Stop-words are words considered non-informative, like words the, in, of, ..., which are often ignored. The search word "and" used in Table 1 is usually considered as a stop word [13]. Also, term, token, and word are often used interchangeably [14]. For a query q, let t represent a search term in q and d represent a document of interest. To illustrate Lucene's scoring algorithm, we have definitions as follows:

m : number of terms in the search query q

D : set of documents in index, so we have $n = |D|$

D_t: number of documents containing term t

The ranking score of query q for document d correlates to the cosine-distance or dot-product between document and query vectors in a Vector Space Model (VSM) of Information Retrieval. A document whose vector is closer to the query vector in that model is scored higher. The score is computed as follows [4]:

$$R(q,d) = C(q,d)N(q)\sum_{t \text{ in } q}(T(t,d)I(t)^2B(t)N(t,d)) \tag{14}$$

The factors influencing the score are as follows:

5.1 $T(t,d)$

$T(t,d)$ correlates to the term's frequency $F(t,d)$, defined as the number of times term t appears in the currently scored document d. Documents that have more occurrences of a given term receive a higher score. The default computation for $T(t,d)$ in DefaultSimilarity is: [4]

$$T(t,d) = \sqrt{F(t,d)} \tag{15}$$

The more times a term is found in a document's field, the higher the score it gets. This concept is most intuitive. Obviously, it doesn't matter how many times the term may appear in some other field, it's the searched field that is relevant (whether explicitly targeted or the default) [14].

5.2 $I(t)$

$I(t)$ stands for Inverse Document Frequency. This value correlates to the inverse of D_t (the number of documents in which the term t appears). This means rarer terms give higher contribution to the total score. The default computation for $I(t)$ in DefaultSimilarity is: [4]

$$I(t) = 1 + \log(\frac{n}{D_t + 1})) \tag{16}$$

It is the inverse of the document frequency that is positively correlated with the score. In short, rare terms result in higher scores [14].

5.3 $C(q,d)$

$C(q,d)$ is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms. This is a search time factor computed in $C(q,d)$ by the Similarity in effect at search time [14].

According to Lucene java doc, the method for $C(q,d)$ is: public abstract float coord(int overlap,int maxOverlap). This method computes a score factor based on the fraction of all query terms that a document contains. This value is multiplied into scores. The presence of a large portion of the query terms indicates a better match with the query, so implementations of this method usually return larger values when the ratio between these parameters is large and smaller values when the ratio between them is small [4].

If we use $m(q,d)$ to represent the number of query terms matched in the document, then we can compute $C(q,d)$ as follows:

$$C(q,d) = \frac{m(q,d)}{m} \tag{17}$$

Obviously, $C(q,d)$ must be a real number $\in [0,1]$. The greater the number of queried terms match, the greater the score, all things being equal otherwise. Any mandatory clauses must match and the prohibited ones must not match, leaving the relevance of this piece of the score to situations where there are multiple optional clauses [14].

5.4 $B(t)$

$B(t)$ is the user-specified boost factor for search term t. It can be computer by the method t.getBoost(). $B(t)$ is a search time boost of term t in the query q as specified in the query text (see query syntax), or as set by application calls to setBoost() . Notice that there is really no direct API for accessing a boost of one term in a multi term query, but rather multi terms are represented in a query as multi TermQuery objects, and so the boost of a term in the query is accessible by calling the sub-query getBoost() [4].

TermQuery is defined as: a query that matches documents containing a term. This may be combined with other terms with a BooleanQuery [4].

5.5 $N(q)$

Query normalization $N(q)$ is a normalizing factor used to make scores between queries comparable. This factor does not affect document ranking (since all ranked documents are multiplied by the same factor), but rather just attempts to make scores from different queries (or even different indexes) comparable. This is a search time factor computed by the Similarity in effect at search time. The default computation in DefaultSimilarity is [4]:

$$N(q) = N(\text{sumOfSquaredWeights}) = \frac{1}{\text{sumOfSquaredWeights}} \tag{18}$$

The sum of squared weights (of the query terms) is computed by the query Weight object. For example, a boolean query computes this value as [4]:

$$\text{sumOfSquaredWeights} = B(q)^2 \sum_{t \text{ in } q} (I(t)B(t))^2 \tag{19}$$

A boolean query is defined as: a query that matches documents matching boolean combinations of other queries, e.g. TermQuerys, PhraseQuerys or other BooleanQuerys [4].

$B(q)$ is the boost factor for query q, it can be obtained by the method q.setBoost and q.getBoost. The setBoost and getBoost method for a query q is defined as following [4]:

public void setBoost(float b):
: Sets the boost for this query clause to b. Documents matching this clause will (in addition to the normal weightings) have their score multiplied by b.

public float getBoost():
: Gets the boost for this clause. Documents matching this clause will (in addition to the normal weightings) have their score multiplied by b. The boost is 1.0 by default.

5.6 $N(t, d)$

$N(t, d)$ encapsulates a few (indexing time) boost and length factors:

5.6.1 Document boost $B(d)$

$B(d)$ is set by calling d.setBoost() before adding the document to the index. Call d.getBoost to get the document boost.The method setBoost for documents is defined as following [4]:

public void setBoost(float boost):
: Sets a boost factor for hits on any field of this document. This value will be multiplied into the score of all hits on this document. The default value is 1.0.

5.6.2 Field boost $B(f)$

$B(f)$ is set by calling f.setBoost() before adding the field to a document. Call f.getBoost to get the field boost. The method getBoost for a field f is defined as following [4]:

float getBoost():
: Returns the boost factor for hits for this field. The default value is 1.0. Note: this value is not stored directly with the document in the index. Documents returned from IndexReader.document(int) and Hits.doc(int) may thus not have the same value present as when this field was indexed.

5.6.3 Field Length Normalization: $L(f)$

computed when the document is added to the index in accordance with the number of tokens of this field in the document, so that shorter fields contribute more to the score. $L(f)$ is computed by the Similarity class in effect at indexing.

The shorter the matching field is (measured in number of indexed terms), the greater the matching document's score will be. For example, if there was a band named just Smashing, and another named Smashing Pumpkins, then this factor in the scoring would be higher for the first band upon a search for just Smashing, as it has one word, while the other has two [4].

$L(f)$ is implemented as follows [4]:

$$L(f) = \frac{1}{\sqrt{\text{numTerms}}} \tag{20}$$

We use the method lengthNorm to compute $L(f)$, which is defined as : public abstract float lengthNorm(String fieldName, int numTokens) [4].

Parameters are defined as follows:

Parameters:
  fieldName: the name of the field
  numTerms: the total number of tokens contained in fields named fieldName of doc.

Returns:
  a normalization factor for hits on this field of this document

5.6.4 Summary for $N(t,d)$

When a document is added to the index, all the above factors are multiplied. If the document has multiple fields with the same name, all their boosts are multiplied together as follows [4]:

$$N(t,d) = B(d)L(f) \prod_{\text{field } f \text{ in } d = t} B(f) \tag{21}$$

However the resulted norm value is encoded as a single byte before being stored. At search time, the norm byte value is read from the index directory and decoded back to a float norm value. This encoding/decoding, while reducing index size, comes with the price of precision loss - it is not guaranteed that decode(encode(x)) = x. For instance, decode(encode(0.89)) = 0.75. Also notice that search time is too late to modify this norm part of scoring, e.g. by using a different Similarity for search [4].

6 Use of Lucene and Solr by Synergies

In the Synergies' XSLT transformation that converts articles into Solr readable format, we can find .xsl statements as follows:

```
<xsl:variable name="boostDocType">
   <xsl:choose>
```

```
      <xsl:when test="/unit:unit/@unittype = ‘article’">3</xsl:when>
      <xsl:when test="/unit:unit/@unittype = ‘review’">2</xsl:when>
      <xsl:when test="/unit:unit/@unittype = ‘note’">2</xsl:when>
      <xsl:when test="/unit:unit/@unittype = ‘other’">1</xsl:when>
      <xsl:otherwise>1</xsl:otherwise>
   </xsl:choose>
</xsl:variable>
```
There is a variable name = boostDocType, if the unittype is ‘article’ for a document, then the boostDocType will be set to 3. Correspondingly, the value of the factor B_j for term ‘article’ will be set to 3.

7 Summary

Results in Tables 1 and 2 can be explained successfully by the formula we proposed in (12). Thus, formula (12) is a reasonable assumption for how the GSA ranks search result with metadata biasing. We note that the GSA model presented here can also be used to rank documents, or a combination of documents and web pages.

References

[1] http://code.google.com/apis/searchappliance/documentation/610/. accessed May 22, 2011.

[2] Apache lucene. http://lucene.apache.org. accessed June 22, 2011.

[3] Lucene. http://en.wikipedia.org/wiki/Lucene. accessed June 22, 2011.

[4] Lucene java doc, class similarity. http://lucene.apache.org/java/2_4_1/api/org/apache/lucene/search/Similarity.html. accessed July 18, 2011.

[5] Pagerank. http://en.wikipedia.org/wiki/PageRank. accessed May 22, 2011.

[6] The size of the world wide web. http://www.worldwidewebsize.com/. accessed May 23, 2011.

[7] Lucene as a ranking engine. http://www.wortcook.com/pdf/lucene-ranking.pdf, November 2004. accessed June 23, 2011.

[8] David Austin. How google finds your needle in the web’s haystack. http://www.ams.org/samplings/feature-column/fcarc-pagerank. accessed May 22, 2011.

[9] Monica Bianchini, Marco Gori, and Franco Scarselli. Inside pagerank. *ACM Trans. Internet Technology*, 5(1):92–128, 2005.

[10] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[11] Kurt Bryan and Tanya Leise. The $25,000,000,000 eigenvector: The linear algebra behind google. *SIAM Review*, 48:569–581, March 2006.

[12] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[13] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[14] David Smiley and Eric Pugh. *Solr 1.4 Enterprise Search Server.* Packt Publishing, 2009.

[15] Stefanie Tellex, Boris Katz, Jimmy J. Lin, Aaron Fernandes, and Gregory Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *SIGIR*, pages 41–47, 2003.