Discovery of Similarity Computations of Search Engines *

King-Lup Liu
School of CTI
DePaul University
Chicago, IL 60604
kliu@cs.depaul.edu

Weiyi Meng
Department of Computer Science
SUNY – Binghamton
Binghamton, NY 13902
meng@cs.binghamton.edu

Clement Yu
Department of EECS
University of Illinois at Chicago
Chicago, IL 60607
yu@eecs.uic.edu

ABSTRACT

Two typical situations in which it is of practical interest to determine the similarities of text documents to a query due to a search engine are: (1) a global search engine, constructed on top of a group of local search engines, wishes to retrieve the set of local documents globally most similar to a given query; and (2) an organization wants to compare the retrieval performance of search engines. The dot-product function is a widely used similarity function. For a search engine using such a function, we can determine its similarity computations if how the search engine sets the weights of terms is known, which is usually not the case. In this paper, techniques are presented to discover certain mathematical expressions of these formulas and the values of embedded constants when the dot-product similarity function is used. Preliminary results from experiments on the WebCrawler search engine are given to illustrate our techniques.[1]

Categories and Subject Descriptors

H.3 [**Information Systems**]: Information storage and Retrieval; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition, parameter learning*

Keywords

Search engine, metasearch engine, similarity function, discovery

1. INTRODUCTION

Documents of interest are often found in many different sources. For example, a physicist may be interested in the latest technical reports on a particular topic from the databases of physics research laboratories across the country. There are various approaches to facilitate the retrieval of information scattered in distributed sources [4, 8, 13, 14]. An approach is to construct a global search engine, also known as metabroker or metasearch engine, on top of a group of local search engines (databases). However, the local search engines underlying a global search engine are usually autonomous. With respect to a given query Q, the similarity of a document assigned by a local search engine is likely to be different from that of the same document determined by the global search engine. Documents considered to be most similar to Q by the global search engine may have low similarities in the local search engine and hence are not retrieved. That is, the user may not obtain the documents most similar to his/her query. This problem of retrieving (globally) most similar documents from different local sources has received much interest recently [1, 4, 9, 13, 14, 16, 20, 21, 22]. Some approaches to this problem require that the global search engine know the computation of document similarities to a query by the underlying local search engines [9, 14].

After the global search engine has collected the search results from the local search engines, it must decide how to combine them into a single ranking. This is often known as the *collection fusion problem* [20, 21, 22]. It is a difficult problem as local document similarities from different local search engines may not be comparable. In [15], it is shown that knowing the computation of document similarities by the local search engines may help the merging of search results.

Another reason why it is of interest to discover the computation of document similarities of search engines is as follows. Suppose an organization has a collection of documents, a set of queries and an assessment of the relevance of the documents to the queries. It is interested in obtaining a benchmark of several search engines from the Web on the collection of documents based on the relevance assessment, but the collection is not available in the Web. (One such collection is the TREC collection of documents collected by the United States National Institute of Standards and Technology (NIST).) If the organization can find out the similarities of documents to a given query as computed by a search engine, it can determine the performance of the search engine on the collection with respect to the set of queries, and hence, it is able to compare the retrieval effectiveness of the

*This research is supported by the following organizations: NSF grants under CCR-9816633, CCR-9803974, CDA-9711582 and HRD-9707076, NASA under NAGW-4080 and NAG5-5095, and ARO under BMDO grant DAAH04-96-1-0278 and NAAH04-96-1-0049.

[1]A preliminary version of this paper appears as a poster paper in ACM DL'99

search engines on the collection of documents.

A function, known as the *dot-product* function, computes the similarity of a document to a query as the sum of products of the corresponding term weights in the query and the document. Although the number of possible similarity functions is numerous, many commonly used functions, including the well-known *Cosine* function [18], are simply the dot-product function with the weights of terms computed in a specific manner. In this paper, the techniques presented are based on the assumption that the dot-product similarity function is used. With this assumption, the similarity of a document to a query computed by a local search engine can be determined if we can discover how the search engine determines the weights of terms in a document and in a query. In text retrieval, there are numerous ways of setting the weight of a term [5, 6, 11, 18, 23]. In an Intranet environment, the formulas to compute term weights are probably known to the organization. In the Internet environment, the weight of each term used by a retrieval function in one search engine may be different from that in another search engine and can be unknown, as the information is proprietory. In such a situation, we need some method to determine how weights are assigned to terms by the search engine. Our contribution in this paper consists of

- providing techniques to find out the form of certain term-weight formulas;
- providing techniques to determine the constants embedded in the formulas for computing the weight of a term; and
- providing experimental results to illustrate how our techniques can be utilized to "discover" the similarity computation of the WebCrawler search engine.

In a recent paper [3], the problem of discovering the *language model* for a text database is addressed. A language model describes the words or indexing terms that occur in the database and frequency information indicating how often each term occurs [3]. A method is presented that uses a query-based sampling approach. It is shown that a database selection service (global search engine) can learn the language model of a (uncooperative) database by sampling the contents of the database via the process of running carefully selected queries and retrieving documents. In contrast, the problem we are addressing in this paper is to determine how a (uncooperative) search engine (database) assigns similarities to documents with respect to a query. The techniques we developed for discovering term-weight formulas are also query-based.

The rest of this paper is structured as follows. Section 2 describes the various factors that affect the weight of a term. Section 3 discusses how to determine the constants embedded in the formulas for computing the weight of a term. We assume that the system provides only the rank information of those documents retrieved, i.e., the similarity values of the retrieved documents are not given. Section 4 shows how the mathematical expressions for computing the different term weight components may be derived from the results of probe queries. Experimental results are presented in section 5. We applied our techniques to the well-known WebCrawler news search engine to find out how weights of terms are assigned in queries and in documents. Some preliminary data are supplied to confirm that our discovered weights are reasonably accurate. We obtained on the average 85% accuracy. For the two applications of our techniques, 100% accuracy is not required. Concluding remarks are given in section 6.

2. TERM-WEIGHT FORMULAS

Although the weight of a term can be computed in a variety of ways, it is generally influenced by the following three factors:

Term frequency: The term frequency of a term in a document/query is the number of occurrences of the term in the document/query. The component due to the term frequency is often determined by some expression which involves some constants whose values can be tailored for a particular seach engine. We shall call this term frequency component the **term frequency weight** of the term.

Document frequency: The document frequency of a term is the number of documents indexed by a search engine. The *inverse document frequency (idf)* method assigns a weight, called the *inverse document frequency weight*, to a term t that is usually defined as $\log \frac{N}{df_t}$, where N is the number of documents indexed by the search engine and df_t is the document frequency of term t. However, variations of this *idf* definition are not uncommon and frequently involve some adjustable constants.

Document length normalization: A longer document tends to have more terms and/or terms with higher term frequencies. This means that document retrieval would be biased in favor of long documents. To compensate for the document length, term weights are frequently normalized or divided by a factor involving the document length. Although less common, this document length normalization factor may also involve certain constants.

Note that the query term weights may also be computed using a query length normalization factor. However, as the query length normalization factor is contained in each query term weight, it can be factored out. Thus, with respect to a query, the relative ranking of any two documents is unchanged whether the query term weights contain the query length normalization factor or not. We shall not discuss the discovery of constants embedded in the query length normalization factor.

Suppose that the term frequency weight of a term t in a query Q contains the unknown constants $k_1, \ldots, k_p$; the term frequency weight of a term t in a document $\vec{d}$ has the unknown constants $a_1, \ldots, a_r$; the *idf* weight of term t involves the unknown constants $b_1, \ldots, b_s$; and the document length normalization factor makes use of the unknown constants $c_1, \ldots, c_h$. We shall denote the term frequency weight of a term t in query Q by $qtf_t(Q, K)$, the term frequency weight of t in $\vec{d}$ by $dtf_t(\vec{d}, A)$, the *idf* weight of t by $idf_t(B)$ and the document length normalization factor by $norm(\vec{d}, C)$, where $K = \{k_1, \ldots, k_p\}$, $A = \{a_1, \ldots, a_r\}$, $B = \{b_1, \ldots, b_s\}$ and $C = \{c_1, \ldots, c_h\}$. Below, we give some examples of term frequency weight formulas. More examples of formulas of the term weight components can be found in [7, 10, 19]

Example 1. (i) Term frequency weight of a term t in a query Q, $qtf_t(Q,K)$

1. $k_1 + k_2 \times tf_t(Q)$ $\quad (k_2 > 0)$
2. $(tf_t(Q))^{k_1}$ $\quad (k_1 > 0)$
3. $k_1 + \log(tf_t(Q) + k_2)$

where $tf_t(Q)$ is the term frequency of term t in Q, and k_1 and k_2 are constants.

We were not aware of systems using formula 2. In our experiment, we actually discover the use of formula 2 in WebCrawler.

(ii) Term frequency weight for a term in a document $\vec{d}$, $dtf_t(\vec{d}, A)$:

1. $a_1 + a_2 \times \dfrac{tf_t(\vec{d})}{max\_tf(\vec{d})}$ [17]
2. $a_1 + a_2 \times \dfrac{tf_t(\vec{d})}{tf_t(\vec{d}) + a_3 + a_4 \times \frac{dl(\vec{d})}{avg\_dl}}$ [2]
3. $a_1 + a_2 \times \dfrac{a_3 + \log tf_t(\vec{d})}{a_4 + \log max\_tf(\vec{d})}$ [19]
4. $a_1 + [tf_t(\vec{d})]^{a_2}$

where $tf_t(\vec{d})$ is the term frequency of a term t in a document $\vec{d}$, $max\_tf(\vec{d})$ is the maximum frequency of all terms in $\vec{d}$, $dl(\vec{d})$ is the total number of occurrences of of all terms in document $\vec{d}$, avg_dl is the average number of terms in a document of the database, each a_i is a constant $(i = 1, 2, 3, 4)$ and $a_2 > 0$. ■

We assume that the expressions for $qtf_t(Q,K)$, $dtf_t(\vec{d}, A)$ and $idf_t(B)$ have the following property.

Property 1. (Monotonicity)

(i) *For a given query Q, $qtf_t(Q,K)$ is a strictly increasing function of $tf_t(Q)$, the term frequency of term t in Q (i.e., if t_1 and t_2 are two terms such that $tf_{t_1}(Q) < tf_{t_2}(Q)$, then $qtf_{t_1}(Q,K) < qtf_{t_2}(Q,K)$).*

(ii) *For a given document $\vec{d}$, $dtf_t(\vec{d}, A)$ is a strictly increasing function of $tf_t(\vec{d})$, the term frequency of t in $\vec{d}$.*

(iii) *$idf_t(B)$ is a strictly decreasing function of df_t, the document frequency of term t.*

The above monotonicity property is reasonable and is satisfied by almost all $qtf_t(Q,K)$, $dtf_t(\vec{d}, A)$ and $idf_t(B)$ expressions. It simply says that if the term frequency/document frequency of a term t is strictly higher than that of another term, the term frequency weight/*idf* weight of t must be higher/lower. It can be easily verified that all the examples of term frequency weights given in example 1 also satisfy the monotonicity property.

Another reasonable assumption we make is the following query term independence property.

Property 2. (Query term independence) *For a given query Q, two query terms in Q have the same query term frequency weight if they have the same term frequency in Q and the query term frequency weight $qtf_t(Q,K)$ of a term t in Q is independent of the term frequencies of other query terms present in Q.*

Property 2 essentially says that for a term t in Q, $qtf_t(Q,K)$ depends on its term frequency $tf_t(Q)$ and not on the term t itself, and its value is unaffected by the occurrences of other terms in Q. It can be easily checked that all examples of query term frequency weight given in Example 1(i) have the above property.

3. DETERMINATION OF UNKNOWN CONSTANTS IN TERM WEIGHT FORMULAS

As indicated in the previous section, the formulas for the term weight components, $qtf_t(Q,K)$, $dtf_t(\vec{d}, A)$, $idf_t(B)$ and $norm(\vec{d}, C)$, may contain adjustable constants. In this section, we discuss how to find out these constants once we know the mathematical expressions of these formulas (but not the embedded constants).

After a user has submitted a query Q to a search engine, the retrieved documents are usually ranked in non-increasing order of their similarity values to Q before they are presented to the user. If the search engine also provides the similarities of retrieved documents, we can readily form equations involving the unknown constants. When a sufficient number of such equations are obtained, these unknown constants can be solved (either analytically or using numerical methods). However, the document similarities may not be available or they are transformed from the dot-product similarities by some unknown function. In this case, we do not know precisely how the similarities of different documents to a query or the similarities of the same document to different queries are related. Forming equations involving the embedded unknown constants (in K, A, B and C) becomes a non-trivial task. We describe how this can be done in the following subsection when *we know only the rank order of retrieved documents.*

We assume that the document frequencies of the terms in a search engine are obtainable. Note that many search engines on the Internet provide the document frequency information to the user. Alta Vista and Hotbot, for example, give the document frequency of any term t in the retrieval result in the form of the number of hits if term t is used as a single-term query.

Setting up Equations in the Unknown Constants

As discussed earlier, if the similarities of retrieved documents are not provided, it is difficult to form equations in the constants embedded in the term weight formulas from only the rank order of the documents. We show how this can be overcome in this subsection.

A key step in our procedure to discover the embedded constants is to construct a query Q such that the similarity of Q with two documents are (or approximately) the same. This will allow us to form an equation relating their similarities. We now sketch this key step.

Let Q_t be a single-term query containing a *single* occurrence of term t and Q_{t_1,t_2} a two-term query involving terms t_1 and t_2 which may have *multiple* occurrences in the query. The similarity of a document $\vec{d}$ to a query Q is denoted by $sim(Q, \vec{d})$.

The idea of our approach is this. First, we find two terms t_1 and t_2 and two documents $\vec{d_1}$ and $\vec{d_2}$ such that document $\vec{d_1}$ contains t_1 but not t_2 and document $\vec{d_2}$ contains t_2 but not t_1. As $\vec{d_1}$ has t_1 and $\vec{d_2}$ does not, with respect to the single-term query Q_{t_1}, the rank of $\vec{d_1}$ must be higher than that of $\vec{d_2}$. That is, $sim(Q_{t_1}, \vec{d_1}) \geq sim(Q_{t_1}, \vec{d_2})$. Similarly, with respect to query Q_{t_2}, the rank of $\vec{d_2}$ is higher, or $sim(Q_{t_2}, \vec{d_1}) \leq sim(Q_{t_2}, \vec{d_2})$. Now, consider the two-term query Q_{t_1,t_2}. If we increase (or decrease) the term frequency of t_1 in Q_{t_1,t_2}, by Property 1(i) (monotonicity property), the weight of query term t_1 will increase (or decrease) and, as a result, the similarity of document $\vec{d_1}$ to Q_{t_1,t_2}, $sim(Q_{t_1,t_2}, \vec{d_1})$, will increase (or decrease) (or the rank of $\vec{d_1}$ relative to $\vec{d_2}$ will increase (or decrease)). Similar remark also applies to term t_2. Thus, by suitably setting the term frequencies of t_1 and t_2 in query Q_{t_1,t_2}, we can make the document similarities $sim(Q_{t_1,t_2}, \vec{d_1})$ and $sim(Q_{t_1,t_2}, \vec{d_2})$ equal or approximately the same. That is, we can then form the equation

$$sim(Q_{t_1,t_2}, \vec{d_1}) = sim(Q_{t_1,t_2}, \vec{d_2})$$

We have developed an algorithm that would adjust the term frequencies of the query terms t_1 and t_2 so that the document similarities $sim(Q_{t_1,t_2}, \vec{d_1})$ and $sim(Q_{t_1,t_2}, \vec{d_2})$ are as close as possible. The idea of the algorithm is similar to binary search. The algorithm uses two variables U and L. U keeps the ratio $u_1 : u_2$ of the term frequencies of t_1 and t_2 in query Q_{t_1,t_2} with respect to which document $\vec{d_1}$ has a *higher* rank than $\vec{d_2}$ does. L stores a similar ratio $v_1 : v_2$ except that with respect to Q_{t_1,t_2}, document $\vec{d_1}$ has a *lower* rank. Thus, for the pair of terms t_1 and t_2 and pair of documents $\vec{d_1}$ and $\vec{d_2}$, described in the previous paragraph, initially, $U = 1 : 0$ and $L = 0 : 1$. We then form the ratio $R = (u_1 + v_1) : (u_2 + v_2) = w_1 : w_2$ and eliminate any common factors. Next, we construct the query $Q^*_{t_1,t_2}$ with w_j occurrences of t_j (j = 1, 2). If with respect to $Q^*_{t_1,t_2}$, $\vec{d_1}$ has a higher rank, we replace the ratio in U by R, otherwise the ratio in L is replaced by R. This process is repeated until the difference between U and R is less than some pre-determined positive number. The details of the algorithm is given in [12].

Using the above steps, we can form as many equations involving the embedded constants as we want. When enough equations are constructed, it is possible to solve for the sets of constants K, A, B and C. We have developed a systematic procedure which facilitates the solution of the unknown constants [12].

4. DERIVATION OF TERM WEIGHT FORMULAS

The possible formulas for determining the term weight components are theoretically infinite. However, from the retrieval results of probe queries, we can often derive approximately (sometimes precisely) for a term weight component its mathematical formula or rule out certain groups of mathematical expressions the formula can take. To illustrate the idea of our techniques, we show how this can be done for the *term frequency weight* formulas.

Throughout this section, a query having f_j instances of term t_j, $j = 1, \ldots, n$, will be denoted by $Q_{t_1,\ldots,t_n}(f_1, \ldots, f_n)$. $Q_{t_1,t_2}(v_1, v_2)$, for example, denotes a query with term frequency of t_j equal to v_j $(j = 1, 2)$ (i.e., $tf_{t_j}(Q_{t_1,t_2}(v_1, v_2)) = v_j$, $j = 1, 2$).

Query Term Frequency Weight Formulas

A simple test we can perform is :

- determine whether the ranking of the retrieved documents changes for a query (with more than one term) if the term frequency of each term in the query changes by a constant multiple.

That is, we want to check if the retrieval results of the queries $Q_{t_1,\ldots,t_n}(f_1, \ldots, f_n)$ and $Q_{t_1,\ldots,t_n}(c \times f_1, \ldots, c \times f_n)$ are identical for any positive constant c. Suppose the same set of documents is retrieved and the ranking of these documents does not change for these queries. Then, under the query term independence assumption (Property 2), the formula derived for the term frequency weight of a term t in a query Q, $qtf_t(Q, K)$, is $[tf_t(Q)]^k$ for some constant k (has the form shown in Example 1(i).2) ($tf_t(Q)$ being the term frequency of t in Q). The details of the derivation is given in [12].

By studying the ranking of documents as we vary the term frequencies of terms in a query, we can determine whether the query term frequency weight formula, $qtf_t(Q, K)$, has the form $k_1 + k_2 \times tf_t(Q)$ (Example 1(i).1) or $k_1 + k_2 \times \log(tf_t(Q) + k_3)$ (Example 1(i).3) for some constants k_1, k_2 and k_3. For our investigation, we do the following

1. find three terms t_1, t_2 and t_3 with t_2 and t_3 having equal document frequency;
2. find two documents $\vec{d_1}$ and $\vec{d_2}$ such that
   (a) the term frequencies of t_2 and t_3 in $\vec{d_i}$ are the same (i.e., $i = 1, 2$); and

(b) the rank of document $\vec{d_1}$ is higher than that of $\vec{d_2}$ in the retrieval result for query Q_{t_1} but is lower in the retrieval result for Q_{t_2,t_3};

3. fix the term frequency of t_1 in the three term query $Q_{t_1,t_2,t_3}(\mu, y, z)$ to some constant μ; and

4. find pairs of query term frequencies y_j and z_j $(j = 1, \ldots, m)$ for t_2 and t_3 respectively such that with respect to each of the queries $Q_{t_1,t_2,t_3}(\mu, y_j, z_j)$, the similarities of documents $\vec{d_1}$ and $\vec{d_2}$ are the same (approximately).

In step 4, each pair (y_j, z_j) can be obtained as follows. We set the term frequency of t_3 to the value z_j and adjust y, the term frequency of t_2 in $Q_{t_1,t_2,t_3}(\mu, y, z_j)$, to y_j using a method similar to binary search (in a similar manner as that described in the previous section).

The query term frequency weight of a term t in a query Q depends on $tf_t(Q)$, the term freqency of t in the query. In the following, we denote $qtf_t(Q, K)$, the formula for the query term frequency weight of a query term t, by $H(v)$, where $v = tf_t(Q)$.

As the document frequencies of t_2 and t_3 are equal (step 1), and the term frequencies $tf_{t_2}(\vec{d_i})$ and $tf_{t_3}(\vec{d_i})$ are the same $(i = 1, 2)$ (step 2), the following set of equalities can be established $(j = 1, \ldots, m)$

$$\alpha + [H(y_j) + H(z_j)] = 0 \qquad (1)$$

where α is a constant.

Valuable information can be derived about the formula of the query term frequency weight, $H(v)$, by examining the pairs of values y_j and z_j $(j = 1, \ldots, m)$. If the sums $y_j + z_j$ $(j = 1, \ldots, m)$ are the same (approximately), we can infer that the formula of the term frequency weight of a term in a query, $H(v)$, is a linear function of v, the term frequency of the term in the query [12]. (Using our original notation, this is equivalent to saying that the formula $qtf_t(Q, K) = k_1 + k_2 \times tf_t(Q)$ for some constants k_1 and k_2 i.e., has the same form as that shown in Example 1(i).1.) We may study the pairs of values $(y_j + z_j)$ and $y_j z_j$ $(j = 1, \ldots, m)$. If these values indicate a linear relationship between the sum $y_j + z_j$ and the product $y_j z_j$, then we can conclude that the query term frequency weight formula, $H(v)$, is a linear function of $\log(v + k)$ (see Example 1(i).3), where k is some constant [12]. In general, if $f(y_j) + f(z_j)$ is constant for some function $f()$, then $H(v)$ is a linear function of $f(v)$. To verify our conclusion, we may repeat the steps described above to obtain another set of pairs of data using a different pair of documents and a different set of terms.

Document Term Frequency Weight Formulas

Note that the complexity of our computation of the embedded constants will be greatly reduced if the term frequency weight of a term t in a document $\vec{d}$, $dtf_t(\vec{d}, A)$, is 0 whenever the term frequency of t in $\vec{d}$, $tf_t(\vec{d})$, is 0. The following experiment is designed to check whether the term frequency weight formula has such property.

Let t_1, t_2 and t_3 be three terms. Let $\vec{d_1}$ and $\vec{d_2}$ be two documents such that (i) the rank of $\vec{d_1}$ is higher than that of $\vec{d_2}$ for the single-term query Q_{t_1} but is lower for query Q_{t_2} and (ii) both documents do not contain t_3 (i.e. $tf_{t_3}(\vec{d_1}) = tf_{t_3}(\vec{d_2}) = 0$). Then,

1. determine the term frequencies of the query terms in Q_{t_1,t_2}, with respect to which the documents $\vec{d_1}$ and $\vec{d_2}$ have the same similarities (approximately); and

2. fix the number of occurrences of t_3 to some positive number in the query Q_{t_1,t_2,t_3} and determine the term frequencies of the query terms t_1 and t_2 in Q_{t_1,t_2,t_3}, with respect to which the documents $\vec{d_1}$ and $\vec{d_2}$ have the same similarities (approximately).

If the query term freqency weights of t_1 and t_2 in Q_{t_1,t_2} obtained in step (1) are the same as the corresponding weights in Q_{t_1,t_2,t_3} obtained in step (2), then it can be readily shown that the formula for the document term frequency weight has the above-mentioned property (i.e. if $tf_t(\vec{d}) = 0$, then $dtf_t(\vec{d}, A) = 0$) provided that the document length normalization factors of the two documents are not equal (which is highly likely). To verify our result, we can repeat the above steps using different sets of documents and/or different sets of terms.

It is possible that a non-zero term frequency weight is assigned to a term in a document even though the document does not contain the term. However, this is very uncommon. In the following discussion, we shall assume that if a document does not have a term, the term frequency weight of the term in the document is zero.

To determine the formula for the term frequency weight of a term in a document, we perform the following steps.

1. find a collection of terms $t_2, \ldots, t_r$ each of which has the same document frequency;

2. find two documents $\vec{d_1}$ and $\vec{d_2}$ such that all $t_2, \ldots, t_r$ occur in $\vec{d_2}$ but none of these terms appears in $\vec{d_1}$;

3. find a term t_1 that appears in $\vec{d_1}$ but not in $\vec{d_2}$; and

4. for each pair of terms t_1 and t_j $(j = 2, \ldots, r)$, determine the query term frequencies of t_1 and t_j in the two-term query Q_{t_1,t_j} such that the similarities of $\vec{d_1}$ and $\vec{d_2}$ to Q_{t_1,t_j} are equal (approximately).

Because document $\vec{d_1}$ does not have t_j $(j = 2, \ldots, r)$ and document $\vec{d_2}$ does not have t_1, we have

$$sim(Q_{t_1,t_j}, \vec{d_1}) = \frac{qtf_{t_1}(Q_{t_1,t_j}, K) \times idf_{t_1}(B) \times dtf_{t_1}(\vec{d_1}, A)}{|Q_{t_1,t_j}| \times norm(\vec{d_1}, C)}$$

and

$$sim(Q_{t_1,t_j}, \vec{d_2}) = \frac{qtf_{t_j}(Q_{t_1,t_j}, K) \times idf_{t_j}(B) \times dtf_{t_j}(\vec{d_2}, A)}{|Q_{t_1,t_j}| \times norm(\vec{d_2}, C)}$$

As the two similarities shown above are the same (step 4), equating the two expressions on the left-hand side, we obtain

$$dtf_{t_j}(\vec{d_2}, A) = \lambda \times \frac{qtf_{t_1}(Q_{t_1,t_j}, K)}{qtf_{t_j}(Q_{t_1,t_j}, K)} \quad (2)$$

where $\lambda = \frac{norm(\vec{d_2},C)\times idf_{t_1}(B)\times dtf_{t_1}(\vec{d_1},A)}{norm(\vec{d_1},C)\times idf_{t_j}(B)}$, which is the same for all j. Note that as the document frequencies of $t_2, \ldots, t_r$ are the same (step 1), the *idf* weights $idf_{t_j}(B)$ $(j = 2, \ldots, r)$ are equal.

Suppose the formula for the query term frequency weight, $qtf_t(Q, K)$, has been determined. From the query term frequencies obtained in step 4, $\frac{qtf_{t_1}(Q_{t_1,t_j}, K)}{qtf_{t_j}(Q_{t_1,t_j}, K)}$ can be determined and we let x_j be the computed value. The term frequency weight of t_j in $\vec{d_2}$, $dtf_{t_j}(\vec{d_2}, A)$, depends on $tf_{t_j}(\vec{d_2})$, the term frequency of t_j in $\vec{d_2}$. Let u_j be the term frequency $tf_{t_j}(\vec{d_2})$. Then, $dtf_{t_j}(\vec{d_2}, A)$ is a function of u_j. We denote this function by $F(u_j)$. Using the notation just defined, from (2), we have $F(u_j) = \lambda \times x_j$ $(j = 2, \ldots, r)$. By studying the $(r-1)$ pairs of values (u_j and x_j), we can often determine the form of the mathematical expression of $F()$ (or the formula for the document term frequency weight). If there is a linear relationship among (u_j, x_j), then $F()$ is a linear function of the term freqency; that is, the expression for $dtf_t(\vec{d}, A)$ is a linear function of the term frequency $tf_t(\vec{d})$ (Example 1(ii).1 is such an example). More generally, if there is a linear relationship among $(\theta(u_j), x_j)$ for some known function $\theta()$, the expression for the formula $dtf_t(\vec{d}, A)$ is a linear function of $\theta(tf_t(\vec{d}))$. Example 1(ii).3 is an example in which $\theta()$ is the logarithm function. A mathematical formula of the form shown in Example 1(ii).4 can be determined by examining if there is a linear relationship among $(\log u_j, \log x_j)$ $(j = 2, \ldots, r)$.

5. EXPERIMENTAL RESULTS

To test the effectiveness of our techniques, we applied them to "discover" the similarity computation of the well-known WebCrawler search engine. We aimed at discovering how the WebCrawler news search engine assigns the weights of query terms and terms appearing in a news article page. News articles were used for our experiments as they are predominantly text. In comparison to other web pages, they are compositionally simpler. In the retrieval list for a query, WebCrawler also gives a relevance rating for each news article retrieved. As we do not know how it is computed, the relevance rating is not used in the term weight estimation. However, when our algorithm was used to set the numbers of occurrences of terms in a query so that two given documents had approximately the same similarities to the query, the relevance rating was used to gauge the closeness of the similarities of these two documents. If the relevance ratings for the two documents were the same, we concluded that their similarities to the query were sufficiently close, and not otherwise.

In the following, we describe how the formula used by WebCrawler to determine the query term frequency weight of a query term was discovered. We first submitted a collection of queries, each having the same query terms. For any two queries in the collection, the term frequencies of one are a constant multiple of the corresponding term frequencies of the other. (For example, if one of the query is {voters, turnout, turnout}, a query in the collection might be {voters, voters, turnout, turnout, turnout, turnout}.) We observed that the retrieval results for these queries were identical. The documents, their rank order and relevance ratings were the same for each query. We observed the same phenomenon for several similar collections of queries (with different query terms). Thus, we concluded that by property 2 (query term independence assumption), the formula for the query term frequency weight was of the form $(tf_t(Q))^k$ for some constant k ($tf_t(Q)$ being the term frequency of term t in query Q) (see the discussion in the previous section).

Having determined the mathematical expression of the formula, we performed the steps, described in the previous section, leading to the formation of (1), a set of equalities. However, in these steps, our primary concern was to find two particular pairs of term frequencies (y_1, z_1) and (y_2, z_2). The first pair had the same value for the two term frequencies (i.e., $y_1 = z_1$) and the second pair had value 0 for one of term frequencies, say z_2. Other pairs were also found, which were needed for verification purposes only. Thus, we obtained $\alpha + [H(y_1) + H(z_1)] = 0$ and $\alpha + [H(y_2) + H(0)] = 0$ (see (1)). It follows that $2 \times H(y_1) = H(y_2) + H(0)$. Note that $H(v)$ is the query term frequency weight for a query term with term frequency equal to v $(= tf_t(Q))$, which we determined to be v^k (i.e., $H(v) = v^k$). Hence, we have $2 \times y_1^k = y_2^k$, from which we computed the value of k (since both y_1 and y_2 were known). The value we obtained for k is 0.3625. That is, the formula for the query term frequency weight $qtf_t(Q)$ is determined to be $[tf_t(Q)]^{0.3625}$. We found that this formula is consistent with all the data we collected.

The formulas that we estimated for the other term weight components are as follows:

- *Document term frequency weight*

  $$dtf_t(\vec{d}, A) = (f_M + 3.5 \times f_T)^{0.175}$$

  where f_T is the number of occurrences of term t in the title and f_M is the number of occurrences of t in the main text of document $\vec{d}$.

- *Inverse document frequency (idf) weight*

  $$idf_t(B) = \log \frac{220000}{df_t}$$

  where df_t is the document frequency of term t.

- *Document length normalization factor* $norm(\vec{d}, C)$ is the Euclidean norm formula.

To test how accurate the estimated term weight formula approximates the actual term weight formula used by WebCrawler, we submitted a set of 50 queries to its news search engine. These 50 queries include 10 single-term queries, 15 two-term queries, 15 three-term queries, 5 four-term queries and 5 five-term queries. For each of these queries, more than 100 documents were retrieved by the news search engine of

Table 1: Sample queries and retrieval results

Query	10	20	30	40	50
q1	8	16	25	32	42
q2	7	18	26	33	41
q3	9	16	24	34	43
q4	9	17	26	35	45
q5	8	17	27	33	42

q1 : tenants
q2 : voter turnout (2)
q3 : retreat peril (5) impeachment (2)
q4 : oath inauguration (3) celebration president
q5 : city (2) business season economy (3) inflation (2)

Table 2: Experimental results

nQt	10	20	30	40	50
1	85	83.5	84.6	85	84.4
2	83.3	83.7	85.33	84	84.3
3	85.3	84.6	85.8	84.7	84
4	84	83	85.3	85	86
5	86	85	86.7	86	85.2

WebCrawler; we downloaded the first 100 documents, computed their similarities using the term-weight formulas we obtained from our experiments and ranked these 100 documents in non-decreasing order of their similarities. As a user is usually interested in no more than the top 50 documents, for each query, we compared the first 50 documents retrieved by the news search engine and the first 50 documents according to the estimated document similarities. Table 1 shows the retrieval results obtained for five of the 50 submitted queries. In the row for query q2, for example, the number 26 in the fourth entry (under column 30) means that among the top 30 documents ranked according to the similarities computed using the estimated term weights, 26 of these were found among the top 30 documents returned by WebCrawler for q2, which consists of one occurrence of the term "voter" and two occurrences of the term "turnout". All other numbers in the table (except those in the first row) are interpreted similarly. In the annotation for the queries, a number, enclosed in brackets, following a term indicates the number of occurrences of the term in the query.

We find that on average, the top 10 documents predicted by our computations included 84.6% of the top 10 documents retrieved by the WebCrawler news search engine. The corresponding percentages for the top 20, 30, 40 and 50 documents predicted by us are 84, 85.5, 84.8 and 84.5 respectively. In Table 2 are more detailed results showing the average percentages for queries with different numbers of query terms. In the third row, for example, the number 2 under column nQt means that the average percentages in this row were computed for queries having 2 distinct query terms; the number 83.3 in the second entry (under column 10) indicates that for queries with 2 query terms, the top 10 documents according to our computation included on the average 83.3% of the top 10 documents returned by WebCrawler.

100% accuracy is not attained by our prediction. This is because there are other factors we did not take into account such as pictures and hyperlinks in the web pages. Also, the Webcrawler news search engine does not accept queries with more than 50 words. This also affects the accuracy of the equations we formed.

6. CONCLUDING REMARKS

In this paper, we provided techniques that can be used to find out the form of certain term-weight formulas used by a search engine. We also showed how to determine the unknown constants embedded in term-weight formulas making use of only the ranking order of the retrieved documents. These techniques and strategies were developed under the assumption that the well-known dot-product similarity function is used. To determine the feasibility of our strategies, we chose the news search engine of WebCrawler for our experiments. The formulas that are found to be consistent with our experimental data are (i) $m^{0.3625}$ for the term frequency weight of a term t appearing m times in a query Q, (ii) $(f_M + 3.5 \times f_T)^{0.175}$ for the term frequency weight of a term t in a document that has f_M occurrences of t in its main text and f_T occurrences of t in its title, (iii) $\log \frac{220000}{df_t}$ for the inverse document frequency (idf) weight of a term t in the search engine and (iv) the Euclidean norm of term weights for the document length normalization factor.

To test the accuracy of the formulas determined, we submitted 50 queries to the WebCrawler news search engine. For each of these queries, we downloaded the first 100 news documents and computed their similarities using the estimated term-weight formulas. We found that on average, the top 10 documents predicted by us included 84.6% of the top 10 documents retrieved by the WebCrawler news search engine. The corresponding percentages for the top 20, 30, 40 and 50 documents were 84, 85.5, 84.8 and 84.5 respectively.

7. ADDITIONAL AUTHORS

Additional authors: Chonghua Zhang (School of CTI, DePaul University, email: `czhang@cs.depaul.edu`) and Naphtali Rishe (School of Computer Science, Florida International University, email: `rishen@fiu.edu`).

8. REFERENCES

[1] C. Baumgarten. A probabilistic model for distributed information retrieval. In *Proceedings of ACM SIGIR Conference*, 1997.

[2] J. Broglio, J. Callan, W. Croft, and D. Nachbar. Document retrieval and routing using the inquery system. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pages 29–38. NIST Special Publication 500-225, April 1994.

[3] J. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proceedings of ACM SIGMOD*, 1999.

[4] J. Callan, Z. Lu, and W. Croft. Searching distributed collections with inference networks. In *Proceedings of ACM SIGIR Conference*, 1995.

[5] W. Croft and D. Harper. Using probabilistic models of information retrieval without relevance information. *Journal of Documentation*, 35:285–295, 1979.

[6] E. Fox. Extending the boolean and vector space models of information retrieval with p-norm queries and multiple concept types. Cornell University, August 1983.

[7] W. Frakes and B.-Y. R., editors. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, 1992.

[8] L. Gravano and H. Garcia-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *VLDB*, 1995.

[9] L. Gravano and H. Garcia-Molina. Merging ranks from heterogenous internet sources. In *VLDB*, 1997.

[10] D. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Kluwer Academic Publishers, 1998.

[11] K. Kwok. A network approach to probabilistic information retrieval. *ACM Transactions on Information Systems*, 13:325–354, 1995.

[12] K. Liu, W. Meng, C. Yu, and N. Rishe. Discovery of similarity computations of search engines. Technical report, Depaul University, 2000. (http://www.depaul.edu/∼kliu/disc_sim.ps)

[13] K. Liu, C. Yu, W. Meng, W. Wu, and N. Rishe. A statistical method for estimating the usefulness of text databases. *IEEE Transactions on Knowledge and Data Engineering*. (to appear).

[14] W. Meng, K. Liu, C. Yu, X. Wang, Y. Chang, and N. Rishe. Determining text databases to search in the internet. In *VLDB*, 1998.

[15] W. Meng, C. Yu, and K. Liu. Detection of heterogeneities in a multiple text database environment. In *CoopIS*, 1999.

[16] A. Moffat and J. Zobel. Information retrieval for large document collections. In *Proceedings of the Third Text REtrieval Conference TREC-3*, 1994.

[17] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[18] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McCraw-Hill, New York, 1983.

[19] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of ACM SIGIR Conference*, 1996.

[20] G. Towell, E. Voorhees, N. Gupta, and B. Johnson-Laird. Learning collection fusion strategies for information retrieval. In *12th Int'l Conf. on Machine Learning*, 1995.

[21] E. Voorhees, N. Gupta, and B. Johnson-Laird. The collection fusion problem. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, 1994.

[22] E. Voorhees, N. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proceedings of ACM SIGIR Conference*, 1995.

[23] S. Wong and Y. Yao. On modeling information retrieval with probabilistic inference. *ACM Transactions on Information Systems*, 13:38–68, 1995.