A Method for Calculating Term Similarity on Large Document Collections

Wolfgang W. Bein *
School of Computer Science
University of Nevada
Las Vegas, NV 89154-4019
bein@cs.unlv.edu

Jeffrey S. Coombs and Kazem Taghva
Information Science Research Institute
University of Nevada
Las Vegas, NV 89154-4021
{jscoombs, taghva}@isri.unlv.edu

Abstract

We present an efficient algorithm called the Quadtree Heuristic for identifying a list of similar terms for each unique term in a large document collection. Term similarity is defined using the Expected Mutual Information Measure (EMIM). Since our aim for defining the similarity lists is to improve information retrieval (IR), we present the outcome of an experiment comparing the performance of an IR engine designed to use the similarity lists. Two methods were used to generate similarity lists: a brute-force technique and the Quadtree Heuristic. The performance of the list generated by the Quadtree Heuristic was commensurate with the brute force list.

1 Background

To facilitate the retrieval of OCR documents, the Information Science Research Institute has begun construction of a retrieval system called Hairetes [7]. Hairetes enhances a traditional retrieval system by incorporating the technique of Retrieval by General Logical Imaging (RbGLI) as developed by Crestani and Van Rijsbergen [1, 2, 4].

The basic idea of RbGLI is to retrieve not only those documents which contain terms in a query, but also documents which contain terms that are *similar* to those in the query but do not contain the query terms themselves. In standard retrieval systems only terms that actually appear in a query can be used to determine which documents should be retrieved. Thus, if we enter a query such as "Find all documents that discuss magnetic field intensity tests from faults at Yucca Mountain," only documents which contain at least one occurrence of one of the query terms could be retrieved. The aim of RbGLI is to expand the query so that documents which do *not* contain any of the query terms would still receive some weight if they contain terms which are similar to query terms.

For OCR texts in particular, we hope that documents which contain terms that are misrecognized by the OCR process might still be retrieved if the "correct" counterpart to a misrecognized term is found to be similar to a misrecognized form. Since a user is likely to enter the correct counterpart, it would be useful if the retrieval would also pull up documents which contain the misrecognized term.

We have followed Crestani in selecting EMIM, the Expected Mutual Information Measure, as our measure of word similarity [2]. EMIM, we will see in Section 2, requires counting the co-occurrences of terms in a document. Since counting co-occurrences is computationally very expensive, we propose a heuristic to find EMIM-similar terms in a large document collection in Section 3. The results of experiments are reported in Section 4. These indicate that the algorithm performs well when compared to results obtained by brute-force computation.

2 The Expected Mutual Information Measure

We consider a collection of N documents which contain M keys, where we assume that documents and keys are indexed by natural numbers, *i.e.* the set of documents is $\mathcal{D} = \{1, \ldots, N\}$ and the set of keys is $\mathcal{K} = \{1, \ldots, M\}$. For each key $k \in \mathcal{K}$, we define the the key-document incidence vector, $t_k : \mathcal{D} \to \{0, 1\}$, where for $d \in \mathcal{D}$,

$$t_k(d) = \begin{cases} 1 & \text{if } k \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases} .$$

The table below shows an example of two key-document incidence vectors t_k and t_ℓ:

*The work of this author was supported by ISRI during summer 2002.

$\mathcal{D}$	...	t_k	t_ℓ	...
1	...	1	0	...
2	...	1	1	...
3	...	1	0	...
4	...	0	0	...

For term $k \in \mathcal{K}$, we define the quantities f_k^1 – the *occurrence* of k, and f_k^0 – the *non-occurrence* of k:

$$f_k^i = \begin{cases} |\{d \in \mathcal{D} \mid k \text{ occurs in } d\}| & \text{for } i = 1 \\ |\{d \in \mathcal{D} \mid k \text{ does not occur in } d\}| & \text{for } i = 0 \end{cases}.$$

Furthermore, for two terms $k, \ell \in \mathcal{K}$, we define *mixed co-occurrences* $n_{k,\ell}^{i,j}$, for $i, j \in \{0, 1\}$:

$$n_{k,\ell}^{i,j} = |\{d \in \mathcal{D} \mid \ t_k(d) = i \wedge t_\ell(d) = j\}|.$$

The value $n_{k,\ell}^{11}$ is the number of documents in which the two terms both appear and is called the *co-occurrence* of the two terms.

We define the *Expected Mutual Information Measure (EMIM)* as:

$$EMIM(k,\ell) = \sum_{i=0}^{1}\sum_{j=0}^{1} n_{k,\ell}^{i,j} \log_2\left(\frac{n_{k,\ell}^{i,j}}{f_k^i f_\ell^j}\right) \qquad (1)$$

(We note that in the definition of $EMIM$ (1), there sometimes is an extra factor $C = \frac{\log_2 N}{N}$ present; however within one data set C is a constant, see also van Rijsbergen [3].)

Both M and N are large numbers, in practice we might expect $M, N > 10^6$; calculating $EMIM$ efficiently is therefore non-trivial. We can assume that for key $k \in \mathcal{K}$, the quantity f_k^1 is known. Then, from Table 1 it is clear, that to calculate $EMIM(k,\ell)$ for $k, \ell \in \mathcal{K}$, only the co-occurrence $n_{k,\ell}^{11}$ is needed, as all other values can be calculated by simple additions once $n_{k,\ell}^{11}$ is known. Of course, determining $n_{k,\ell}^{11}$ is a non-trivial task for a large document collection and represents the primary difficulty for using EMIM as the similarity measure for generating similar term lists for each term in the inverted file of a typical document retrieval system.

$n_{k,\ell}^{11}$	$n_{k,\ell}^{01}$	$\boxed{\sum = f_\ell^1}$
$n_{k,\ell}^{10}$	$n_{k,\ell}^{00}$	$\sum = f_\ell^0$
$\boxed{\sum = f_k^1}$	$\sum = f_k^0$	$\boxed{N}$

Table 1. The Contingency Table

It can be assumed that all documents have been preprocessed such that the following two queries can be performed in $O(\log M + \log N)$ time:

1. For any $k \in \mathcal{K}$, provide a pointer to a list of all distinct documents, which contain key k.
2. For any $d \in \mathcal{D}$, provide a pointer to a list of all distinct keys, which are contained in document d.

Such preprocessing can be accomplished by setting up two B-trees, one for each query type; this takes time $O((M+N)(\log M + \log N))$. For very large data collections such preprocessing is tedious, albeit not impossible (as compared with a brute-force $O((M+N)^2)$).

We are interested in preprocessing the data further to facilitate an efficient (logarithmic run-time) approximation of the following EMIM query:

- Given a constant $c \geq 1$, for a chosen $k \in \mathcal{K}$, provide pointers to keys $k_1, \ldots, k_c$ such that $k_1, \ldots, k_c \in \mathcal{K}$ are the keys with highest EMIM to k.

To apply RbGLI, we require a list of c similar terms for each of the M terms in a document collection, where c will tend to be a small number (c = 5 in experiments.) These c terms will be the "most" similar to a given term in the lexicon of our retrieval engine in the sense of having the highest EMIM relative to the given term. In the next Section we show a heuristic that gives such similar terms efficiently.

3 The Quadtree Heuristic

For our heuristic we apply the following preprocessing steps to the data collection:

1. Find a "representative" set of R reference keywords words $r_1, \ldots r_R$, where $R \in O(\log M)$.
2. For each key $k \in \mathcal{K}$, create the reference incidence vector $a(k) = (a_1(k), \ldots, a_R(k))$, where $a_i = EMIM(k, r_i)$, for $i = 1, \ldots, R$.
3. For each (i,j), $i, j = 1, \ldots, R, i \neq j$: Initialize an empty two-dimensional quadtree $T_{i,j}$ with granularity $\epsilon_0 > 0$. The parameter ϵ_0 is chosen in such a way that there are no more than c elements at the leaves the tree.
4. Select integer α, $0 < \alpha < \sqrt{\binom{R}{2}}$.
5. For each $k \in \mathcal{K}$, insert the α best pairs (best according to $a_i + a_j$ value) $(a_{i_1}(k), a_{j_1}(k))$, ..., $(a_{i_\alpha}(k), a_{j_\alpha}(k))$, into their respective quadtrees $T_{i_1,j_1}, \ldots T_{i_\alpha,j_\alpha}$.

We note that these steps can be carried out using the same $O((M+N)(\log M + \log N))$ time complexity as required by the usual preprocessing described in Section 2 for the B-trees setup.

The co-occurrence query is implemented as follows:

Recall	Average Precision	Average Interpolated Precision
0	55.43	59.09
10	41.50	44.01
20	37.96	38.91
30	34.36	35.06
40	31.39	31.87
50	26.84	27.34
60	24.45	24.55
70	21.22	21.55
80	18.48	18.50
90	15.05	15.24
100	9.20	9.20
Average:	**28.72**	**29.58**

	11-Point Precision	3-Point Precision
	28.716	27.760
Interpolated	29.576	28.248

Table 2. Average Precision and Recall (Top); Overall Averages for Brute Force (Bottom)

Recall	Average Precision	Average Interpolated Precision
0	55.55	59.05
10	41.51	43.95
20	37.97	38.86
30	34.29	35.00
40	31.44	31.93
50	26.86	27.35
60	24.42	24.53
70	21.17	21.50
80	18.49	18.52
90	15.05	15.25
100	9.19	9.19
Average:	**28.72**	**29.56**

	11-Point Precision	3-Point Precision
	28.723	27.777
Interpolated	29.557	28.243

Table 3. Average Precision and Recall (Top); Overall Averages for Quadtree Heuristic (Bottom)

1. For key k, find the two highest values in the reference incidence vector; let i_0 and j_0 be the corresponding indices.

2. In the quadtree T_{i_0,j_0}, find all keys $\tilde{\mathcal{K}}$ that are in the square of $a_{i_0}(k)$ and $a_{j_0}(k)$.

3. From $\tilde{\mathcal{K}}$ extract the c best keys. If fewer than c are found repeat with new pair i_0, j_0.

We note that the actual query step has $O(\log M + \log N))$ run-time for each key queried.

4 Experiments

To determine if the Quadtree Heuristic provides a reasonable list of similar terms, the following experiment was performed. We compared the lists of similar terms generated by a brute-force method with the Quadtree Heuristic in a retrieval engine designed to use such lists. If the two lists performed similarly, this gives some support to the idea that the faster Quadtree Heuristic is the better method for generating the lists of similar terms.

A vector-space retrieval system using the cosine measure as defined by Witten [11] was built using Berkeley DB data structures [6]. The document collection indexed consisted of 1055 OCR documents from the Licensing Support Network (LSN), a collection of OCR documents donated to the Information Science Research Institute by the Department of Energy for research purposes [10]. The total number of pages in the 1055 collection was 75,236, and the average number of pages was 71, see [8].

Similar term lists were generated in two ways. The first used a brute-force approach which compared each unique term in the lexicon of the retrieval engine with every other distinct term. EMIM was calculated for each pair, and for each term, the five terms with highest EMIM relative to it were selected for its similarity list. As this process was very time consuming, we computed EMIM only for terms occurring in three or more documents.

The second method used the Quadtree Heuristic (Section 3) to generate the similarity lists. We selected 100 reference keywords randomly from a list of terms which had a document frequency between 20 and 150. The idea was that the most effective reference keywords would fall in the class of "medium frequency" terms as these had been shown to be most useful for document retrieval [5].

In both cases the top five EMIM-similar terms were added to the retrieval system. A query on the retrieval system was processed as follows. Each term t in the query was stemmed using a Porter stemmer, and stop words were removed. For every document which contained a query term t, a weight was calculated using the cosine measure defined in Witten [11].

In addition to this standard way of calculating document

weights, the weights from terms in t's similarity list were factored in. Let s_1 through s_5 be the terms in t's similarity list. For every document d which contained one of the similar terms s_i but *not* t, a fraction of s_i's cosine weight was added to the document's weight. This fraction is called the "opinionated" factor by Crestani [1]. We calculated the "opinionated" factor of a similar term s_i as

$$s_i = \frac{c - (i-1)}{\sum_{j=1}^{c} j} \tag{2}$$

where c was the total number of terms in the similarity list, and i ($1 \leq i \leq c$) was the position of s_i in the list. Since we set $c = 5$ for our experiments, the similar terms s_1 through s_5 would respectively add $\frac{5}{15}$, $\frac{4}{15}$, $\frac{3}{15}$, $\frac{2}{15}$, and $\frac{1}{15}$ of the cosine weight to the documents in which they appeared.

Forty queries were submitted to the retrieval engine augmented by the similarity list. These queries had been developed by researchers familiar with the LSN document collection [8]. An example of one such query was "Find all documents that discuss magnetic field intensity tests from faults at Yucca Mountain." Five geologists then formulated relevancy judgments to determine which documents were most relevant to a given query.

The standard measures of retrieval effectiveness, *precision* and *recall*, were calculated based on these relevancy judgments [8]. Their percentages were averaged over the forty queries and the results presented in Tables 2 and 3. The first column of the top part in each Table lists the percentage of the recall value. The *recall value* is the number of relevant documents retrieved at a given point divided by the total number of relevant documents. Next to the recall values arrayed in the standard 11-point display appears the average percentage of the precision at the 11 recall points. *Precision* is the number of relevant documents retrieved divided by the total number of documents retrieved.

In the 11-point display, the precision represents the maximum precision achieved within a given recall range. For example, in the first row of the top part of Table 3, within the range of 0% $\leq$ recall $<$ 10%, 55.55% was the average percentage of the maximum precision achieved in this range. At recall point 40% in the same Table, on the average a maximum of 31.44% precision was achieved. At the recall point 100%, the precision at exactly the point when 100% recall was reached is given. *Interpolated precision* takes the maximum precision for a given interval and all higher recall levels [11].

Since there are eleven points from 0% to 100% in the recall column, the average of all eleven precision values is the 11-point average. This value is typically used to compare performances of retrieval systems. In addition, a 3-point value which averages the precision at recall points 20%, 50%, and 80% is another common measure. The 11-point and 3-point averages of precision in both its standard and interpolated form are provided for both experiments.

Comparing the overall averages listed in the bottom of Tables 2 and 3 show that the similarity lists created by the Quadtree Heuristic performed as well as those produced by the slower brute-force approach. The Quadtree Heuristic required only a few hours to generate its similarity list on a Sun *Blade 1000* machine while the brute-force required several days.

5 Conclusion

We note that in the formulation of the Quadtree Heuristic of Section 3 there are a number of choices depending on the specific document collection application. A good choice of representative reference keywords is crucial. In our experiments we used a choice based on "medium frequency"; however, choices based on the specifics of the collection are likely to improve precision. Other choices involve picking good values for α and ϵ_0.

The results obtained here are for a relatively small collection. However, we expect for our method to scale up favorably. Experiments are under way to apply the Quadtree Heuristic to a number of real-world collections available at ISRI [9].

References

[1] Fabio Crestani and C.J. Van Rijsbergen. A study of kinematics in information retrieval. *ACM Transactions on Information Systems*, 16:225–255, 1998.

[2] Fabio Crestani, Ian Ruthven, M. Sanderson, and C.J. van Rijsbergen. The troubles with using a logical model of ir on a large collection of documents. experimenting retrieval by logical imaging on TREC. In *Proceedings of the Fourth Text Retrieval Conference (TREC-4)*, 1995.

[3] C. J. Van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33(2):106–109, June 1977.

[4] C. J. Van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29:481–485, 1986.

[5] G. Salton, H. Wu., and C.T. Yu. The measurement of term importance in automatic indexing. *Journal of the American Society for Information Science*, pages 175–186, May 1981.

[6] Sleepycat Software. *Berkeley DB*. New Riders, 2001.

[7] Kazem Taghva and Jeffrey Coombs. Hairetes: A search engine for OCR documents. In *Proc. of 5th*

IAPR Intl. Workshop on Document Analysis Systems, Lecture Notes in Computer Science, pages 412–422, Princeton, NJ, August 2002. Springer-Verlag.

[8] Kazem Taghva, Thomas Nartker, Julie Borsack, and Allen Condit. Determining the usefulness of manually assigned keywords for a vector space system. In *Proc. of the IEEE Intl. Conf. on Information Technology: Coding and Computing*, pages 242–246, Las Vegas, NV, April 2002. IEEE Computer Society.

[9] Kazem Taghva, Tom Nartker, Julie Borsack, and Allen Condit. UNLV-ISRI document collection for research in OCR and information retrieval. In *Proc. IS&T/SPIE 2000 Intl. Symp. on Electronic Imaging Science and Technology*, San Jose, CA, January 2000.

[10] Kazem Taghva, Tom Nartker, Julie Borsack, Steve Lumos, Allen Condit, and Ron Young. Evaluating text categorization in the presence of OCR errors. In *Proc. IS&T/SPIE 2001 Intl. Symp. on Electronic Imaging Science and Technology*, pages 68–74, San Jose, CA, January 2001.

[11] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, 2nd edition, 1999.