HKU CSIS Tech Report TR-2002-09

An Efficient Algorithm for Incremental Update of Concept Spaces

Felix Cheung Ben Kao David Cheung C.Y. Ng

Department of Computer Science and Information Systems
The University of Hong Kong
{*kmcheung, kao, dcheung, cyng*}*@csis.hku.hk*

Abstract. The vocabulary problem in information retrieval arises because authors and indexers often use different terms for the same concept. A thesaurus defines mappings between different but related terms. It is widely used in modern information retrieval systems to solve the vocabulary problem. Chen et al. proposed the *concept space* approach to automatic thesaurus construction. A concept space contains the associations between every pair of terms. Previous research studies show that concept space is a useful tool for helping information searchers in revising their queries in order to get better results from information retrieval systems. The construction of a concept space, however, is very computationally intensive. In this paper, we propose and evaluate an efficient algorithm for the incremental update of concept spaces. In our model, only *strong* associations are maintained, since they are most useful in thesauri construction. Our algorithm uses a pruning technique to avoid computing *weak* associations to achieve efficiency.

Keywords: concept space, thesaurus, information retrieval, text mining

1 Introduction

The vocabulary problem has been studied for many years [6, 3]. It refers to the failure of a system caused by the variety of terms used by its users during human-system communication. Furnes et al. studied the tendency of people using different terms in describing a similar concept. For example, they discovered that for spontaneous word choice for concepts, in certain domain, the probability that two people choose the same term is less than 20% [6]. In an information retrieval system, if the keywords that a user specifies in his query are not used by the indexer, the retrieval fails.

To solve the vocabulary problem, a thesaurus is often used. A thesaurus contains a list of terms along with the relationships between them. During searching, a user can make use of the thesaurus to design the most appropriate search strategy. For example, if a search retrieves too few documents, a user can expand his query by consulting the thesaurus for similar terms. On the other hand, if a search retrieves too many documents, a user can use a more specific term suggested by the thesaurus. Manual construction of thesauri is a very complex process and often involves human experts. Previous research works have been done on automatic thesaurus construction [5].

In [7], Chen et al. proposed the concept space approach to automatic thesaurus generation. A concept space is a network of terms and their weighted associations. The association between two terms is a quantity between 0 and 1, computed from the co-occurrence of the terms from a given document collection. Its value represents the strength of similarity between the terms. If two terms never co-exist in a document, their associations are zero. When the association from a term a to another term b is close to 1, term a is highly related to term b in the document collection. Based on the idea of concept space, Schatz et al. constructed a prototype system to provide interactive term suggestion to searchers of the University of Illinois Digital Library Initiative test-bed [8]. Given a term, the system retrieves all the terms from a concept space that has non-zero associations to the given term. The associated terms are presented to the user in a list, sorted in decreasing order of association value. The user then selects new terms from the list to refine his queries interactively. Schatz showed that users could make use of the terms suggested by the concept space to improve the recall of their queries.

The construction of a concept space involves two phases: (1) an automatic indexing phase in which a document is processed to build inverted lists, and (2) a co-occurrence analysis phase in which the associations of every term pair are computed. Since there could be hundreds of thousands of terms in a document collection, a complete concept space that lists out the association values for all the term pairs is gigantic. For the purpose of thesaurus construction, fortunately, most of the association values are not used. Chen et al. suggested that for productive user-system interaction, only highly relevant concepts should be suggested to searchers [2]. For example, their worm thesaurus originally has 1,708,551 co-occurrence pairs and each term has a few thousand associated terms. They used 100 as the maximum number of related concepts for any term. If a term has more than 100 related terms, only the 100 terms with the highest association values are retained. They successfully removed about 60% of the less relevant co-occurrence pairs. In this paper, we call a concept space that only contains highly-ranked associations a *partial concept space*.

In a dynamic environment, such as the Web, the collection of documents on which a concept space is built changes with time. To capture the dynamics, a previously built (partial) concept space needs to be updated accordingly. The simplest approach to maintaining a concept space is to reconstruct it from scratch, using the updated set of documents. For large collections, unfortunately, such a brute force approach is too time-consuming. Our goal is to study the incremental update problem of partial concept spaces. We propose and evaluate an efficient pruning algorithm that achieves a significant speedup comparing with the brute-force method.

The rest of the paper is organized as follows. In Section 2 we give a formal definition of concept spaces and the incremental update problem. Section 3 briefly discusses the brute-force method. Section 4 discusses our pruning algorithm and its implementation details. Experiment results comparing the performance of the algorithms are shown in Section 5. Finally, Section 6 concludes the paper.

2 Definitions

A concept space contains the associations, W_{jk} and W_{kj}, between any two terms j and k found in a document collection. Chen and Lynch [7] define W_{jk} by the formula:

$$W_{jk} = \frac{\sum_{i=1}^{N} d_{ijk}}{\sum_{i=1}^{N} d_{ij}} \times \mathit{WeightingFactor}(k). \tag{1}$$

The symbol d_{ij} represents the weight of term j in document i based on the term-frequency-inverse-document-frequency (TFIDF) measure [1]:

$$d_{ij} = \mathit{tf}_{ij} \times \log(\frac{N}{\mathit{df}_j} \times w_j),$$

where

$$\begin{aligned} \mathit{tf}_{ij} &= \text{number of occurrences of term } j \text{ in document } i, \\ \mathit{df}_j &= \text{number of documents in which term } j \text{ occurs}, \\ w_j &= \text{number of words in term } j, \\ N &= \text{number of documents.} \end{aligned}$$

The symbol d_{ijk} represents the combined weight of both terms j and k in document i. It is defined as:

$$d_{ijk} = \mathit{tf}_{ijk} \times \log(\frac{N}{\mathit{df}_{jk}} \times w_j), \tag{2}$$

where

$$\begin{aligned} \mathit{tf}_{ijk} &= \text{number of occurrences of both terms } j \text{ and } k \text{ in document } i, \\ &\quad \text{i.e., } \min(\mathit{tf}_{ij}, \mathit{tf}_{ik}), \\ \mathit{df}_{jk} &= \text{number of documents in which both terms } j \text{ and } k \text{ occur.} \end{aligned}$$

Finally, $\mathit{WeightingFactor}(k)$ is defined as:

$$\mathit{WeightingFactor}(k) = \frac{\log(N/\mathit{df}_k)}{\log N}.$$

The term $\mathit{WeightingFactor}(k)$ is used as a weighting scheme (similar to the concept of inverse document frequency) to penalize general terms (terms that appear in many documents). Terms with a high df_k value has a small weighting factor, which results in a small association value. Note that the associations are asymmetric, that is, W_{jk} and W_{kj} are not necessarily equal. Chen showed that this asymmetric similarity function (W_{jk}) gives a better association measure than the popular cosine function [7].

In the following discussion, for simplicity, we assume that $w_j = 1$ for all j (i.e., all terms are single-word ones). We thus remove the term w_j from the formula of d_{ij} and d_{ijk}.

As we have mentioned in the introduction, for the purpose of thesaurus construction, only the highly-ranked associations are needed. In particular, given a term j and a user-specified parameter n, we assume that only the n largest associations $W_{jk_1}, W_{jk_2}, \ldots, W_{jk_n}$ of j are kept. Such associations are called *strong associations.* An association W_{jk} that does not crack into the top n values of j is called a *weak association* and is ignored. Setting $n = 100$, for example, has shown to be sufficient for some chosen domains [2]. We call the set of all strong associations a *partial concept space.*

Given a document collection D, we assume that a partial concept space, CS_D is constructed. Let ΔD be a set of documents that is added to D to form a new document collection D', the problem of incremental concept space update is to compute the partial concept space $CS_{D'}$ with respect to D' given D and CS_D.

For notational convenience, the symbols (e.g., df_{ijk}) that we used in the various formulae for concept space construction refer to the quantities with respect to the original document collection D. We use the prime notation (e.g., df'_{ijk}) to denote those quantities with respect to the updated document collection D'. Also, a preceding Δ (e.g., Δdf_{ijk}) denotes a quantity with respect to ΔD. Table 1 summarizes our notations.

Symbol	Description
D	Original document collection
ΔD	A set of documents added to D
D'	$D \cup \Delta D$
N	the number of documents in D
df_j	the number of documents in D that contain term j
tf_{ij}	the number of occurrences of term j in document i
w_j	the number words in term j, assumed 1 for simplicity
df_{jk}	the number of occurrences of both terms j and k in document i
tf_{ijk}	the number of documents in which both terms j and k occur
W_{jk}	association for the term pair j and k
n	the number of associations that are kept for each term
CS_D	the partial concept space of D
Δx	a quantity x with respect to ΔD.
x'	a quantity x with respect to D'
$\widehat{W_{jk}}$	an upper bound of W_{jk}
$\widehat{W'_{jk}}$	an upper bound of W'_{jk}

Table 1. Notations

We assume that the size of ΔD is relatively small compared with that of D. It is thus computationally inexpensive to process ΔD to obtain the various "delta" values.

3 Concept Space Construction

Concept space construction is a two-phase process. In the first phase (automatic indexing), a term-document matrix, *TF* is constructed. Given a document i and a term j, the matrix *TF* returns the term frequency, tf_{ij}. In practice, *TF* is implemented using inverted lists. That is, for each term j, a linked list of [document-id,term-frequency] tuples is maintained. Each tuple records the occurrence frequency of term j in the document with the corresponding id. Documents that do not contain the term j are not included in the inverted list of j.

Besides the matrix, *TF*, the automatic indexing phase also calculates the quantity df_j (the number of documents containing term j) as well as $\sum_{i=1}^{N} tf_{ij}$ (the sum of the term frequency of term j over the whole document collection) for each term j. These numbers are stored in arrays for fast retrieval during the second phase (co-occurrence analysis).

In the co-occurrence analysis phase, associations of every term pair are calculated. According to Equation 1 (page 3), to compute W_{jk}, we need to compute the values of three factors, namely, $\sum_{i=1}^{N} d_{ijk}$, $\sum_{i=1}^{N} d_{ij}$, and $\mathit{WeightingFactor}(k)$. Note that

$$\sum_{i=1}^{N} d_{ij} = \sum_{i=1}^{N}[tf_{ij} \times \log\left(\frac{N}{df_j}\right)] = \log\left(\frac{N}{df_j}\right) \times \sum_{i=1}^{N} tf_{ij}. \qquad (3)$$

Since both df_j and $\sum_{i=1}^{N} tf_{ij}$ are already computed and stored during the automatic indexing phase, $\sum_{i=1}^{N} d_{ij}$ can be computed in constant time. Similarly, $\mathit{WeightingFactor}(k)$ can be computed in constant time as well.

Computing $\sum_{i=1}^{N} d_{ijk}$, however, requires much more work. From Equation 2, one needs to compute df_{jk} (i.e., the number of documents containing both terms j and k) and $\sum_{i=1}^{N} tf_{ijk}$ in order to find $\sum_{i=1}^{N} d_{ijk}$. Figure 1 shows the algorithm Weight for computing W_{jk}. The execution time of Weight is dominated by the for-loop in line 3. Basically, most of the work is spent on scanning the inverted lists of terms j and k to determine $\sum_i d_{ijk}$.

To compute the partial concept space of D', a brute-force approach would be to compute the associations of all term-pairs. For any term j, the associations of j are sorted and only the n largest ones (i.e., those that are strong) are retained. The brute-force method can be made significantly more efficient by first constructing a two-dimensional triangular bit matrix C in the automatic indexing phase. Given two terms j and k, the matrix C indicates whether j and k ever co-exist in any documents. We notice that $W'_{jk} = 0$ if j and k do not co-exist in any documents of D'. The function Weight is thus only executed for

```
WEIGHT($j, k$)
 1  $df_{jk} \leftarrow 0$
 2  $sum\_tf_{ijk} \leftarrow 0$
 3  for each $(i, tf_{ij})$ in the adjacency list of $j$
 4  do
 5      if there exists $(i, tf_{ik})$ in the adjacency list of $k$
 6          then
 7              $df_{jk} \leftarrow df_{jk} + 1$
 8              if $tf_{ij} < tf_{ik}$
 9                  then
10                      $sum\_tf_{ijk} \leftarrow sum\_tf_{ijk} + tf_{ij}$
11                  else
12                      $sum\_tf_{ijk} \leftarrow sum\_tf_{ijk} + tf_{ik}$
13  $sum\_d_{ijk} \leftarrow sum\_tf_{ijk} \times \log(N/df_{jk})$
14  $sum\_d_{ij} \leftarrow sum\_tf_{ij} \times \log(N/df_j)$
15  $weighting\_factor\_k \leftarrow \log(N/df_k)/\log N$
16  return $sum\_d_{ijk} \times weighting\_factor\_k/sum\_d_{ij}$
```

Fig. 1. Function Weight

those j, k pair such that the entry $C(j, k)$ is set. In typical document collections, matrix C is very sparse. For storage efficiency, the matrix is compressed using bit-vector compression techniques [4].

4 Pruning Method

The baseline brute-force algorithm is not particularly efficient. It basically computes all possible non-zero associations before filtering out those that are weak. As an example, we ran the baseline algorithm on a collection of 191,966 documents. The execution time was 146.5 minutes. The main source of inefficiency lies in the Weight function, which scans two inverted lists for every non-zero association. In our collection, there are about 60 millions non-zero associations, and hence the baseline algorithm performed about 120 millions inverted lists scanning.

Our approach to a more efficient algorithm for the incremental update problem is to use an efficient method to compute an upper bound of W'_{jk} (denoted by $\widehat{W'_{jk}}$) using the information of the partial concept space constructed for the old collection D. We then decide whether the association W'_{jk} is a strong association of term j (w.r.t. D') by comparing the upper bound $\widehat{W'_{jk}}$ with a threshold σ_j. The threshold σ_j is chosen such that if $\widehat{W'_{jk}} < \sigma_j$, then W'_{jk} cannot be a strong association; the value W'_{jk} is thus not computed. As we will see later in Section 5, this pruning technique significantly reduces the execution time.

Applying the pruning method thus requires two issues be addressed: (1) how to compute an upper bound $\widehat{W'_{jk}}$, and (2) how to determine the threshold σ_j for a term j.

To determine an upper bound $\widehat{W'_{jk}}$, we assume that certain information about the old collection D is kept. In particular, we assume that for each term j, we keep two values: df_j and $\sum_{i=1}^{N} tf_{ij}$. These values allow us to compute $\sum_{i=1}^{N} d_{ij}$ and $\mathit{Weightingfactor}(k)$ in constant time (see Equation 3). Also, we assume that for each *weak association* W_{jk} of term j w.r.t. the old collection D, an upper bound $\widehat{W_{jk}}$ is available[1]. Finally, we assume that the values of strong associations are kept (in the partial concept space CS_D).

Since $\widehat{W_{jk}}$ is an upper bound of W_{jk}, we have

$$\widehat{W_{jk}} \geq W_{jk} = \frac{\sum_{i=1}^{N} d_{ijk}}{\sum_{i=1}^{N} d_{ij}} \times \mathit{Weightingfactor}(k).$$

Define

$$K_{jk} = \frac{\widehat{W_{jk}} \times \sum_{i=1}^{N} d_{ij}}{\mathit{Weightfactor}(k)},$$

we have,

$$K_{jk} \geq \sum_{i=1}^{N} d_{ijk} = \sum_{i=1}^{N} tf_{ijk} \times \log(\frac{N}{df_{jk}}).$$

By definition,

$$W'_{jk} = \frac{\sum_i d'_{ijk}}{\sum_i d'_{ij}} \times \mathit{WeightingFactor}'(k).$$

Note that

$$\begin{aligned}\sum_i d'_{ij} &= (\sum_i tf_{ij} + \sum_i \Delta tf_{ij}) \times \log(\frac{N + \Delta N}{df_j + \Delta df_j}),\\ \mathit{WeightingFactor}'(k) &= \frac{\log((N + \Delta N)/(df_k + \Delta df_k))}{\log(N + \Delta N)},\\ \sum_i d'_{ijk} &= (\sum_i tf_{ijk} + \sum_i \Delta tf_{ijk}) \times \log(\frac{N + \Delta N}{df_{jk} + \Delta df_{jk}}).\end{aligned}$$

Assuming that $\sum_i tf_{ij}$ and df_j are available and that ΔD is small enough to be processed to obtain $\sum_i \Delta tf_{ij}$, Δdf_j and Δdf_k, we can efficiently compute $\sum_i d'_{ij}$ and $\mathit{WeightingFactor}'(k)$.

Computing $\sum_i d'_{ijk}$, however, requires scanning inverted lists and thus is expensive. To compute an upper bound of W'_{jk} efficiently, we must derive an

[1] If collection D is obtained by adding documents to an even older collection D^- and the partial concept space of D is obtained by applying the pruning algorithm, then the upper bound $\widehat{W_{jk}}$ is obtained as a by-product and is kept for the next incremental update.

efficient method to compute an upper bound of $\sum_i d'_{ijk}$. Here, we consider 3 cases.

Case 1: terms j and k do not co-exist in any document of ΔD.

In this case, $\sum_i \Delta tf_{ijk}$ and Δdf_{jk} are 0. Hence,

$$\begin{aligned}\sum_i d'_{ijk} &= \sum_i tf_{ijk} \times \log(\frac{N+\Delta N}{df_{jk}}),\\ &\le \frac{K_{jk}}{\log(N/df_{jk})} \times \left(\log(\frac{N+\Delta N}{N}) + \log(\frac{N}{df_{jk}})\right),\\ &= K_{jk} \times \left(1 + \frac{\log((N+\Delta N)/N)}{\log(N/df_{jk})}\right).\end{aligned}$$

Notice that df_{jk} is the number of documents in D that contains both terms j and k, we have $df_{jk} \le \min(df_j, df_k) = df_{min}$. Therefore,

$$\begin{aligned}\sum_i d'_{ijk} &\le K_{jk} \times \left(1 + \frac{\log((N+\Delta N)/N)}{\log(N/df_{min})}\right),\\ &= K_{jk} \times \frac{\log((N+\Delta N)/df_{min})}{\log(N/df_{min})}.\end{aligned}$$

We thus define the upper bound $\widehat{W'_{jk}}$ of W'_{jk} by

$$\widehat{W'_{jk}} = \frac{K_{jk}}{\sum_i d'_{ij}} \times \frac{\log((N+\Delta N)/df_{min})}{\log(N/df_{min})} \times \mathit{Weightfactor}'(k).$$

Case 2: terms j and k co-exist in documents of D and ΔD.

Consider the formula for $\sum_i d'_{ijk}$ again.

$$\sum_i d'_{ijk} = \underbrace{\sum_i tf_{ijk} \times \log(\frac{N+\Delta N}{df_{jk}+\Delta df_{jk}})}_{T_1} + \underbrace{\sum_i \Delta tf_{ijk} \times \log(\frac{N+\Delta N}{df_{jk}+\Delta df_{jk}})}_{T_2}.$$

With a derivation similar to that of case 1, we have

$$T_1 \le K_{jk} \times \frac{\log(\frac{N+\Delta N}{df_{min}+\Delta df_{jk}})}{\log(N/df_{min})}.$$

For T_2, assume that all the "delta" values can be computed efficiently from ΔD, and observe that $df_{jk} \ge 1$ for case 2, we have

$$T_2 \le \sum_i \Delta tf_{ijk} \times \log(\frac{N+\Delta N}{1+\Delta df_{jk}}).$$

We thus define the upper bound $\widehat{W'_{jk}}$ by

$$\widehat{W'_{jk}} = \left(K_{jk} \times \frac{\log(\frac{N+\Delta N}{df_{min}+\Delta df_{jk}})}{\log(N/df_{min})} + \sum_i \Delta tf_{ijk} \times \log(\frac{N+\Delta N}{1+\Delta df_{jk}})\right) \times \frac{\mathit{Weightfactor}'(k)}{\sum_i d'_{ij}}.$$

Case 3: terms j and k only co-exist in documents of ΔD.

In this case, both df_{jk} and $\sum_i tf_{ijk}$ are 0. With the "delta" quantities made available by processing the small ΔD, the values of $\sum_i df'_{ijk}$ and hence W'_{jk} can be computed exactly. Thus, we set $\widehat{W'_{jk}} = W'_{jk}$.

Note that in all three cases, the computational cost of determining the upper bound $\widehat{W'_{jk}}$ is small, since we do not scan the inverted lists of the large document collection D.

Recall that, for a term j, our pruning method uses a threshold σ_j to determine whether the association W'_{jk} should be computed. In particular, if the upper bound $\widehat{W'_{jk}}$ is less than σ_j, then the association W'_{jk} must be weak and should not be computed. To determine σ_j, we compute all n associations W'_{jk_i}'s for which W_{jk_i} is strong w.r.t the old collection D. σ_j is then set to the minimum value of such W'_{jk_i}'s. Given a term k, if $\widehat{W'_{jk}} < \sigma_j$, we know that W'_{jk} must be smaller than all the n W'_{jk_i}'s. Hence, W'_{jk} must not be strong.

Our pruning algorithm for the incremental update problem then goes as follows. For a term j, and a strong association W_{jk_p} w.r.t. D, we compute the association W'_{jk_p} w.r.t. to the updated collection D' using the Weight function. Among the n such associations of term j, we determine the threshold σ_j. After that, for any term k such that W_{jk} is weak w.r.t. D, we compute the upper bound $\widehat{W'_{jk}}$ of W'_{jk}. If $\widehat{W'_{jk}}$ is larger than σ_j, W'_{jk} is computed using the Weight function. Finally, only the n largest associations of j are kept in the partial concept space of D'. The upper bounds calculated in the algorithm are also retained for the next incremental update.

4.1 Quantization

With our pruning method, if the association W_{jk} is weak w.r.t. to D, we have to determine an upper bound $\widehat{W'_{jk}}$ of W'_{jk}. To compute $\widehat{W'_{jk}}$, we require that an upper bound $\widehat{W_{jk}}$ (w.r.t. D) be available[2]. Since the number of such bounds is quadratic with respect to the total number of keywords, the amount of storage required for storing all the $\widehat{W_{jk}}$'s is very big. Fortunately, we do not need to represent the bounds in high precision. Quantization techniques can be applied so that a bound is represented by a small number of bits.

Let W_{jk_n} be the n-th largest association of term j. If W_{jk} is a weak association in D, we know that $\widehat{W_{jk}} < W_{jk_n}$. We divide the interval $[0, W_{jk_n}]$ into 16 equal quantization levels, and $\widehat{W_{jk}}$ is then rounded up to the nearest level. Each bound can thus be represented by a 4-bit codeword. We call the modified pruning method that uses such quantized bounds the *common quantization algorithm*, CQA.

Note that the quantization error introduced makes a bound *looser* than what it should be. A looser bound would make the pruning less effective. Hence, the

[2] $\widehat{W'_{jk}}$ is expressed in terms of K_{jk}, which is in turn expressed in terms of $\widehat{W_{jk}}$.

execution time of CQA is expected to be a bit larger than that of the basic pruning algorithm.

To reduce the quantization error, we can quantize the *differences* between bounds instead of their absolute values. Given a term j, we first compute all the bounds of the weak associations of j. These bounds are then sorted in decreasing value forming a sequence $\widehat{W_{jk_{n+1}}}, \widehat{W_{jk_{n+2}}}, \ldots$. We put the n-th largest association (W_{jk_n}) of j in front of the sequence and compute the differences between successive values in the sequence as shown below.

$$W_{jk_n} \underbrace{-}_{\rho_{n+1}} \widehat{W_{jk_{n+1}}} \underbrace{-}_{\rho_{n+2}} \widehat{W_{jk_{n+2}}} \cdots$$

The largest difference ρ_{max} is determined and we divide the interval $[0,\rho_{max}]$ into 16 levels. Each difference ρ_i can then be represented by a 4-bit codeword. Knowing the value of W_{jk_n} and the differences, all the bounds can be re-computed. We call this quantization scheme the *differential quantization algorithm*, DQA.

5 Performance Evaluation

In this section we evaluate the performance of our pruning algorithm and its variants CQA and DQA. We applied the algorithm on "The Ohsumed Test Collection" [9], which is a medical document collection. The document collection consists of 348,566 abstracts with 240,247 terms. The document database is 169 MB large (after stop-word removal and stemming). We ran the algorithms on a 700 MHz Pentium III Xeon machine.

In our experiment, half of the documents are randomly picked as the original collection D. Collection D contains 174,566 documents. We compute the partial concept space of D. Also, for any weak association W_{jk} of D, we compute its upper bound, $\widehat{W_{jk}}$. For the experiment evaluating CQA and DQA, the upper bounds are quantized according to the quantization schemes described in the last section. We partition the other half (174,000 documents) into 10 equal parts, with 17,400 documents apiece. These parts are added to D successively and cumulatively. The first update thus increases the collection size by 10%, while the 10th update increases the collection size by about 5%.

Figure 2 shows the runtime of the pruning algorithm (without quantization) over the 10 updates under different values of n. (Recall that n is the number of strong associations per term that are kept in a partial concept space.) From the figure, we see that the execution time is larger when the update number increases. This is because the collection size is made bigger by the updates. For example, the collection before the 10th update contains 331,166 documents, which is about 90% larger than the collection before the 1st update. We see that the execution time of the pruning algorithm is linearly proportional to the size of the collection.

Figure 2 also shows that a larger n increases the execution time of the pruning algorithm. Recall that in the pruning algorithm, for a given term j, we use the n-th largest association value W_{jk_n} of j as the pruning threshold. A larger n means

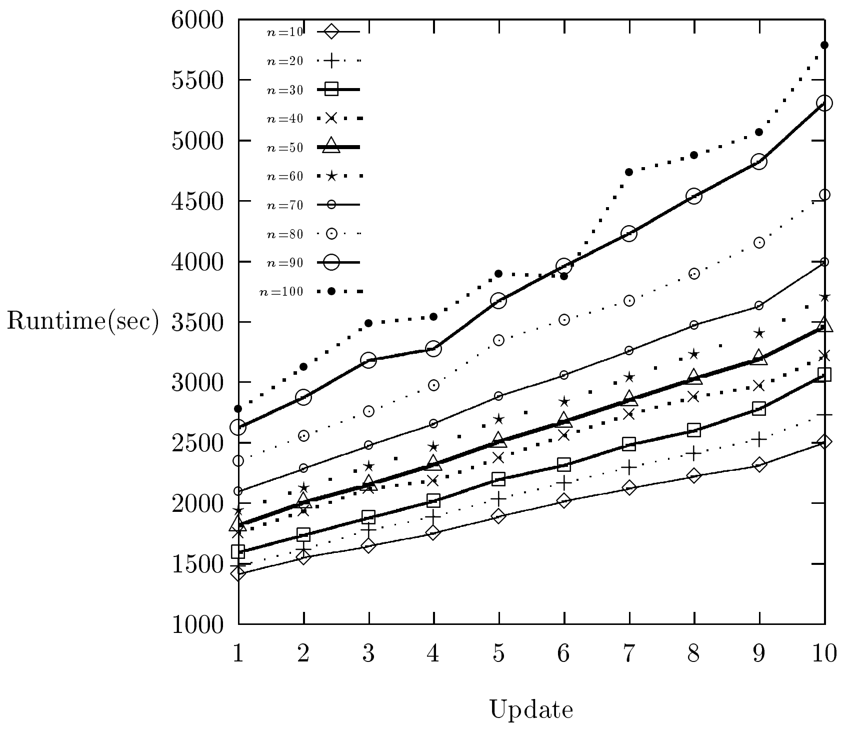

Fig. 2. Runtime of the pruning algorithm over the 10 updates under different values of n

a smaller pruning threshold and thus more associations have to be computed. This fact is illustrated by Figure 3, which shows the number of associations computed by the pruning algorithm under different values of n.

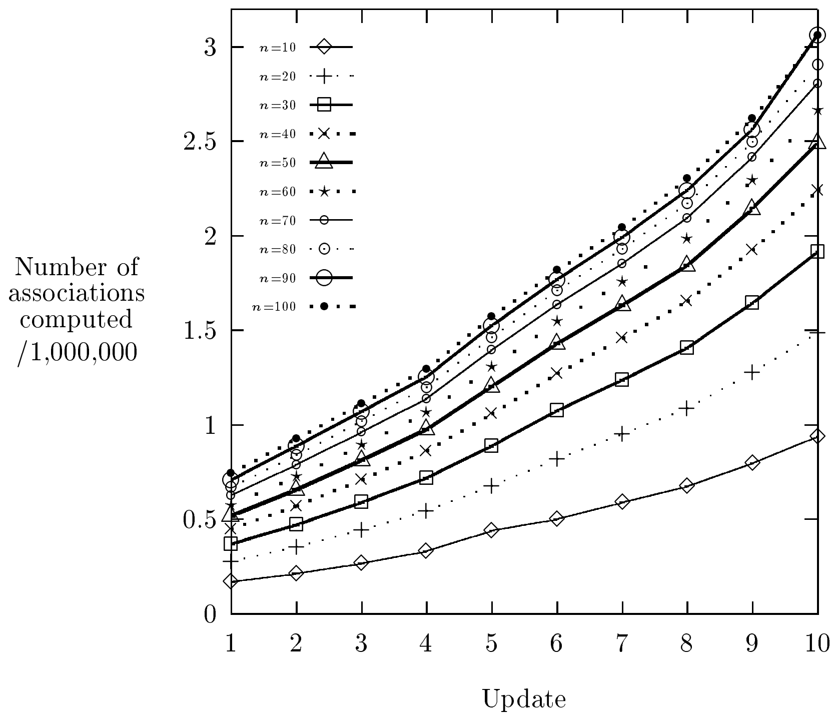

Fig. 3. The number of associations computed by the pruning algorithm under different values of n

Figure 4 compares the performance of the 4 algorithms when n is set to 100. We observe that the pruning algorithm and its quantization variants CQA and DQA significantly outperform the baseline brute-force approach. This is because the pruning methods avoid the expensive computation of much of the associations. Figure 5 compares the number of associations computed by the baseline algorithm and by the pruning algorithm. From the figure, we see that our pruning method is very effective in avoiding the computation of weak associations.

From Figure 4, we see that the execution time of CQA is slightly higher than that of the basic pruning algorithm. This is because the quantization noise makes the association upper bounds less tight. As we have explained in Section 4, this reduces the pruning effectiveness of CQA. Figure 6 shows the number of associations computed by the three pruning algorithms. We see that CQA computes roughly twice as many associations compared with the basic pruning algorithm and DQA.

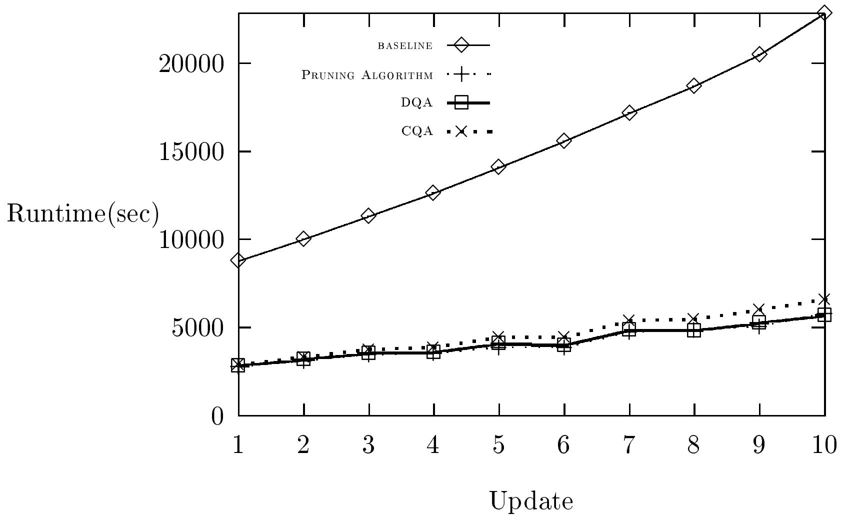

Fig. 4. Algorithms' runtimes over the 10 updates, $n = 100$

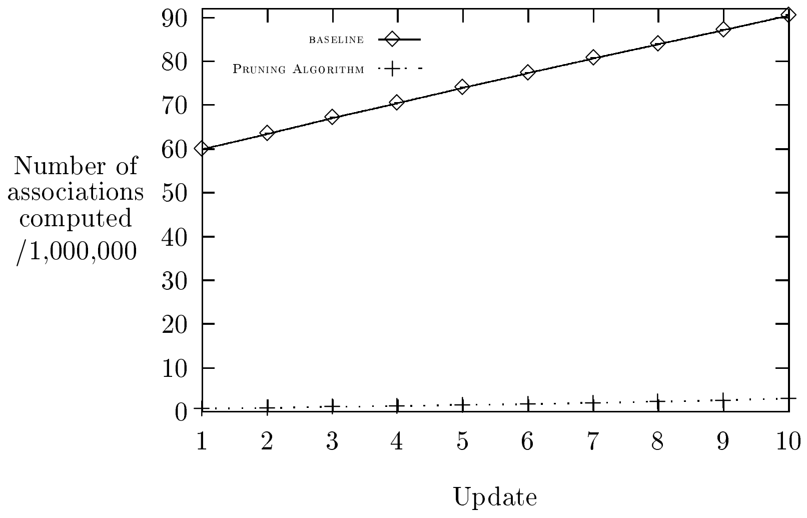

Fig. 5. The number of associations computed by the pruning algorithm and the baseline algorithm

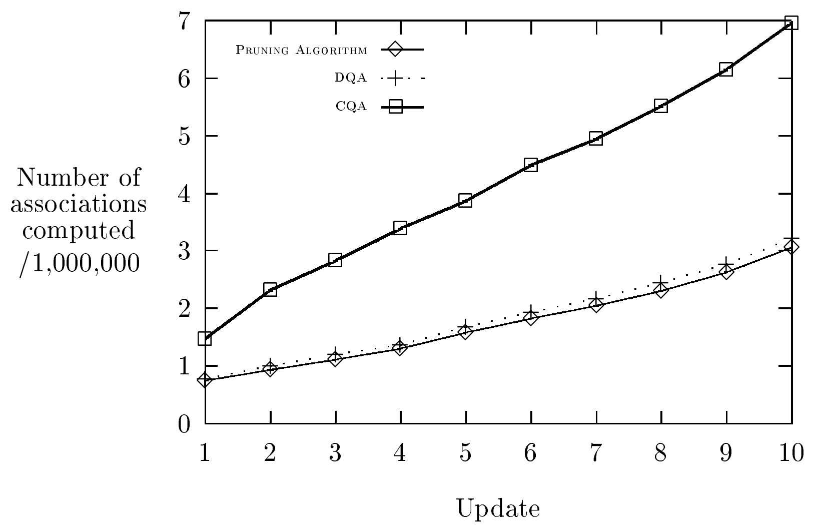

Fig. 6. Number of associations computed by the basic pruning algorithm, CQA, and DQA

By employing differential quantization, quantization errors are made much smaller in DQA. Figure 6 shows that the pruning effectiveness of DQA is very close to that of the basic pruning algorithm. The result is that DQA is as efficient as the basic method.

Another factor that affects the performance of the pruning algorithms is the size of added documents (ΔD). In our description of the algorithms, we assume that ΔD is small enough such that the time spent in processing ΔD to obtain the various "delta" values is acceptable. Figure 7 shows the execution times of the various algorithms in the first update when $|\Delta D|$ changes from 17,400 to 174,000. (For reference, $|D| = 174,566$.) From the figure, we see that the execution time of the baseline brute-force algorithm increases with $|\Delta D|$. This is because the baseline algorithm scans the whole collection ($D + \Delta D$) to compute association values. A larger ΔD means a larger collection and hence more work for the baseline algorithm. Also, since most of the work done by the pruning algorithms is on processing ΔD, we observe a similar increase in execution times for the pruning algorithms as ΔD increases.

Finally, we remark that the baseline algorithm is more storage efficient than the pruning algorithms. Basically, the baseline algorithm only maintains a partial concept space that stores only the strong associations. The basic pruning method, on the other hand, stores the upper bounds of the weak associations as well. This storage overhead could be significant. To reduce the storage cost, CQA and DQA quantize the bounds. In our experiment, the storage requirement of CQA and DQA ranges from 300MB to 400MB. This is about 1/4 to 1/3 of the storage required to maintain a full concept space.

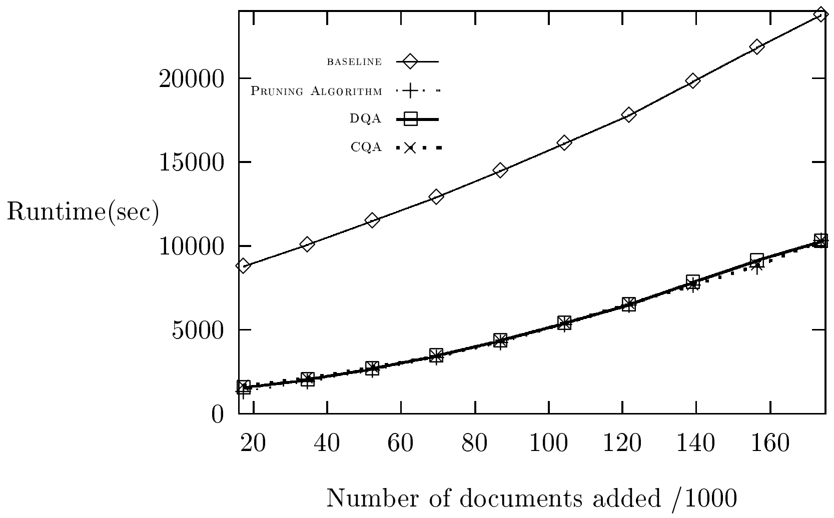

Fig. 7. Algorithms' performance vs. $|\Delta D|$, $n = 10$

6 Conclusion

This paper studied the problem of incremental update of concept spaces. Previous studies have shown that the concept space approach to automatic thesaurus construction is a useful tool for information retrieval. The construction and incremental update of concept spaces, however, are very time consuming. In many applications, a full concept space is not needed, in particular, only a few strong associations per keyword are used. We proposed a pruning algorithm for incremental update of concept spaces that contain only strong associations. To reduce the storage requirement of the pruning algorithm, we propose two quantization variants, namely, CQA and DQA. Our experiment shows that the pruning algorithms are very effective in avoiding the computation of weak associations. As an example, if the number of associations to be maintained for each keyword (i.e., n) is 10, and that the size of the added documents is about 10% of the original collection, our experiment registered a 9-time speedup of the pruning algorithms compared with the brute-force approach.

References

1. R. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.
2. H. Chen, T. Yim, D. FYE, and B. SCHATZ. Automatic thesaurus generation for an electronic community system. *Journal of American Society for Information Science*, 46(3):175–193, 1995.
3. Hsinchun Chen, Joanne Martinez, Tobun D. Ng, and Bruce R. Schatz. A concept space approach to addressing the vocabulary problem in scientific information re-

trieval: an experiment on the worm community system. *Journal of American Society for information Science*, 48(1):17–31, 1997.

4. Y. Choueka, A.S. Fraenkel, S.T. Klein, and E. Segal. Improved hierarchical bit-vector compression in document retrieval systems. In *In Proc. 9th ACM-SIGIR Conference on Information Retrieval*, pages 88–97, Pisa, Italy, September 1986.
5. W.B. Frakes and R. Baeza-Yates. *Information Retreival: Data Structures and Algorithms.* Prentice Hall, 1992.
6. G.W. Furnas et al. The vocabulary problem in human-system communicaiton. *Comm. ACM*, 30(11):964–971, 1987.
7. H.Chen and K.J. Lynch. Automatic construction of networks of concepts characterizing document databases. *IEEE Transaction of Systems, Man, and Cybernetics*, 22(5):885–902, Sep/Oct 1992.
8. B.R. Schatz, E. Johnson, P. Cochrane, and H. Chen. Interactive term suggestion for users of digital libraries: using subject thesauri and co-occurrence lists for information retrieval. In *Digital Library 96*, Bethesda MD, 1996.
9. Hersh WR, Buckley C, Leone TJ, and Hickam DH. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual ACM SIGIR Conference*, pages 192–201, 1994. http://www1.ics.uci.edu/pub/machine-learning-databases/ohsumed/.