Chapter 1

EFFICIENT CLUSTERING OF VERY LARGE DOCUMENT COLLECTIONS

Inderjit S. Dhillon, James Fan and Yuqiang Guan

Abstract An invaluable portion of scientific data occurs naturally in text form. Given a large unlabeled document collection, it is often helpful to organize this collection into clusters of related documents. By using a vector space model, text data can be treated as high-dimensional but sparse numerical data vectors. It is a contemporary challenge to efficiently preprocess and cluster very large document collections. In this paper we present a time and memory efficient technique for the entire clustering process, including the creation of the vector space model. This efficiency is obtained by (i) a memory-efficient multi-threaded preprocessing scheme, and (ii) a fast clustering algorithm that fully exploits the sparsity of the data set. We show that this entire process takes time that is linear in the size of the document collection. Detailed experimental results are presented — a highlight of our results is that we are able to effectively cluster a collection of 113,716 NSF award abstracts in 23 minutes (including disk I/O costs) on a single workstation with modest memory consumption.

Keywords: clustering, large document collections, hash tables, spherical k-means, vector space model

1. Introduction

Large collections of documents are becoming increasingly common. The public internet currently has more than 1.5 billion web pages, while private intranets also contain an abundance of text data. A vast amount of important scientific data appears as technical abstracts and papers. Given such large document collections it is important to organize them into structured ontologies. This organization facilitates navigation and search, and at the same time provides a framework for continual maintenance as document repositories grow in size.

Manual construction of structured ontologies is one possible solution and has been adopted to organize the internet (www.yahoo.com) and to structure library content. However, this process has the obvious disadvantage of being too labor intensive, and is viable only in large corporations. Thus it is desirable to seek automatic methods for organizing unlabeled document collections. Given a collection of unlabeled data points, *clustering* refers to the problem of automatically assigning class labels to the data and has been widely studied in statistical pattern recognition and machine learning [DH73, Mit97].

A starting point for applying clustering algorithms to unstructured text data is to create a *vector space model*, alternatively known as a *bag-of-words model* [SM83]. The basic idea is (a) to extract unique content-bearing words from the set of documents treating these words as *features* and (b) to then represent each document as a vector of certain weighted word frequencies in this feature space. Observe that we may regard the vector space model of a text data set as a *word-by-document matrix* whose rows are words and columns are document vectors. Typically, a large number of words exist in even a moderately sized set of documents where a few thousand words or more are common. Thus for large document collections, both the row and column dimensions of the matrix are quite large. However, as we will discuss later in greater detail, this matrix is typically very sparse with almost 99% of the matrix entries being zero.

Using the vector space model various classical clustering algorithms such as the k-means algorithm and its variants, hierarchical agglomerative clustering, and graph-theoretic methods have been explored in the text mining literature; for detailed reviews, see [Ras92, Wil88]. Recently, there has been a flurry of activity in this area, see [BGG$^+$98, CKPT92, SS97, ZE98]. A substantial amount of this work has concentrated on clustering web search results where the document collections to be clustered are not very large.

In this paper, our main concern is in obtaining a highly efficient process for clustering very large document collections. Our main motivation is that we want to cluster collections in excess of more than 100,000 documents in a reasonable amount of time on a single processor. Thus our main emphasis is on high speed and scalability with modest main memory consumption. The clustering process involves reading the text documents from disk, and preprocessing them to form the vector space model before using a particular clustering algorithm. It turns out that main memory consumption and disk I/O costs in this process can be prohibitive. In order to alleviate these problems, we employ a memory-efficient multi-threaded approach to reading and preprocessing the docu-

ments. For creating the vector space model, we use efficient and scalable data structures such as local and global hash tables. Finally, we use the highly efficient and effective *spherical k-means* algorithm that fully exploits the sparsity of the data [DM01]. The above steps lead to a highly efficient clustering process — as a result we are able to preprocess and cluster a collection of 113,716 NSF award abstracts documents in 23 minutes on a Sun workstation with modest memory consumption.

This paper is organized as follows. In Section 2 we highlight the challenges in obtaining an efficient and effective process for clustering large document collections. Section 3 describes our multi-threaded algorithm for creating the vector space model. In Section 4 we present the spherical k-means algorithm which is ideally suited for clustering high-dimensional sparse text data. Section 5 presents speed and memory consumption results on some document collections, in particular we examine the results on a collection of 113,716 NSF award abstracts. Finally, Section 6 concludes with a short summary and discussion of future work.

A quick word about notation. Lower case bold letters such as $\mathbf{x}$, $\mathbf{c}$ will denote vectors while $\|\mathbf{x}\|$ will denote the 2-norm of the corresponding vector. Thus when we say that $\|\mathbf{x}\| = 1$ we mean that $\mathbf{x}^T\mathbf{x} = 1$.

2. Challenges

As mentioned above, the preprocessing phase leads to the creation of the vector space model. Given a large document collection, the following steps are involved in this phase: (a) read all input documents from disk, (b) parse into word tokens using efficient regular expression pattern matching, (c) lookup words rapidly in hash tables to track the number of occurrences of each word, and finally, (d) output the vector space model.

Efficient parsing using regular expression pattern matching is a well-studied problem. Existing software tools such as FLEX provide a quick and easy way to construct the required parser [Pax96]. Also, efficient hash table indexing and access is provided in public-domain software [MS96]. Thus effective solutions to steps (b) and (c) above are easy to obtain.

However I/O costs in steps (a) and (d) can be substantial, especially if the documents need to be accessed from a Network File System(NFS) [Cal99]. In such a case, the time taken by these steps can be quite large and often unpredictable depending on other traffic on the NFS. Our approach to solve this problem is to use multiple threads [NBF96] so that input documents can be read in a parallel fashion.

Another challenge is main memory consumption. After all the documents have been read and processed there is a final phase which involves a sweep and modification of the entire vector space model. One simple approach to facilitate this phase is to retain the partially formed vector space model in main memory while the rest of the documents are being processed. We implemented such an approach and on our test collection of 113,716 documents, this required nearly 500 MBytes of main memory. Extrapolating this memory requirement to 1 million documents, we can see that this main-memory based implementation will require several gigabytes of memory. This cost is unacceptable since our goal is to preprocess and cluster a million documents on current workstations consuming less than 256 MBytes of main memory. In Section 3 we introduce such a memory efficient algorithm that allows us to preprocess a large number of documents without sacrificing much speed.

In our test collection of 113,716 documents there are a total of more than 150,000 unique words. After a pruning step, we retain 26,000 unique words in the vector space model. Thus, the document vectors are very *high-dimensional*. However, typically, most documents contain only a small subset of the total number of words. Hence, the document vectors are very *sparse* — a sparsity of 99% is common. The major challenge is to find a clustering algorithm that can yield good effective solutions for very high-dimensional data and at the same time, exploit the sparsity of the vector space model. Since we are interested in clustering very large document collections we seek algorithms that consume time and memory that is linear in the size of the document collection.

3. Efficient Preprocessing

In this section, we describe our preprocessing algorithm for creating the vector space model. Before we describe the details we give a high level description of the task at hand.

3.1. Vector Space Model

The basic idea is to represent each document as a vector of certain weighted word frequencies. In order to do so, the following parsing and extraction steps are needed.

1 Ignoring case, extract all unique words from the entire set of documents.

2 Eliminate non-content-bearing "stopwords" such as "a", "and", "the", etc. For sample lists of stopwords, see [FBY92, Chapter 7].

3 For each document, count the number of occurrences of each word.

4 Using heuristic or information-theoretic criteria, eliminate non-content-bearing "high-frequency" and "low-frequency" words [SM83].

5 After the above elimination, suppose w unique words remain. Assign a unique identifier between 1 and w to each remaining word, and a unique identifier between 1 and d to each document.

The above steps outline a simple preprocessing scheme. In addition, one may extract word phrases such as "New York," and one may reduce each word to its "root" or "stem", thus eliminating plurals, tenses, prefixes, and suffixes [FBY92, Chapter 8].

The above preprocessing yields the number of occurrences of word j in document i, say, f_{ji}, and the number of documents which contain the word j, say, d_j. Using these counts, we can represent the i-th document as a w-dimensional vector $\mathbf{x}_i$ as follows. For $1 \leq j \leq w$, set the j-th component of $\mathbf{x}_i$, to be the product of three terms

$$x_{ji} = t_{ji} \cdot g_j \cdot s_i,$$

where t_{ji} is the *term weighting component* and depends only on f_{ji}, while g_j is the *global weighting component* and depends on d_j, and s_i is the *normalization component* for $\mathbf{x}_i$. Intuitively, t_{ji} captures the relative importance of a word in a document, while g_j captures the overall importance of a word in the entire set of documents. The objective of such weighting schemes is to enhance discrimination between various document vectors for better retrieval effectiveness [SB88].

There are many schemes for selecting the term, global, and normalization components, see [Kol97] for various possibilities. In this paper we use the popular tfn scheme known as *normalized term frequency-inverse document frequency*. This scheme uses $t_{ji} = f_{ji}$, $g_j = \log(d/d_j)$ and $s_i = \left(\sum_{j=1}^{w} (t_{ji} g_j)^2\right)^{-1/2}$. Note that this normalization implies that $\|\mathbf{x}_i\| = 1$, i.e., each document vector lies on the surface of the unit sphere in R^w. Intuitively, the effect of normalization is to retain only the *proportion* of words occurring in a document. This ensures that documents dealing with the same subject matter (that is, using similar words), but differing in length lead to similar document vectors.

3.2. Preprocessing Algorithm

The input to the preprocessing step is a data directory that contains all the documents to be processed. The documents may also be contained within subdirectories of the input directory. The output is the vector space model described in the above section, which can be represented as

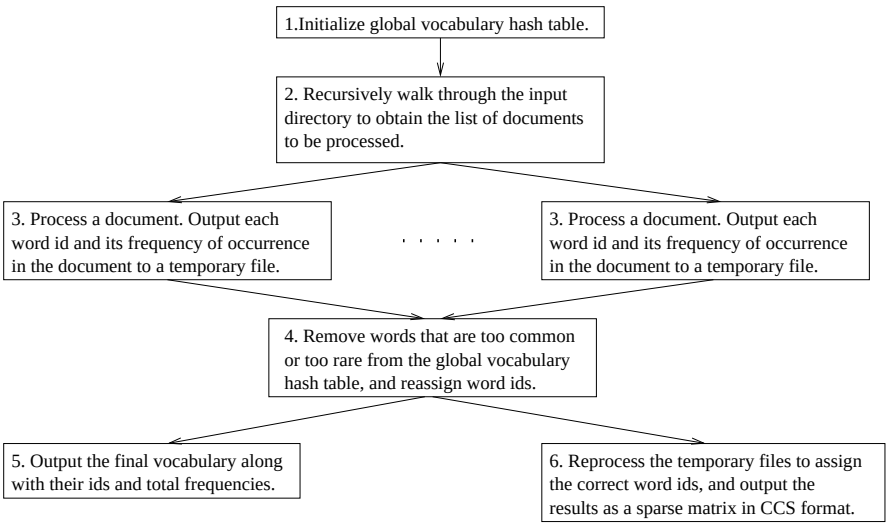

Figure 1.1. Outline of the preprocessing algorithm

a highly sparse word-by-document matrix. We store this sparse matrix by using the Compressed Column Storage(CCS) format [DGL89]. In this format, we record the value of each non-zero element, along with its row and column index. The column indices represent the input documents, the row indices represent ids of distinct words present in the document collection, and the non-zero entries in the matrix represent the frequencies of words in documents.

Figure 1.1 gives an outline of the preprocessing algorithm. The algorithm first initializes a global hash table. To resolve whether a word has been encountered previously, a local and a global hash table are used. Both these hash tables use words as keys and store the corresponding row indices and frequencies as values. As the names suggest, the global hash table keeps track of words and their occurrences in the entire document collection while the local hash table does so for just one document. After initializing the global hash table, the algorithm recursively walks through the input directory to obtain the list of documents to be processed. The preprocessing algorithm then creates several threads of computation. The purpose of each thread is to process a set of documents independently and output its results into temporary files. Details of this processing are given in Figure 1.2 which we discuss in the next paragraph. After all the threads have finished, the global hash table is examined, and words that are too common or too rare are removed from the global hash table. Unique word ids are assigned to the

1. Initialize the local hash table (this hash table uses words as keys, and stores row indices and frequencies as values).
2. Get a token from the document, and convert the token to lowercase.
3. Discard the token if it is too long or too short or is a non-content "stopword" such as "a", "and", "the", etc.
4. If the word already exists in the local hash table, increment the frequency of that entry, otherwise insert the word into the local hash table with row index -1 and frequency 1.
5. After the whole document has been processed, set the row indices in the local hash table to the corresponding ones in the global hash table. If a word in the local hash table does not exist in the global hash table, assign a new row index to the word and add to the global hash table (note that this requires locking the global hash table to prevent simultaneous modification by another thread).
6. Output the contents of the local hash table (row indices and frequencies) to a temporary file, and discard the local hash table.

Figure 1.2. Details on the processing of a document by each thread (step 3 of Figure 1.1)

words that still remain in the global hash table. The temporary files are then reloaded, the word ids are resolved and then the final vocabulary and word-by-document matrix are output.

Figure 1.2 describes the various steps performed by each preprocessing thread. Two decisions warrant further explanation — the use of temporary files for storing the partial vector space model and the way in which the local and global hash tables are accessed. As mentioned in the last section, storing the partial vector space model in main memory would require a few gigabytes of main memory and is thus prohibitive for modern workstations. Hence to reduce main memory consumption we store the contents of the local hash table onto temporary files. Since this only leads to local disk access, the resulting overhead is not substantial.

The global hash table is accessed and modified by all processing threads and hence is a shared resource. In order to achieve maximum parallelism, we need to minimize the number of times the global hash table is locked and modified by each processing thread. We achieve this

by using a local hash table to process the entire document first, and then making a block access to the global hash table. This access involves resolving the word ids and possible modification of the global hash table, at which time this data structure needs to be locked. See Figure 1.2 for details, especially step 5.

The major main memory consumption in our preprocessing algorithm is due to the global vocabulary table. The partial vector space model is stored in temporary files instead of main memory. Thus the overall main memory requirement is $O(W)$, where W is the number of distinct words that appear in the document collection. It is well known that W grows slower than linearly with the size of the document collection — *Heaps' Law* states that the number of unique words in a text of size d grows as $O(d^{\beta})$, where β is a positive number less than one[Hea78]. Thus the main memory consumption grows slower than linearly with the size of the document collection. The computation time for the preprocessing step is approximately linear with respect to the input data size since each word in a document is processed in $O(1)$ amortized time. Performance results on large document collections that validate these claims are given in Section 5.

4. Efficient Clustering of High-Dimensional Text Data

Given the vector space model, the document vectors may be represented by $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$, where each $\mathbf{x}_i \in R^w$. Recall that w stands for the number of unique words in the vector space model and d is the total number of documents. A clustering of the document collection is its partitioning into the disjoint subsets $\pi_1, \pi_2, ..., \pi_k$, i.e.,

$$\bigcup_{j=1}^{k} \pi_j = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_d\} \quad \text{and} \quad \pi_j \cap \pi_l = \phi, \quad j \neq l.$$

The most important and challenging characterisitics of the vector space models that arise from text data are high dimensionality and sparsity. Typically, w is in the thousands and a sparsity of 99% is common. For purposes of efficiency, it is important that the clustering algorithm exploit the sparsity of the data while giving meaningful results at the same time. The spherical k-means algorithm satisfies both these properties and hence is our algorithm of choice. We now briefly formalize this algorithm highlighting its salient features. More details may be found in [DM01].

Any text clustering algorithm needs an objective notion of similarity between documents. A widely used measure of similarity is the cosine

of the angle between two document vectors [FBY92, SM83]. Cosine similarity is easy to interpret and simple to compute for sparse vectors and has been used in other information retrieval applications, such as query retrieval. Based on cosine similarity, we can define the "goodness" or "coherence" of cluster π_j as

$$\sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i^T \mathbf{c}_j, \tag{4.1}$$

where each $\mathbf{x}_i$ is assumed to be normalized such that $\|\mathbf{x}_i\| = 1$ and $\mathbf{c}_j$ is the normalized centroid of cluster π_j,

$$\mathbf{c}_j = \frac{\sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i}{\| \sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i \|}.$$

By the Cauchy-Schwarz inequality,

$$\sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i^T \mathbf{z} \leq \sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i^T \mathbf{c}_j, \qquad \forall \mathbf{z} \in R^w,$$

and thus the normalized centroid is the vector that is closest in cosine similarity (in an average sense) to all the document vectors in the cluster π_j. As a result, we also call the vector $\mathbf{c}_j$'s as *concept vectors*.

Aggregating (4.1) over all clusters, we can measure the goodness of any given partitioning $\{\pi_j\}_{j=1}^k$ using the following *objective function*:

$$\mathcal{Q}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i^T \mathbf{c}_j. \tag{4.2}$$

Intuitively, the objective function measures the combined coherence of all the k clusters. Having posed the objective function, we now present an algorithm that attempts to maximize its value.

4.1. Spherical *k*-means Algorithm

It is well known that finding the partitioning that maximizes (4.2) is NP-Complete [KPR98, Theorem 3.1]; also, see [GJW82]. Thus we seek a heuristic algorithm that can quickly find a good local maximum. For this purpose, we use a variant of the classical k-means algorithm [DH73] that uses the cosine similarity measure. Since both the document and concept vectors lie on the surface of a high-dimensional sphere, we call this variant the *spherical k-means* algorithm. This algorithm proceeds as follows.

1 Initialize clustering. Start with some initial partitioning of the document vectors, namely $\{\pi_j^{(0)}\}_{j=1}^k$. Let $\{\mathbf{c}_j^{(0)}\}_{j=1}^k$ be the concept vectors of the associated partitioning. Set the iteration count t to 0.

2 Re-assign document vectors. For each document vector $\mathbf{x}_i$, $1 \leq i \leq d$, do the following:

a. Compute $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ for all $l = 1, 2, \ldots, k$.

b. From among all $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ computed above, find $j = \arg\max_l \mathbf{x}_i^T \mathbf{c}_l^{(t)}$ (break ties arbitrarily if $\mathbf{x}_i$ has largest cosine similarity with more than one concept vector).

This induces the new partitioning

$$\pi_j^{(t+1)} = \{\mathbf{x}_i : j = \arg\max_l \mathbf{x}_i^T \mathbf{c}_l^{(t)}\}, \quad 1 \leq j \leq k.$$

3 Update concept vectors. Compute the concept vectors corresponding to the new partitioning:

$$\mathbf{s}_j = \sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i, \qquad \mathbf{c}_j^{(t+1)} = \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}, \quad 1 \leq j \leq k.$$

4 Check the stopping criterion. If the stopping criterion is satisfied, then exit. Otherwise increase t by 1 and go to step 2 above.

In our implementation, the stopping criterion is:

$$|\mathcal{Q}(\{\pi_j^{(t)}\}_{j=1}^k) - \mathcal{Q}(\{\pi_j^{(t+1)}\}_{j=1}^k)| \;\leq\; 10^{-3} \cdot |\mathcal{Q}(\{\pi_j^{(t)}\}_{j=1}^k)|.$$

It can be shown that the above algorithm yields a gradient-ascent scheme. In particular, we can show that the objective function value in (4.2) does not decrease from one iteration to the next, i.e.,

$$\mathcal{Q}(\{\pi_j^{(t)}\}_{j=1}^k) \;\leq\; \mathcal{Q}(\{\pi_j^{(t+1)}\}_{j=1}^k).$$

Like any other gradient-ascent scheme, the spherical k-means algorithm is prone to local maxima. A careful selection of initial partitions

$\{\pi_j^{(0)}\}_{j=1}^k$ is important. One can either (a) randomly assign each document to one of the k clusters, (b) first compute the concept vector for the entire document collection and randomly perturb this vector to get k starting concept vectors or (c) try several initial clusterings and select the best in terms of the largest objective function. We use strategy (b) in our implementation.

We now examine the time and memory complexity of the above algorithm. We assume that the number of non-zero entries in the sparse matrix is nz, and that the above algorithm iterates τ times before stopping. Using our initialization strategy, step 1 of the algorithm costs $O(nz + k \cdot w)$ operations. For each iteration, step $2a$ costs $O(nz \cdot k)$ operations while $2b$ costs $O(k \cdot d)$ comparisons. Step 3 updates the concept vectors and costs $O(nz + k \cdot w)$ operations. Thus the total time complexity for τ iterations is

$$\mathcal{O}((nz \cdot k + k \cdot d + k \cdot w) \cdot \tau).$$

Typically $nz \gg \max(d, w)$, hence it is clear that step $2a$ is the most computationally expensive step in the algorithm and the overall complexity of the algorithm is $O(nz \cdot k \cdot \tau)$.

The main memory consumed by the algorithm is for storing the document vectors and for old and new copies of the concept vectors. Storing the document vectors in the CCS sparse matrix form requires $4(2nz+d+4)$ bytes while the concept vectors require $8kw$ bytes; hence the memory consumption is modest.

4.2. Similarity Estimation

The computational bottleneck in the spherical k-means algorithm is the dot product computation between all the document vectors and concept vectors (see step 2a). During the course of the algorithm it turns out that the first few iterations lead to a lot of movement of documents between clusters. However, just after a few iterations the clusters become more stable, see the solid line in the left plot of Figure 1.3. Consequently, the overall objective function value also settles down after 4-5 iterations as seen in the right plot of Figure 1.3.

When the clusters become stable, there are potential savings if we can somehow avoid computing unnecessary dot products between document vectors and 'far away' concept vectors. We now introduce a technique for estimating cosine similarities which allows us to avoid such dot product computations.

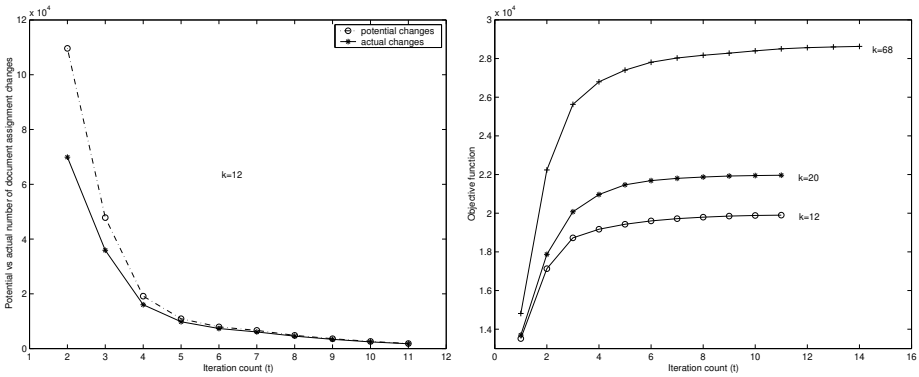

Figure 1.3. Clusters stabilize after just a few iterations.

Suppose $\mathbf{c}_l^{(t)}$ is the concept vector of cluster l at iteration t and $\mathbf{x}$ is a document vector. Then,

$$|\mathbf{x}^T\mathbf{c}_l^{(t)} - \mathbf{x}^T\mathbf{c}_l^{(t+1)}| \;\leq\; \|\mathbf{x}\|\|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t+1)}\| \;=\; \|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t+1)}\|,$$

$$\Rightarrow \mathbf{x}^T\mathbf{c}_l^{(t)} - \|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t+1)}\| \leq \mathbf{x}^T\mathbf{c}_l^{(t+1)} \leq \mathbf{x}^T\mathbf{c}_l^{(t)} + \|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t+1)}\| \quad (4.3)$$

The right side of inequality (4.3) gives a similarity upper bound which can be used profitably in the $(t+1)$-st iteration to avoid computing $\mathbf{x}^T\mathbf{c}_l^{(t+1)}$. The idea is to store in memory the quantities $\|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t+1)}\|$ and upper bounds for $\mathbf{x}^T\mathbf{c}_l^{(t)}$. Suppose $\mathbf{x}$ belongs to the cluster j at the t-th iteration. Then, at iteration $t+1$, we can update in $O(1)$ time the similarity upper bound for $\mathbf{x}^T\mathbf{c}_l^{(t+1)}$, $l \neq j$. If this similarity upper bound is smaller than $\mathbf{x}^T\mathbf{c}_j^{(t+1)}$ (which is computed exactly) there is no need to explicitly compute $\mathbf{x}^T\mathbf{c}_l^{(t+1)}$, otherwise the exact value needs to be computed. As the clusters become more and more stable, this similarity estimation can dramatically cut down on computation (see Figure 1.4).

In our implementation, we store the similarity upper bounds in the $d \times k$ matrix U. We obtain the new algorithm by replacing step 2 of the spherical k-means algorithm by the following step (here we assume that similarity estimation is started after t_{min} iterations).

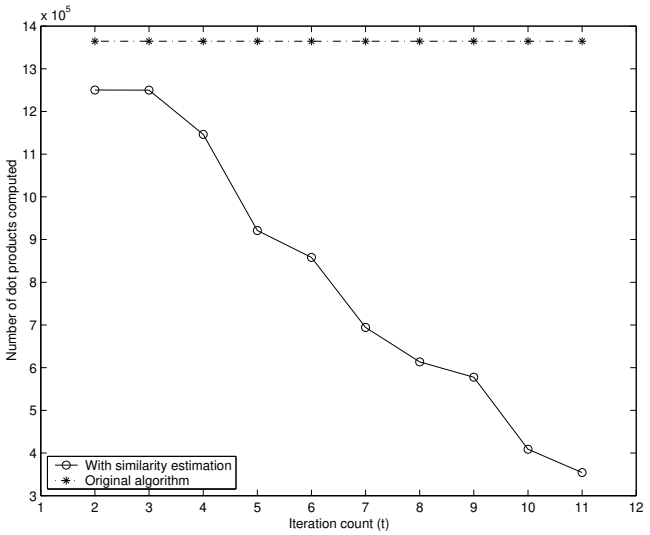

Figure 1.4. Computational Savings due to Similarity Estimation

2. Re-assign document vectors. For each document vector $\mathbf{x}_i$, $1 \leq i \leq d$, do the following:

a. If $t \leq t_{min}$, do step 2a as in Section 4.1.
else if $t = t_{min}$, set $U(i,l) = \mathbf{x}_i^T \mathbf{c}_l^{(t)}$ for all $l = 1, 2, \ldots, k$.
else if $t > t_{min}$, do the following steps for all $l = 1, 2, \ldots, k$,

a1. Set $U(i,l) = U(i,l) + \|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t-1)}\|$.

a2. Compute $\mathbf{x}_i^T \mathbf{c}_j^{(t)}$ where $\mathbf{x}_i$ belongs to cluster j at iteration $t-1$.
If $U(i,l) > \mathbf{x}_i^T \mathbf{c}_j^{(t)}$, compute $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ and set $U(i,l) = \mathbf{x}_i^T \mathbf{c}_l^{(t)}$.

b. From among all $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ computed above, find $j = \arg\max_l \mathbf{x}_i^T \mathbf{c}_l^{(t)}$.

Figure 1.4 shows the considerable savings in the number of dot product computations as the iteration count increases in a typical run of the algorithm. To obtain these savings, an extra storage requirement of $4dk$ bytes is required to store the matrix U. Additionally, an extra $O(dk)$ operations to update the matrix U are required (see step a1 above).

However, these extra costs are small compared to the resulting savings in computation time, see Section 5.1.2 for an example.

4.3. Clustering for Dimensionality Reduction

We now highlight another use of our clustering algorithm. Using the vector space model, each document may be represented as a high-dimensional vector of words. Clearly the occurrence of one word in a document is not independent of other words. Thus there is a great deal of redundancy in this collection of vectors. Dimensionality reduction is a technique that tries to represent each document as a vector with fewer dimensions that are more independent. It turns out that the above clustering algorithm also yields a fast and effective technique for dimensionality reduction.

Let $\{\mathbf{c}_j\}_{j=1}^k$ denote the concept vectors corresponding to a clustering of the document vectors. The *concept matrix* C_k is defined to be a $d \times k$ matrix such that, for $1 \leq j \leq k$, the j-th column of the matrix is the concept vector $\mathbf{c}_j$, i.e,

$$C_k = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_k].$$

An approximation X_k to the word-document matrix X may be obtained by taking the least squares projection of X onto the column space of C_k, i.e., the linear subspace spanned by the concept vectors. This may be expressed as

$$X_k = C_k Z_k^*,$$

where Z_k^* is a $k \times d$ matrix that is to be determined by solving the following least-squares problem:

$$Z_k^* = \arg\min_Z \|X - C_k Z\|_F^2.$$

It is well known that a closed form solution exists for this least-squares problem, namely,

$$Z_k^* = (C_k^T C_k)^{-1} C_k^T X. \tag{4.4}$$

The i-th column of Z_k^* gives the reduced k-dimensional representation of the i-th document vector. Typically the original dimensionality w is in the thousands while k is much smaller. In previous work, we have shown that empirically, the quality of the reduced dimensional representation given by (4.4) is comparable to the "best" possible, namely the k-truncated SVD, see [DM01] for details. Thus our clustering process followed by dimensionality reduction can be used in various applications, such as query retrieval, text classification, etc.

5. Experimental Results

In this section, we give experimental results for the entire clustering process — this includes time and memory consumed by the preprocessing phase as well as by the clustering algorithm. In addition, we need to evaluate the goodness of the clustering produced.

Evaluating clustering results is a tricky business. However, in situations where documents are already categorized(labelled), we can compare the clusters with the "true" class labels. For this comparison we use the measures of purity and entropy as defined below, see also [SGM00].

Suppose we are given c categories (true class labels) while the clustering algorithm produces k clusters. Cluster π_l's *purity* can be defined as

$$P(\pi_l) = \frac{1}{n_l} \max_h (n_l^{(h)}),$$

where $n_l = |\pi_l|$ and $n_l^{(h)}$ is the number of documents in π_l that belong to class h, $h = 1, \ldots, c$. Note that each cluster may contain samples from different classes. Purity gives the ratio of the dominant class size in the cluster to the cluster size itself. A high purity value implies that the cluster is a "pure" subset of the dominant class.

Additionally, we also use *entropy* as a quality measure, which is defined as follows:

$$H(\pi_l) = -\frac{1}{\log c} \sum_{h=1}^{c} \frac{n_l^{(h)}}{n_l} \log \left(\frac{n_l^{(h)}}{n_l} \right).$$

Entropy is a more comprehensive measure than purity. It considers the distribution of classes in a cluster. Note that we have normalized entropy to take values between 0 and 1. An entropy value of 0 means the cluster is comprised entirely of one class, while an entropy value near 1 is bad since it implies that the cluster contains a uniform mixture of classes.

We now examine these quality measures on a sample collection. We formed a collection of 3893 documents by merging the popular MEDLINE, CISI, and CRANFIELD sets. MEDLINE consists of 1033 abstracts from medical journals, CISI consists of 1460 abstracts from information retrieval papers, while CRANFIELD consists of 1400 abstracts from aeronautical systems papers (ftp://ftp.cs.cornell.edu/pub/smart). We preprocessed this collection by proceeding as in Section 3. After removing common stopwords, the collection contained 22149 unique words from which we eliminated 17839 low-frequency words appearing in less than 8 documents (roughly 0.2% of the documents), and 7 high-frequency

words appearing in more than 585 documents (roughly 15% of the documents). We were finally left with 4303 words using which we created 3893 document vectors using the tfn scheme. Each document vector has dimension 4303, however, on an average, each document vector is about 99% sparse.

For this contrived collection, we used our clustering algorithm to form 3 clusters. The following table shows the "confusion matrix" for this clustering, from which we can see that the clusters can be easily identified with the three classes — 'MEDLINE', 'CISI' and 'CRANFIELD'.

	MEDLINE	CRANFIELD	CISI
patients(1023)	**1020**	2	1
boundary(1388)	1	**1383**	4
library (1481)	12	14	**1455**

In the above table the rows denote clusters. Note that we have denoted the clusters by the most frequently occurring word in the cluster from among the words in the vector space model — 'patients', 'boundary' and 'library'. The above table says that the cluster 'patients' has 1023 documents of which 1020 belong to the 'MEDLINE' class, the 'boundary' cluster has 1388 documents while the 'library' cluster has 1481 documents. Notice that the confusion matrix is almost diagonal which shows that the clustering algorithm nearly recovered the three classes. This fact is also reflected by the high purity and low entropy values of the three clusters shown below.

Cluster#	Purity	Entropy
0	.996094	.0252647
1	.996398	.0233619
2	.982444	.0914646

5.1. Case Study on a Large Scientific Collection

To show the efficiency and effectiveness of our algorithms, we now present results on a large real-life collection of NSF award abstracts. We obtained the NSF data set by downloading abstracts of grants awarded by the National Science Foundation between Jan 1958 and August 1999 from www.nsf.gov. For our experiments we extracted titles and abstracts (ignoring fields such as 'Type', 'NSF Org', 'Date', etc.) of awards made in the 8 NSF organizations: Directorate for Biological Sciences (BIO), Directorate for Computer and Information Science and Engineering (CISE), Directorate for Education and Human Resources (EHR), Directorate for Engineering (ENG), Directorate for Geosciences (GEO), Directorate for Mathematical and Physical Sciences (MPS), Office of

Polar Programs (OPP) and Directorate for Social, Behavioral, and Economic Sciences (SBE). This results in a total of 113716 abstracts. The number of documents per NSF organization is as follows:

Class label	ENG	CSE	MPS	BIO	GEO	EHR	SBE	OPP
# documents	19365	8275	23037	16768	15667	14112	13725	2767

The most populous class is MPS with over 23000 awards, with ENG being second. On the other hand, OPP contains the fewest (2767) number of documents. Both the mean and median class size is about 142000.

We preprocessed this NSF collection by proceeding as in Section 3. After removing common stopwords, the collection contained 153527 unique words from which we eliminated 127103 low-frequency and high-frequency words. We were finally left with 26424 words using which we created 113716 document vectors using the tfn scheme. Each document vector has dimension 26424, however, on an average, each document vector contains only about 72 nonzero components and thus is more than 99% sparse.

In terms of the words contained in these documents, Table 1.1 shows the top ten most common words in the NSF data set. Note that we refer to the most frequently occurring word as having rank 1, the second most frequent word as having rank 2, and so on. As is common in most document collections the majority of the words occur in very few documents. Figure 1.5 shows the distribution of all the word frequencies versus their rank on a log-log scale. This distribution approximately fits the so-called *Zipf's law* [Zip49].

Rank	Word	Frequency
1	abstract	154467
2	research	152134
3	title	143257
4	project	73720
5	study	51178
6	data	43703
7	system	40314
8	university	39491
9	program	39203
10	science	38530

Table 1.1. Top 10 most common words in the NSF data set

As mentioned above, there are a total of 153527 unique word in the NSF collection. In general, the number of unique words W grows slower

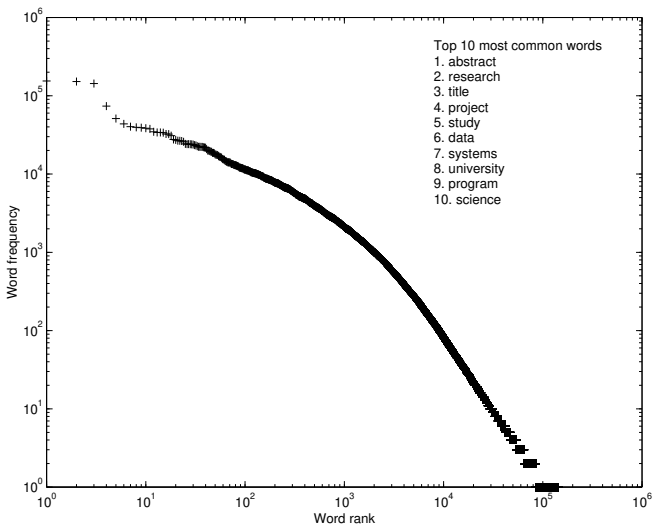

Figure 1.5. Distribution of word frequencies on a log-log scale (Zipf's Law)

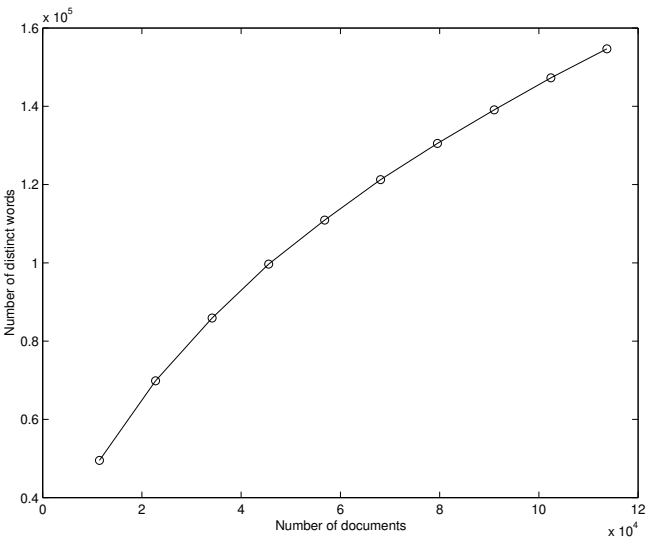

Figure 1.6. Vocabulary size vs. number of documents (Heap's Law)

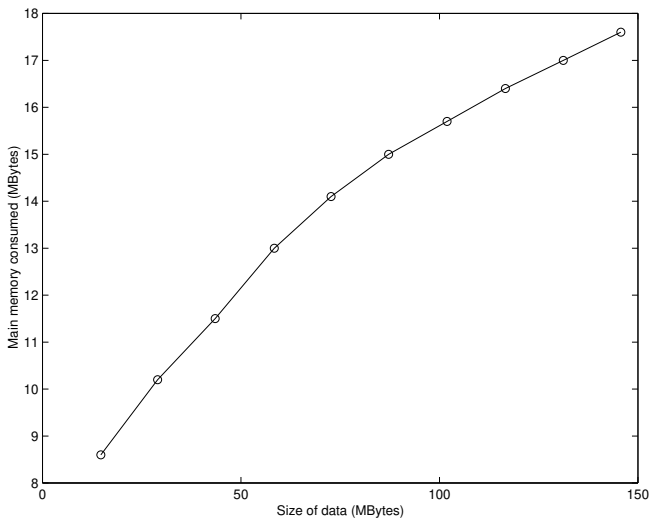

Figure 1.7. Main memory consumed vs. size of data

than linearly with the number of documents, i.e., $W = O(d^{\beta})$ where $0 < \beta < 1$. This behaviour known as *Heap's law* [Hea78] is seen in Figure 1.6.

5.1.1 Preprocessing Results. We now show the results of our clustering process on this large NSF collection. The preprocessing phase takes time that grows linearly with the size of the data set, while the main memory consumed also grows linearly with the number of unique words. Experiments on subsets of the NSF collection validate these claims, see Figures 1.7 and 1.8.

Figure 1.7 shows that we consume less than 18 MBytes of main memory to preprocess the entire NSF collection of 113716 documents. Extrapolating this number to a collection of one million documents, we see that 160 MBytes of main memory would be sufficient, which implies that our algorithms can carry out this large task on a single workstation with just 256 MBytes of main memory. From Figure 1.8 we see that the entire NSF data set is preprocessed in less than 20 minutes. Note that this includes disk I/O time. Our implementation is multi-threaded and hence the time taken is not very sensitive to the traffic over the Network File Server.

5.1.2 Clustering Results. In this section, we examine the speed and quality of the clustering algorithm. As discussed in Sec-

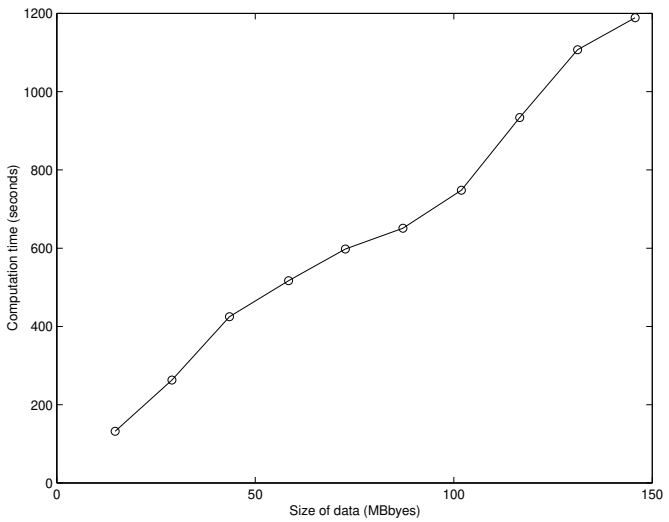

Figure 1.8. Computation time vs. size of data

tion 4.1, the spherical k-means algorithm forms k clusters in $O(nz \cdot k)$ time. Figure 1.9 gives the time taken to cluster the NSF data set (113716 documents, 26424 words, 8141987 non-zero entries) into a varied number of clusters. Times for both the original clustering algorithm and its modification that uses similarity estimation are shown (see Section 4.2). The similarity estimation technique yields considerable savings in computation time when the number of clusters is large. The running time of the algorithm is also seen to increase linearly with the number of clusters k. To form 100 clusters, 1400 seconds of computational time is needed. On the other hand, to form 12 clusters only 200 seconds are needed. Thus, combined with the preprocessing time of about 1190 seconds, the entire clustering process for 12 clusters is seen to take only about 23 minutes.

We now present a sample clustering obtained by the spherical k-means algorithm. We clustered the entire set of 113716 documents into 12 clusters. In the following table, we have listed the 10 dominant words and their weights in each of the concept vectors that correspond to the 12 clusters. Note that we have denoted each of the 12 clusters by the dominant word, e.g., 'theory', 'chemistry', 'physics', etc.

By examining the top 10 words it is easy to see the 'concept' contained in each cluster. For example, the 'conference' cluster containing 6652 documents appears to be about NSF awards that support conferences, meetings, workshops and symposiums for international scientists and researchers.

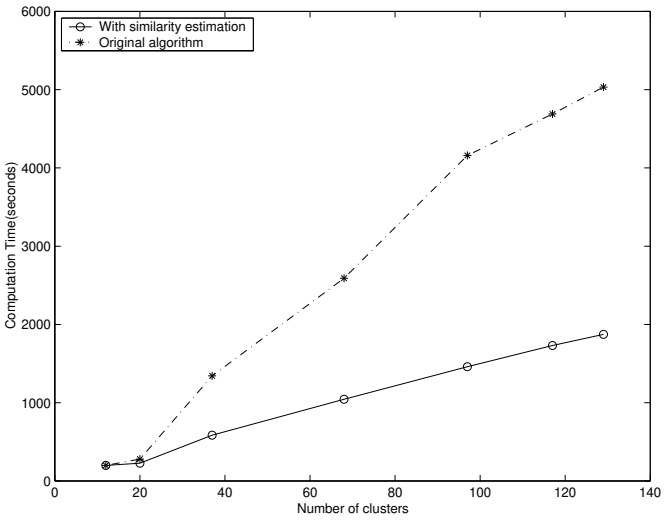

Figure 1.9. Time required to cluster the NSF collection

We now see the extent to which our clustering algorithm is able to recover the 8 NSF classes: BIO, CSE, EHR, ENG, GEO, MPS, OPP and SBE. The following gives the confusion matrix between the 12 clusters and 8 classes.

Note that the largest number in each row of the above table is boldfaced. We have ordered the clusters and classes so that the confusion matrix has large numbers near the diagonal. The fact that the numbers near the diagonal are larger than those away from the diagonals implies that many of the clusters can be identified with NSF organizations. For example, the 'theory' and 'chemistry' clusters can be identified with MPS, the 'social' cluster with SBE, the 'protein' and 'species' clusters with BIO and the 'ocean' cluster can be identified with GEO. Interestingly, some of the clusters indicate the multidisciplinary nature of some NSF organizations. The 'conference' cluster is seen to have many awards from MPS, ENG, SBE and BIO but no particular organization is dominant. Also, organizations such as MPS are seen to have subcategories — 'mathematical theory', 'chemistry reactions' and 'quantum physics'. The numbers in the confusion matrix also indicate the inter-relationships between various NSF orgainzations. For example, the 'materials' cluster indicates closeness between MPS and ENG while the 'design' cluster shows the closeness between ENG and CSE.

The quality measures of purity and entropy for this clustering are given in the following table.

theory(7442)	theory(.339),mathematical(.295),equations(.25),geometry(.192), problems(.186),sciences(.177),differential(.168) algebraic(.165),manifolds(.126),spaces(.114)
chemistry(7420)	chemistry(.448),reactions(.217),organic(.21),nmr(.194), molecules(.182),metal(.169),chemical(.151),compounds(.142), spectroscopy(.122),reaction(.12)
physics (9835)	physics(.189),quantum(.166),magnetic(.166),particle(.144), electron(.135),stars(.13),solar(.129),energy(.123), laser(.115),theoretical(.112)
materials(11589)	materials(.292),phase(.192),properties(.156),films(.14), polymer(.135),surface(.125),thin(.116),optical(.114), temperature(.11),liquid(.107)
design(13255)	design(.243),algorithms(.216),control(.18),parallel(.175), performance(.149),problems(.144),software(.118),models(.115), computer(.112),optimization(.1)
conference(6652)	conference(.47),workshop(.399),international(.23),held(.207), meeting(.181),symposium(.178),travel(.135),participants(.112), scientists(.107),researchers(.096)
mathematics(7374)	mathematics(.342),teachers(.333),school(.259),education(.177), college(.151),year(.150),teacher(.132),summer(.125), schools(.12)
laboratory(11302)	laboratory(.303),equipment(.241),undergraduate(.225), computer(.186),engineering(.185),courses(.175),student(.132), biology(.126),faculty(.124),projects(.124)
social(8267)	social(.27),economic(.180),political(.167),policy(.143), dissertation(.13),language(.126),public(.104),cultural(.103), decision(.1),market(.087)
protein(9212)	protein(.273),cell(.258),cells(.235),proteins(.22),gene(.196), genes(.179),dna(.153),expression(.138),molecular(.131), regulation(.119)
species(8822)	species(.326),plant(.167),populations(.151),plants(.133), population(.125),evolutionary(.125),evolution(.101), variation(.098),ecological(.096)
ocean(12546)	ocean(.232),ice(194),climate(.153),mantle(.147),sea(.134), water(.118),seismic(.106),circulation(.097),pacific(.096), isotopic(.092)

Table 1.2. Top 10 words associated with each cluster (the numbers in the right column give the word's weight in the concept vector)

Despite the above quality measures, it is sometimes best to have an informal way of examining a clustering. For this purpose, we have created a sequence of web pages to browse through the clusters, and the documents and keywords contained in them. The clustering presented above is available at http://www.cs.utexas.edu/users/dml/demos/nsfbrowser/.

	MPS	ENG	CSE	EHR	SBE	OPP	BIO	GEO
theory(7442)	**6502**	196	240	92	336	0	43	33
chemistry(7420)	**4747**	572	8	991	363	24	377	338
physics(9835)	**4901**	1649	31	146	849	147	176	1936
materials(11589)	3257	**7094**	33	171	541	12	89	392
design(13255)	952	4754	**5850**	357	840	5	349	148
conference(6652)	1149	**1767**	544	383	1176	146	947	540
mathematics(7374)	410	208	772	**5658**	143	21	65	97
laboratory(11302)	767	1344	541	**5658**	209	410	1036	1337
social(8267)	84	438	221	317	**6686**	126	326	69
protein(9212)	164	784	20	162	383	25	**7578**	96
species(8822)	61	153	7	121	1690	252	**5602**	936
ocean(12546)	43	406	8	56	509	1599	180	**9745**

Table 1.3. Confusion matrix between the 8 "true" classes and 12 clusters for the entire NSF collection

Cluster	Purity	Entropy
theory	.87369	.275307
chemistry	.639757	.585657
physics	.498322	.670075
materials	.612132	.499253
design	.441343	.643922
conference	.265634	.911829
mathematics	.76729	.429274
laboratory	.500619	.766223
social	.808758	.397713
protein	.822623	.347316
species	.635003	.535591
ocean	.776742	.389061

Table 1.4. Purity and entropy results for the computed clusters of the entire NSF collection

6. Conclusions and Future Work

In this paper, we have presented a time and memory efficient scheme for clustering very large document collections. We employ a multi-threaded approach to reading and preprocessing the documents to mitigate disk I/O costs, and use the effective spherical k-means algorithm to cluster the documents. Our experimental results show that we are able to preprocess and cluster $113,716$ NSF award abstracts in 23 minutes on a Sun workstation with modest memory consumption. We have also

demonstrated that the quality of the clustering is good. Based on the experiments we have done in this paper, we predict that we can use our scheme to cluster a million documents into 12 clusters in less than 4 hours on a Sun workstation.

In future work, we will continue our focus on improving the efficiency and scalability of our scheme. The dot product computation between all the document vectors and concept vectors is the computational bottleneck in the spherical k-means algorithm. To cut down on this computation, techniques like "truncation" which project each document vector onto a small subspace of the total word space will be investigated, see [SS97]. Meanwhile, more sophisticated handling of I/O threads will be studied in order to cut down the I/O cost which is the bottleneck for preprocessing. Parallelizing the whole process can be one of the approaches. Hierarchical clustering will also be studied in future work to discover the inherent hierarchical structure and the 'correct' number of clusters in the data set.

References

[BGG+98] D. Boley, M. Gini, R. Gross, E.-H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation on the World Wide Web using WebACE. *AI Review*, 1998.

[Cal99] Brent Callaghan. *NFS Illustrated.* Addison-Wesley, 1999.

[CKPT92] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *ACM SIGIR*, 1992.

[DGL89] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Trans Math Soft*, pages 1–14, 1989.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[DM01] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, January 2001. Also appears as IBM Research Report RJ 10147, July 1999.

[FBY92] W. B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[GJW82] M. R. Garey, D. S. Johnson, and H. S. Witsenhausen. The complexity of the generalized Lloyd-Max problem. *IEEE Trans. Inform. Theory*, 28(2):255–256, 1982.

[Hea78] J. Heaps. *Information Retrieval - Computational and Theoretical Aspects.* Academic Press, 1978.

[Kol97] T. G. Kolda. *Limited-Memory Matrix Methods with Applications.* PhD thesis, The Applied Mathematics Program, University of Maryland, College Park, Mayland, 1997.

[KPR98] Jon Kleinberg, C. H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4):311–324, December 1998.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[MS96] D. Musser and A. Saini. *STL Tutorial and Reference Guide.* Addison-Wesley, 1996.

[NBF96] Bradford Nichols, Bick Buttlar, and Jackie Proulx Farrell. *Pthreads Programming.* O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1996.

[Pax96] Vern Paxson. Flex user manual, November 1996.

[Ras92] E. Rasmussen. Clustering algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 419–442. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[SB88] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 4(5):513–523, 1988.

[SGM00] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proceedings of the AAAI2000 Workshop on Artificial Intelligence for Web Search*, pages 58–64, Austin, Texas, July 2000. AAAI/MIT Press.

[SM83] G. Salton and M. J. McGill. *Introduction to Modern Retrieval.* McGraw-Hill Book Company, 1983.

[SS97] H. Schütze and C. Silverstein. Projections for efficient document clustering. In *ACM SIGIR*, 1997.

[Wil88] P. Willet. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[ZE98] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *ACM SIGIR*, 1998.

[Zip49] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison Wesley, Reading, MA, 1949.