2007

High Performance Text Document Clustering

Yanjun Li
Wright State University

HIGH PERFORMANCE TEXT DOCUMENT CLUSTERING

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

Yanjun Li
M.S., Wright State University, 2003
B.S., Franklin University, 2001
B.S., University of International Business and Economics, China, 1993

2007
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

May 30, 2007

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Yanjun Li ENTITLED High Performance Text Document Clustering BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

Soon M. Chung, Ph.D.
Dissertation Director

Thomas Sudkamp, Ph.D.
Director, Computer Science and Engineering
Ph.D. Program

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on
Final Examination

Soon M. Chung, Ph.D.

Arthur Goshtasby, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Kefu Xue, Ph.D.

Michael L. Talbert, Ph.D.

Kevin Kirby, Ph.D.

ABSTRACT

Li, Yanjun. Ph.D., Department of Computer Science and Engineering, Wright State University, 2007. High Performance Text Document Clustering.

Data mining, also known as knowledge discovery in database (KDD), is the process to discover interesting unknown knowledge from a large amount of data. Text mining is to apply data mining techniques to extract interesting and nontrivial information and knowledge from unstructured text. Text clustering is one of important techniques of text mining, which is the unsupervised classification of similar documents into different groups.

This research focuses on improving the performance of text clustering. We investigated the text clustering algorithms in four aspects: document representation, documents closeness measurement, high dimension reduction and parallelization. We propose a group of high performance text clustering algorithms, which target the unique characteristics of unstructured text database.

First, two new text clustering algorithms are proposed. Unlike the vector space model, which treats document as a bag of words, we use a document representation which keeps the sequential relationship between words in the documents. In these two algorithms, the dimension of the database is reduced by considering the frequent word (meaning) sequences, and the closeness of two documents is measured based on the sharing of frequent word (meaning) sequences.

Second, a text clustering algorithm with feature selection is proposed. This algorithm gradually reduces the high dimension of database by performing feature selection during the clustering. The new feature selection method applied is based on the well-known χ^2 statistic and a new statistical data which can measure the positive and negative term-category dependence.

Third, a group of new text clustering algorithms is developed based on the k-means algorithm. Instead of using the *cosine* function, a new function involving global information

is proposed to measure the closeness between two documents. This new function utilizes the *neighbor matrix* introduced in [26]. A new method for selecting initial centroids and a new heuristic function for selecting a cluster to split are adopted in the proposed algorithms.

Last, a new parallel algorithm for bisecting k-means is proposed for the message-passing multiprocessor systems. This new algorithm, named PBKP, fully utilizes the data-parallelism of the bisecting k-means algorithm, and adopts a *prediction* step to balance the workloads of multiple processors to achieve a high speedup.

Comprehensive performance studies were conducted on all the proposed algorithms. In order to evaluate the performance of these algorithms, we compared them with existing text clustering algorithms, such as k-means, bisecting k-means [59] and FIHC [21]. The experimental results show that our clustering algorithms are scalable and have much better clustering accuracy than existing algorithms. For the parallel PBKP algorithm, we tested it on a 9-node Linux cluster system and analyzed its performance. The experimental results suggest that the speedup of PBKP is linear with the number of processors and data points. Moreover, PBKP scales up better than the parallel k-means with respect to the desired number of clusters.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Text Clustering

Data mining can be defined as the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [18]. Generally, data mining is the process of analyzing data from different perspectives and summarizing it into useful information. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large databases. Data could be any facts, numbers, or text that can be processed by a computer. Text Mining is the discovery of new, previously unknown information by automatically extracting information from text documents.

Data clustering is one of important techniques of data mining, which is the unsupervised classification of similar data objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset share some common trait according to some defined distance measure. Text clustering is the organization of a collection of text documents into clusters based on similarity. Intuitively, documents within a valid cluster are more similar to each other than those belonging to a different cluster. In other words, documents in one cluster share similar topics.

It is important to understand the difference between clustering and classification. In classification, we are provided with a collection of pre-classified training data, and the problem is to label a newly encountered unlabeled data. Typically, the given labled data (training data) are used to learn the descriptions of classes, which in turn are used to label new data. In the

case of clustering, the problem is to group a given collection of unlabeled data into meaningful clusters. Since clustering needs little prior information about the data than classification, it could be applied in many situations that classification could not be performed.

Typical text clustering activity involves the following steps [30]:

- document representation (optionally including feature extraction and/or selection).
- definition of a document similarity measure.
- clustering or grouping.

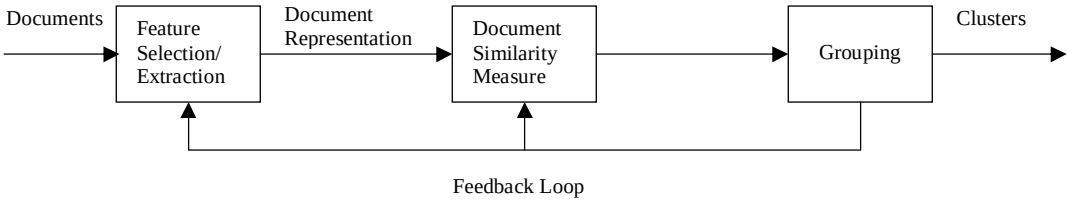


Figure 1.1: Stages in Text Clustering

Figure 1.1 shows a typical sequencing of these three steps, including a feedback path where the grouping process output could affect subsequent feature selection/extraction and similarity computations [31]. Document representation refers to the number of clusters, the number of documents, and the number, type and scale of the features available to the clustering algorithm. Feature selection is the process of identifying the most effective subset of the original features to use in clustering [30]. Feature extraction is the use of one or more transformations of the input features to produce new salient features [30]. Either or both of these techniques can be used to obtain an appropriate set of features to use in clustering. Document similarity is usually measured by a pair-wise similarity function. A simple similarity measure, like *cosine* function, is often used to reflect the similarity between two documents. The grouping step of text clustering can be performed in a number of ways. Hierarchical clustering algorithms produce a nested series of partitions based on a criterion for merging or splitting clusters based on similarity. Partitional clustering algorithms identify the partitions that optimizes a clustering criterion. The performance of text clustering

algorithm could be evaluated by the cluster validity analysis, which is the assessment of a clustering procedure's output. There are three types of validation studies. An external assessment of validity compares the recovered structure to a priori structure. An internal examination of validity tries to determine if the structure is intrinsically appropriate for the data. A relative test compares two structures and measures their relative merit.

1.2 Motivation

Since text documents are written in natural language, which is different from structured data, existing data clustering algorithms do not perform very well on text documents. When choosing a text clustering algorithm, there are many critical questions to ask, such as: How should the data be represented? Which similarity measure is appropriate for text clustering? How should domain knowledge be utilized in text clustering? How can a very large data set be clustered efficiently? In order to answer these questions, first, let's look at closely the special requirements for text clustering.

- Finding a suitable model to represent the document is a nontrivial issue. Most of the text documents are written in a human language, which is context-sensitive. As the accurate meaning of a sentence has a close relationship with the sequential occurrences of words in it, the document model better preserves the sequential relationship between words in the document.

- In the vector space model, the length of document vectors are normalized. However, in real life, the sizes of vocabularies for different topics are different from each other.

- In the real world, people may use different word forms to express the same word meaning, and the same word form to express different word meanings. In text clustering, if only word forms are used as features, the topics of documents may not be fully captured. Word meanings are better than word forms in terms of representing the topics of documents. Thus, it is beneficial to involve ontology into the text clustering algorithm.

- Not all the words in a document are closely related with the topic of the document. Documents without these irrelevant words will certainly provide valuable information for clustering. How to identify and remove them is a nontrivial problem.

- Associating a meaningful label to each final cluster is essential. Then, the user can easily find out what the cluster is about since the label can provide an adequate description of the cluster. However, it is time-consuming to determine the labels after the clustering process is finished.

- Overlapping between document clusters should be allowed because a document can cover multiple topics. For example, a morning news may have information about a war followed by information about the popularity of teas in US.

- The high dimension of text documents should be reduced. Usually there are about 200-1000 unique words in a typical document. In order to efficiently process a huge text database like WWW, the text clustering algorithm should have a way to reduce the high dimension.

- The number of clusters is unknown prior to the clustering. It is difficult to specify a reasonable number of clusters for a data set when you have little information about it. Instead of telling the clustering algorithm what is the number of clusters, it makes more sense to let the clustering algorithm find it out by itself.

These issues have motivated our research, and its goal is to show a new perspective in text clustering methodology and algorithms. A group of text clustering algorithms proposed target the unique characteristics of text documents and deliver better clustering results than existing algorithms.

1.3 Contributions

In this research, we develop a group of text clustering clustering algorithms with high performance, and the main contributions of our research are as follows:

- Novel text clustering algorithms based on frequent word (meaning) sequences are proposed and implemented. The unique features of these algorithms are : 1) A new frequent word sequence mining algorithm is developed, which is quite scalable as the database size increases. 2) A new document representation is proposed, which keeps the sequential relationship between words in documents. 3) The high dimension of text database is reduced in the clustering process since only the frequent word (meaning) sequences are considered. 4) The document closeness is measured based on the shared frequent word(meaning) sequences. 5) Ontology is applied in the text clustering algorithm to use the word meanings.

- A new text clustering algorithm with feature selection is proposed. The clustering quality is expected to be improved by performing feature selection and clustering at the same time. Our contributions in this research are: 1) Based on χ^2, a new statistic data $R_{w,c}$ is proposed. This data could specify whether the term-category dependency is positive or negative. 2) A new supervised feature selection method CHIR is proposed. This method evaluates χ^2 and $R_{w,c}$ together to provide a more accurate rank of features. 3) A new text clustering algorithm is proposed to perform a supervised feature selection method in an unsupervised scenario. At the same time, the high dimension of text database is reduced by performing the feature selection.

- A group of text clustering algorithms are developed to apply the *neighbor matrix* to the family of k-means algorithms. The new features of these algorithms are: 1) A new initial centroids selection method is proposed. The desired initial centroids should be well-distributed to attract sufficient nearby documents. 2) The document closeness measurement is enhanced from the *cosine* function. This new function involves global information into the pair-wise similarity measurement. 3) A new heuristic function is proposed for selecting a cluster to split for the bisecting k-means algorithm. This method provides a new way to evaluate the quality of a cluster.

- A new parallel bisecting k-means algorithm is proposed and implemented. This al-

gorithm fully utilizes the data-parallelism of the bisecting k-means algorithm, and adopts a prediction step to balance the workloads of multiple processors to achieve a high speedup.

1.4 Outline of the Dissertation

The rest of the dissertation is structured as follows: Chapter 2 describes the CFWS and CFWMS text clustering algorithms, which are based on frequent word (meaning) sequences. Then, the performance of these two algorithms are discussed. Chapter 3 introduces a new statistical data $R_{w,c}$. Then, a new feature selection method CHIR and a new clustering algorithm with feature selection (TCFS) are described. Chapter 4 introduces a group of new text clustering algorithms which apply the *neighbor matrix* in the k-means and bisecting k-means algorithms. Chapter 5 describes the new parallel bisecting k-means algorithm PBKP for message-passing multiprocessor systems. This research is concluded in Chapter 6.

Chapter 2

Text Clustering Based on Frequent Sequences

2.1 Introduction

Nowadays in every industry, almost all the documents on paper have their electronic copies. This is because the electronic format provides safer storage and occupies much smaller space. Also, the electronic files provide a quick access to these documents. The text database which consists of documents is usually very large. The Word Wide Web is such a database, and how to explore and utilize this kind of text database is a major question in the areas of information retrieval and text mining. With the development of the World Wide Web, it is getting more and more popular to use web search engines to get information. When a user submits a query, the search result is usually a long list of ranked documents. The users may not find what they want from the top 10 documents on the list. It is time-consuming and annoying to browse the result documents one by one. Thus, when the users cannot find matching one after 10-20 clicks, they may give up. This is why the precision of the retrieval for a given query is an important for the search engine.

In order to increase the precision of the retrieval result, many methods have been proposed [3]. One approach is clustering the retrieval result before showing it to the user. The idea behind it is that the retrieval result usually covers several topics and the user may be interested in just one of them. By clustering the text documents, the documents sharing

the same topic are grouped together. When the clusters are returned, the user can select the group that interests him/her most. This method makes the search engine more efficient and accurate. Text clustering is known as an unsupervised and automatic grouping of text documents into clusters, so that documents within a cluster have a high similarity between them, but they are dissimilar to documents in other clusters [22]. It is different from text classification because of the lack of labeled documents for training.

The main question is which text clustering algorithm is the best for this job. First, let's look at closely the special requirements for the clustering of the text retrieval result.

- Finding a suitable model to represent the document is a nontrivial issue. Most of the text documents are written in a natural language, which is context-sensitive. As the accurate meaning of a sentence has close relationship with the sequential occurrences of words in it, the document model better preserves the sequential relationship between words in the document.

- Associating a meaningful label to each final cluster is essential. Then, the user can easily find out what the cluster is about since the label can provide an adequate description of the cluster. However, it is time-consuming to determine the labels after the clustering process is finished.

- Overlapping between document clusters should be allowed because a document can cover several topics. For example, a morning news may have an information about a war followed by an information about the popularity of teas in US.

- The high dimension of text documents should be reduced. Usually there are about 200-1000 unique words in a typical document. In order to efficiently process a huge text database like WWW, the text clustering algorithm should have a way to reduce the high dimension.

- The number of clusters is unknown prior to the clustering. It is difficult to specify a reasonable number of clusters for a data set when you have little information about

it. Instead of telling the clustering algorithm what is the number of clusters, it makes more sense to let the clustering algorithm find it out by itself.

Our two new text clustering algorithms, named Clustering based on Frequent Word Sequences (CFWS) and Clustering based on Frequent Word Meaning Sequences (CFWMS) are designed to meet the above special requirements of text clustering to varying degrees. A "word" is the word form shown in the documents. A "word meaning" is the lexicalized concept that a word form can be used to express [66]. The key features of these two algorithms are: they treat the text document as a sequence of words (word meanings), instead of a bag of words, and whether documents share frequent word (meaning) sequences or not is used as the measurement of their closeness.

A *frequent word* (*meaning*) *sequence* is defined as a sequence of frequent words (word meanings) appearing in at least a certain number (or percentage) of documents, and we developed an algorithm to find the frequent word (meaning) sequences from a text database.

Finding frequent itemsets is an important data ming topic, and it was originated from the association rule mining of transaction data set. Recently, some text clustering algorithms used frequent word sets to compare the distance between documents. Considering the difference between text documents and transaction data set, using the frequent word sequences is more appropriate for text clustering than using the frequent word sets.

Since the order of words in a document is important, we didn't adopt the vector space model. In our algorithm, each document is reduced to a compact document by keeping only the frequent words (word meanings). In the compact document, we keep the sequential occurrence of words (word meanings) untouched to explore the frequent word (meaning) sequences. By building a Generalized Suffix Tree (GST) for all the compact documents, the frequent word (meaning) sequences and the documents sharing them are found. Then, frequent word (meaning) sequences are used to create clusters and summarize their content for the user. We found our CFWS and CFWMS algorithms are more accurate than other clustering algorithms.

The rest of this chapter is organized as follows: Section 2.2 introduces the related work on

text clustering and the mining of frequent word sequences. Section 2.3 describes the method to find frequent word sequences from a text database in detail, and Section 2.4 describes the new CFWS clustering algorithm. The experimental results of CFWS and the performance comparison with FIHC [21] and bisecting k-means [59] are presented in Section 2.5. Section 2.6 contains some conclusions.

2.2 Related Work

There are two general categories of clustering methods: agglomerative hierarchical and partitioning methods. In the previous researches, both of them are applied to text clustering. Agglomerative hierarchical clustering (AHC) algorithms initially treat each document as a cluster, use different kinds of distance functions to compute the similarity between all pairs of clusters, and then merge the closest pair [30]. This merging step is repeated until the desired number of clusters is obtained. Overlapping of clusters is not allowed in AHC. Comparing with the bottom-up method of AHC algorithms, the family of k-means algorithms [13, 33, 36], which belong to the partitioning category, adopted the top-down method. Initially, the whole database is treated as a cluster. Based on a heuristic function, it selects a cluster to split. The split step is repeated until the desired number of clusters is obtained. These two categories are compared in [59].

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [30] of AHC is reported to be the most accurate one in its category. Bisecting k-means is reported to outperform the k-means as well as the agglomerative approach in terms of accuracy and efficiency. The difference between bisecting k-means and k-means is that bisecting k-means splits a selected cluster into two subclusters, instead of k subclusters. Since these two categories of clustering algorithms are originally designed to cluster formatted data sets, the special characteristics of text databases are not taken care of well. First, there is no description given to each cluster. Each text document cluster should have a description about its content, so that the clustering result can be utilized more efficiently. Second, the number of unique words in a document may be in the order of several hundreds or more, but these algorithms

do not have steps to reduce the high dimension of the text documents during the clustering. Third, the user must specify the desired number of clusters before the clustering process. However, it is difficult to specify a reasonable number of clusters before you closely study the text database. We may run the algorithm several times with different number of clusters to choose a appropriate one, but this is too time-consuming.

Recently, there is a new category of text clustering algorithms developed. They address the special characteristics of text documents and use the concept of frequent word sets for the text clustering. In [63], they proposed a new criterion for clustering transactions using frequent itemsets, instead of using a distance function. The FTC algorithm introduced in [5] used the shared frequent word sets between documents to measure their closeness in text clustering. The FIHC algorithm proposed in [21] went further in this direction. It measures the cohesiveness of a cluster directly by using frequent word sets, such that the documents in the same cluster are expected to share more frequent word sets than those in different clusters. FIHC uses frequent word sets to construct clusters and organize them into a topic hierarchy. These new algorithms are reported to be comparable to bisecting k-means in terms of clustering accuracy. An advantage of these algorithms is that a label is provided for each cluster. The label is the frequent word sets shared by the documents in each cluster. A problem of these algorithms is that they strongly depend on the frequent word sets, which are unordered and cannot represent text documents well in many cases.

The concept of the frequent word set is based on the frequent itemset of the transaction data set. The items in a transaction are independent, so that changing the order of these items in a transaction does not change the result of the data mining performed on the database. But the text document is different. The sequential order of words in a document plays an important part of delivering the meaning of the document. The change of the relative positions of two words may change the content of a document. For example, "association rule" is a concept of data mining. If these two words, "association" and "rule", appear in the reverse order within a document, like "The rule of our association is ...", they represent a totally different meaning. If only the word set {association, rule} is used, it

cannot differentiate these two cases.

Since all the clustering algorithms mentioned above treat each document as a bag of words, they use the vector space model to represent a document after the preprocessing steps, such as the removal of the stop words and the stemming of words. In this model, each text document is represented by a vector of the frequencies of the remaining terms. The information about the position of the words in the text documents is not stored in this model, thus it is not good enough for text documents. In this chapter, we propose a new model to represent the document. The sequential relationship between words in the document is preserved in the model and utilized for the text mining.

Since frequent word sequences can represent the document well, clustering text documents based on frequent word sequences is meaningful. The idea of using word sequences (phrases) for text clustering was proposed in [69]; and then the Suffix Tree Clustering (STC), which is based on this idea, was proposed in [70]. STC does not treat a document as a set of words but rather as a string, in order to use the proximity information between words. STC builds a suffix tree to identify sets of documents that share common phrases, and uses this information to create clusters and summarize their contents.

However, STC does not reduce the high dimension of the text documents, hence its complexity is quite high for large text databases. On the other hand, our CFWS algorithm uses only the frequent word sequences, not all the phrases in the documents, hence the dimension of the documents is reduced dramatically. Moreover, a phrase is meaningful to the clustering result only when it is shared by at least a certain number of documents. For example, suppose that there are only two documents, say A and B, sharing a phrase "Banana Republic"(a fashion brand name) in a document collection, and all other documents in the collection do not have this phrase, while 20 documents, including A and B, have a phrase "fashion trend". It is obvious that a cluster about "fashion trend" is more desirable than a cluster about "Banana Republic" in the final clustering. From this example, we can see that phrases supported by a very small number of documents play a little role in the final clustering result. Thus, we remove these infrequent phrases at the early stage of the process,

so that the dimension of the documents is reduced and the clustering result would not be affected by them.

Ahonen-Myka et al. also pointed out in [46, 47, 48] that the sequential aspect of word occurrences in documents should not be ignored to improve the information retrieval performance. They proposed to use the maximal frequent word sequence, which is a frequent word sequence not contained in any longer frequent word sequence. They claimed that maximal frequent word sequences provide a rich computational representation of the document, which makes future retrieval easy and gives a human-readable description of the document. However, all frequent word sequences in a text database are as important as the maximal frequent word sequences. Frequent word sequences of all length contain more information about the database than the maximal frequent word sequences.

2.3 Finding Frequent Word Sequences

2.3.1 Term Definitions

In our algorithm, a text document d is viewed as a sequence of words, so that it can be represented as $d =< w_1, w_2, w_3, \ldots >$, where $w_1, w_2, w_3, \ldots$ are words appearing in d. Like a frequent itemset in the association rule mining of a transaction data set [1], a word set is frequent when its support is at least the user-specified minimum support. That means, there are at least the specified minimum number (or percentage) of documents containing this word set. A *frequent k-word set* is a frequent word set containing k words.

An ordered sequence of two or more words is called a word sequence. A word sequence S is represented as $< w_1, w_2, \ldots >$. A frequent word sequence is denoted by FS in this chapter. For example, $FS =< w_1, w_2, w_3, w_4 >$, in which w_2 is not necessarily following w_1 immediately in a text document. There could be words between them as long as w_2 is after w_1 and the words between them are not frequent. A text document d supports this word sequence if these four words (w_1, w_2, w_3, and w_4) appear in d in the specified order. A word sequence S is an FS when there are at least the specified minimum number (or

percentage) of documents supporting S. Multiple occurrences of a sequence in the same document is counted as one. The term *phrase* is defined in [70] as an ordered sequence of one or more words, and no gaps are allowed between words. Thus, our definition of *frequent word sequence* is more adaptable to the variations of human languages. For example, "boys play basketball" may be a frequent word sequence supported by both of the following two sentences:

- Young **boys** like to **play basketball**.
- Almost all **boys play basketball**.

A *frequent k-word sequence* is an FS with length k, such as $FS =< w_1, w_2, \ldots, w_k >$, and it has two frequent subsequences of length $k-1$, which are $< w_1, w_2, \ldots, w_{k-1} >$ and $< w_2, w_3, \ldots, w_k >$. For example, $FS =< he,\ play,\ basketball >$ has two frequent subsequences of length 2, which are $< he,\ play >$ and $< play,\ basketball >$. Please note that $< he,\ basketball >$ is not its subsequence of length 2 because "play" is a frequent word in the database and we cannot omit it.

Theorem 1: It is always true that if a word w_i is a member of a frequent k-word sequence, it must be a member of a frequent k-word set. But a member of a frequent k-word set is not necessarily a member of a frequent k-word sequence, where the order of the k words matters. This is very straightforward.

Theorem 2: If a k-word sequence is frequent, all its subsequences of length $k-1$ are frequent. It is easy to prove from the definition of the subsequence.

2.3.2 Algorithm Details

Finding the frequent word sequences has two steps: finding frequent 2-word sets first, then finding frequent word sequences of all length by using the Generalized Suffix Tree (GST) data structure.

2.3.2.1 Finding Frequent 2-Word Sets

The goal of this step is to reduce the dimension of the database (i.e., the number of unique words) by eliminating those words that are not frequent enough to be in a frequent k-word sequence, for $k \geq 2$. This step is simple and straightforward. We use an association rule miner to find the frequent 2-word sets that satisfy the minimum support. All the words in frequent 2-word sets are put into a set *WS*. Based on Theorems 1 and 2, we know that members of the frequent word sequences of all length k, $k \geq 2$, must in the set *WS*.

After finding the frequent 2-word sets, we remove all the words in the documents that are not in *WS*. After the elimination, the resulting documents are called *compact documents*.

Example database: $D = \{d_1, d_2, d_3\}$

- d_1: Young boys like to play basketball.
- d_2: Half of young boys play football.
- d_3: Almost all boys play basketball.

There are 11 unique words in this database D: {all, almost, basketball, boys, football, half, like, of, play, to, young}. If we specify the minimum support as 60%, the minimum support count is 2 for this case. The set of frequent 2-word sets is {{young, boys}, {boys, play}, {boys, basketball}, {young, play}, {play, basketball}}; and WS = {young, boys, play, basketball}. After removing those words not frequent, the database D becomes $D' = \{d'_1, d'_2, d'_3\}$ as follows, where the removed words shown in parenthesis.

Compact documents:

- d'_1: Young boys (*like to*) play basketball;
- d'_2: (*Half of*) young boys play (*football*);
- d'_3: (*Almost all*) boys play basketball;

From this example, we can see that the dimension of D is reduced from 11 to 4. This reduction has a big impact on our next step of building the generalized suffix tree for D'.

2.3.2.2 Building a Generalized Suffix Tree (GST)

Our goal is to find the frequent word sequences of the database. We adopt the suffix tree, a well-known data structure for sequence pattern matching [64], to find the all frequent word sequences. Each compact document is treated as a string of words and inserted into a generalized suffix tree (GST), one by one. Finally, by collecting the information stored in all the nodes of this GST, we can find all the frequent word sequences of this database.

A suffix tree for a string S is actually a compressed trie for the non-empty suffixes of S. A GST is a suffix tree that combines the suffixes of a set of strings. In our case, we build a GST of all the compact documents in the text database, so some modification are made on the structure of the GST to meet our needs. In the rest of this chapter, we will use the terms "suffix tree" and "GST" interchangeably.

- A suffix tree is a rooted, directed tree.
- There are two kinds of nodes: internal nodes and suffix nodes.
- Each internal node has at least two children.
- Each edge is labeled with an non-empty substring of string S, and is represented as l. The label of a node n is the concatenation of the labels on the path from the root to this node. This label is represented as $string_L$ of n, or simply $n.string_L$.
- The labels of different edges coming from the same node must have different starting words.
- For each suffix s of string S, there exists a suffix node whose label is s.
- There is a document id set associated with each suffix node. If a substring of a document ends at a node, then the document id is inserted into the id set of the node. These document ids are used to check the multiple occurrences of a sequence in the same document.

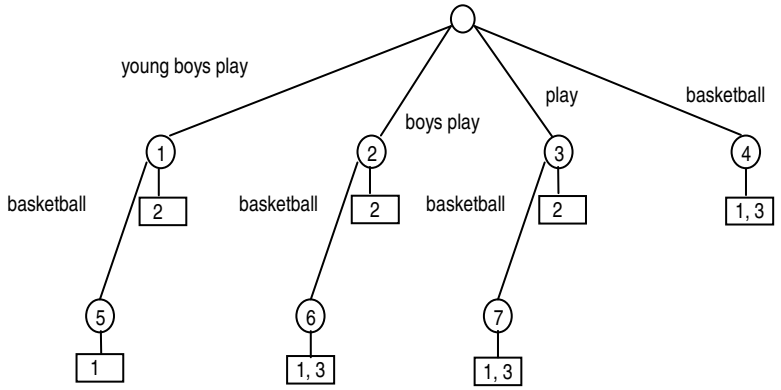


Figure 2.1: Generalized Suffix Tree for the compact documents with 7 nodes.

Node No.	Word Sequence	Length of Word Sequence	Document Ids	Number of Document Ids
1	young boys play	3	1,2	2
2	boys play	2	1,2,3	3
3	play	1	1,2,3	3
4	basketball	1	1,3	2
5	young boys play basketball	4	1	1
6	boys play basketball	3	1,3	2
7	play basketball	2	1,3	2

Table 2.1: Word sequences associated with the nodes in Figure 2.1

Figure 2.1 shows the GST built for the previous example. The nodes of the GST are drawn as circles, each with an assigned node number in it for later references. Each suffix node has a box attached, and it contains the document id set of the suffix node. After building the GST, we traverse the tree by depth-first. On the way down, the labels of the edges are concatenated to become the $string_L$ of each node. On the way up, each child node sends its document id set to its parent. The support count of $string_L$ of this parent node is the size of the union of all the document id sets of its children. By checking the support count and the length of $string_L$ of each node, we can get the information about all the frequent word sequences of the database. In the example shown above, we have 7 nodes in the GST, and the details are given in Table 2.1.

Since the minimum support for frequent words, θ, is set to 60% in this example, the

minimum support for frequent word sequences could not be smaller than 60%. Only those words whose support is at least θ are kept in the compact documents, so that we can find only the frequent word sequences with that minimum support θ. In this example, the minimum length of the word sequence is set to 2, so we can get four frequent word sequences, which are represented by nodes 1, 2, 6, and 7. The maximal frequent word sequences of this database are represented by nodes 1 and 6.

The word sequence of node 2 is a subsequence of the word sequence of node 1, but its id set is a superset of that of node 1. If we find only maximal frequent word sequences, some of the information would be lost. As mentioned in [46], maximal frequent word sequences can be used as content descriptors for documents. However, if we want to summarize the content of this example database, a frequent word sequence "boys play" is the best description. A maximal frequent word sequence "young boys play" covers only the content of first two documents, d_1 and d_2, and another maximal frequent word sequence "boys play basketball" covers only the content of documents d_1 and d_3.

By finding all the frequent word sequences, there may be some duplicate information found, like the sequences of nodes 6 and 7 in this example. If it is necessary to save the space, by comparing their document id sets and word sequence members, we can combine these two nodes into one without losing information. As illustrated by this example, by finding all frequent word sequences, we can have some useful information about the database for further information retrieval and data mining operations.

In [70], the Suffix Tree Clustering (STC) algorithm was proposed, which clusters text documents by constructing the suffix tree of all the sentences of the documents in the collection, and Figure 2.2 shows the suffix tree for the example database D. Comparing this suffix tree of the STC algorithm with our suffix tree shown in Figure 2.1, we can see that the size of the tree is dramatically reduced due to the elimination of infrequent words. As the number of unique words and the size of the compact database are much smaller than those of the original database, our algorithm is clearly more efficient than STC.

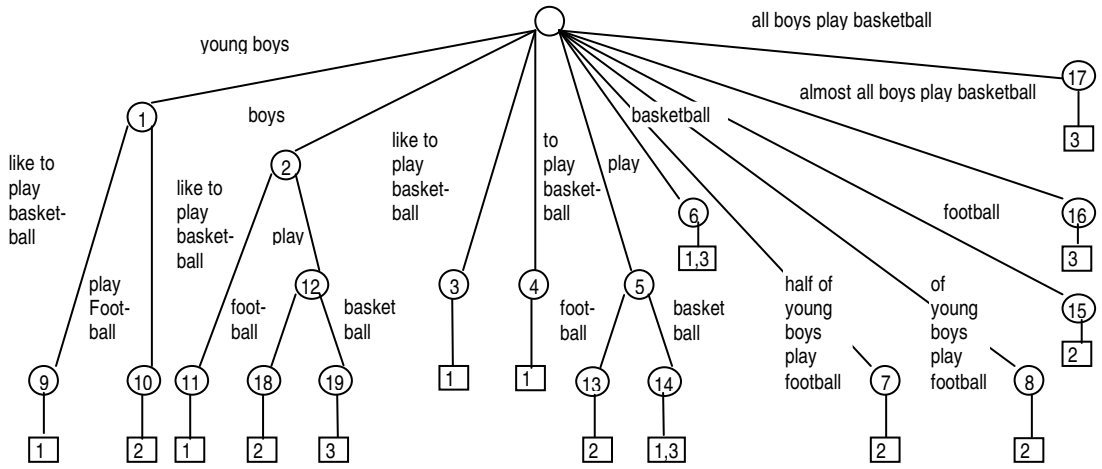


Figure 2.2: Suffix Tree for the original documents with 16 nodes.

2.4 Clustering Based on Frequent Word Sequences

2.4.1 Term Definitions

A *cluster candidate* is a set of text documents supporting the same frequent word sequence (FS). A cluster candidate i is represented by $cc_i[FS_i, Ids_i]$, in which FS_i is the frequent word sequence supported by this cluster candidate and Ids_i is the set of document ids. A *cluster* is a set of text documents covering the same topic. A cluster i is represented as $C_i[T_i, Ids_i]$, where T_i is composed of a group of frequent word sequences and Ids_i is the set of document ids that cover the topic T_i. Thus, a cluster candidate can be considered as a cluster whose topic contains only one frequent word sequence.

The *final clustering* C is a set of clusters $\{C_1, C_2, \ldots\}$. Since we allow overlapping between clusters, the intersection of two clusters is not necessarily empty.

2.4.2 Clustering Based on Frequent Word Sequences (CFWS) Algorithm

Our CFWS algorithm has two steps: building a GST to find frequent word sequences, then combining clustering candidates to obtain the final clustering.

2.4.2.1 Finding Frequent Word Sequences and Collecting the Cluster Candidates

We use the method explained in Section 2.3 to build a GST for the database. The minimum support of the frequent word sequences is usually in the range of 5–15%. When the minimum support is too large, the total number of frequent words would be very small, so that the resulting compact documents would not have enough information about the original data set. In this case, a lot of documents will not be processed because they do not support any frequent word, and the final clustering result will not cover these documents.

Since the document id sets are stored at the suffix nodes of the GST, we can use these sets directly to obtain the cluster candidates. Only the nodes of the tree representing frequent word sequences can produce the cluster candidates. As shown in Table 2.1, we can obtain four cluster candidates for our example database: $cc_1[FS_1 =$ "boys play", $Ids_1 = \{1, 2, 3\}]$, $cc_2[FS_2 =$ "play basketball", $Ids_2 = \{1, 3\}]$, $cc_3[FS_3 =$ "young boys play", $Ids_3 = \{1, 2\}]$, and $cc_4[FS_4 =$ "boys play basketball", $Ids_4 = \{1, 3\}]$.

2.4.2.2 Combining the Cluster Candidates

The FS of the cluster candidate describes what the cluster candidate is about. For example, from $FS_1 =$"boys play", we know that cc_1 covers the documents regarding which game the boys play. However, sometimes we do not need fine clusters, instead we may be interested in a more general topic, such as the popular sports among teenagers. In that case, we can merge the related cluster candidates. The agglomerative clustering algorithm also merges two clusters closely related to each other. However, instead of using a distance function to measure the closeness between two cluster candidates, we use the k-mismatch concept of sequential patterns.

Given a pattern p, a text t, and a fixed number k that is independent of the lengths of p and t, a k-mismatch of p is a $|p|$-substring of t that matches $(|p| - k)$ characters of p. That is, it matches p with k mismatches. In our case, we are checking the mismatches between the frequent word sequences found by building the GST of the document collection. There are

three types of mismatches that can happen between two frequent word sequences FS_i and FS_j: *insertion*, *deletion* and *substitution*. Insertion means that by inserting k words into the shorter pattern FS_i, it becomes the longer pattern FS_j. Deletion means that by deleting k words from the longer pattern FS_j, it becomes the shorter pattern FS_i. Substitution is the relationship between two patterns, FS_i and FS_j, of the same length, such that by substituting k words in FS_i, it becomes FS_j. The following are the examples of three cases when $k = 1$:

- Insertion: $\{|FS_i| < |FS_j| \mid FS_i =$ "boys play"; $FS_j =$ "boys play basketball"$\}$;
  By inserting a word "basketball" in $FS_i \Rightarrow FS_i = FS_j$;
- Deletion: $\{|FS_j| > |FS_i| \mid FS_j =$ "boys play"; $FS_i =$ "play"$\}$;
  By deleting a word "boys" from $FS_j \Rightarrow FS_j = FS_i$;
- Substitution: $\{|FS_i| = |FS_j| \mid FS_i =$"boys play"; $FS_j =$ "girls play"$\}$;
  By substituting a word "boys" of FS_i with a word "girls"$\Rightarrow FS_i = FS_j$;

For a given k, we merge those cluster candidates with k-mismatched patterns (i.e., frequent word sequences). The concept is that two cluster candidates with k-mismatched patterns may cover similar topics. For example, we can say that the topic covered by the cluster candidate i with $FS_i =$"boys play" is close to the topic covered by the cluster candidate j with $FS_j =$"play basketball". So, we can merge them to have a bigger cluster, which covers the topic about who plays which game. The value of parameter k determines how fine the final clustering would be. If k is 0, the cluster candidates are the final clusters, which are the finest clusters that could be found from the GST. As k value increases, the topic covered by each cluster of the final clustering would be more general.

After we merge several cluster candidates into clusters, we may find some clusters have too much overlapping between their document id sets. The overlapping of two clusters, C_i and C_j, can be measured as follows:

$$O(C_i, C_j) = \frac{Ids_i \cap Ids_j}{Ids_i \cup Ids_j} \tag{2.1}$$

If $O(C_i, C_j)$ is larger than the specified overlapping threshold value δ, these two clusters are combined into one cluster. Obviously, the range of δ is $[0, 1]$: When $\delta = 0$, these two clusters are disjoint; and when $\delta = 1$, these two clusters have the same set of documents, which does not mean these two clusters are identical because this set of documents may cover two different topics.

In the end, we collect those documents that are not in any cluster because they do not contain a frequent word sequence. These documents form a cluster by themselves, and their topic could be specified as "other issues".

Our text clustering algorithm CFWS is summarized as follows:

1. Given a collection of text documents $D = \{d_1, d_2, d_3, \ldots d_n\}$, find the set of frequent 2-word sets of D with the user-specified minimum support. Obtain *WS*, the set of all words, each of which is a member of a frequent 2-word set.

2. Reduce each document d_i, $1 \leq i \leq n$, into a compact document d_i' by removing every word w from d_i if $w \notin$ *WS*.

3. Insert each compact document into the GST.

4. Using the depth-first traversing, visit every node in the GST. If a node j has a frequent word sequence FS_j with the set of document ids Ids_j, create a cluster candidate $cc_j[FS_j, Ids_j]$.

5. Merge the related cluster candidates into clusters based on the k-mismatch concept.

6. Combine the overlapping clusters if necessary.

2.5 Clustering Based on Frequent Word Meaning Sequences (CFWMS)

In the previous CFWS algorithm, in order to find frequent word sequences in the text database, we count the occurrences of each word in the documents first. This is a *word*

form matching process, where a word form refers to a literal term in text documents. On the other hand, a *word meaning* refers to the lexicalized concept that a word form can be used to express [66]. In the real world, people may use different word forms to express the same word meaning, and those word forms are called synonyms. A word meaning can be represented by a synonym set, or shortly *synset*, a set of word forms which are synonyms.

In this chapter, a synset is denoted by SS, for example $SS_1 = \{\text{car, auto}\}$. This lexical relation between word forms may affect our clustering result. For example, "auto" is a synonym of "car", so they are interchangeable in documents. In a text database, when the word form matching is performed to find frequent words, the support counts of "car" and "auto" could be 6 and 4, respectively. If the minimum support count is 9, neither of these two word forms is frequent. However, the sum of their support counts is larger than the minimum support count if 6 documents refer to the automobile by using "car" and other 4 documents use "auto". If we treat these two word forms as one, these 10 documents may be grouped into one cluster.

Hyponymy/hypernymy is a semantic relationship between word meanings, which is also very important for text clustering. This relationship is also called subset/superset relationship, and it represents the relationship between a specific word meaning and a general word meaning. For example, the hypernym of a synset {car, railcar} is {vehicle}, and the hyernym of {vehicle} is {conveyance, transport}. If we use "$\rightarrow$" to represent this relationship, these three synsets could be linked as {car, railcar} $\rightarrow$ {vehicle} $\rightarrow$ {conveyance, transport}. In this case, {vehicle} is a direct hypernym of {car, railcar} and {conveyance, transport} is a inherited hypernym of {car, railcar}. Such a link of a synset with its hypernyms is called a *synset link* (SL). Documents containing "car" may share the same topic with other documents containing "vehicle" or "transport". If we perform only the word form matching, we may lose this information.

Word meanings are better than word forms in terms of representing the topics of documents. In order to improve the quality of clustering, we propose a new algorithm named Clustering based on Frequent Word Meaning Sequences (CFWMS), which uses frequent word

meaning sequences as the measurement of the closeness between documents.

In CFWMS, word forms in the documents are converted to the word meanings they express first. After the conversion, each text document is treated as a sequence of word meanings. For example, a text document d can be viewed as $d =< SS_1, SS_2, SS_3, \ldots >$. A word meaning sequence is considered frequent if there are more than certain number (or percentage) of documents containing it. A frequent word meaning sequence is denoted by *FMS* in this chapter.

Since most words have multiple word meanings, identifying the right word meaning that a word form expresses in a certain lexical environment is not a trivial problem. In our algorithm, we use a *meaning union* (MU), which is a union of synset links, to predict the real word meaning. A synset link can be treated as a meaning union by itself. For example, for a word form "box" in a document, there is a meaning union $MU = \{SS_1 \rightarrow SS_2,\ SS_3 \rightarrow SS_4\}$, where SS_1={box}, SS_2={container}, SS_3={box, loge}, and SS_4={compartment}. We expect that one of the synsets in MU is the real word meaning expressed by the word form "box" in this document. We will explain how to find the meaning unions in Section 2.5.1. In this way, a document d, after the conversion, can be represented by meaning unions as $d' =< MU_1, MU_2, MU_3, \ldots >$. Similarly, a frequent word meaning sequence can be also represented as $FMS =< MU_1, MU_2, \ldots >$.

2.5.1 CFWMS Algorithm

Our CFWMS algorithm has three steps: 1) preprocessing of documents to convert word forms into meaning unions; 2) finding frequent word meaning sequences and collecting cluster candidates; and 3) combining cluster candidates to obtain the final clusters.

2.5.1.1 Document Preprocessing Using WordNet

The most important procedure in the preprocessing of documents is to convert the word forms into meaning unions by using an ontology. As an ontology, we used WordNet [66], an on-line lexical reference system. WordNet covers semantic and lexical relations between

word forms and word meanings, such as synonymy, polysemy, and hyponymy/hypernymy.

Since WordNet provides morphological relations between word forms, before the conversion of word forms into word meanings, we perform the removal of stop words, but not stemming. WordNet contains only nouns, verbs, adjectives and adverbs. Since nouns and verbs are more important in representing the content of documents and also mainly form the frequent word meaning sequences, we focus only on nouns and verbs and remove all adjectives and adverbs from the documents. For those word forms that do not have entries in WordNet, we keep them in the documents since these unidentified word forms may capture unique information about the documents.

For each document, two passes are required for the conversion. The first pass is to retrieve the meaning union (MU) from WordNet for every noun and verb in the document. In WordNet, a word form's multiple word meanings represented by synsets are ordered from the most to the least frequently used. For every noun and verb, we select the first two synsets containing the word form. For each synset selected, one direct hypernym synset is retrieved, too. If the word form has only one synset, one inherited hypernym synset is retrieved as well. In this way, each word form has its meaning union which contains at least one synset link. We tried different numbers of synsets and hypernyms for each word form, and found these selections produce a good clustering result in most cases. For example, a document $d_1 =< w_1, w_2, w_3 >$ can be converted to a sequence of meaning unions as $d'_1 =< MU_1, MU_2, MU_3 >$, where $MU_1 = \{SS_4 \rightarrow SS_5,\ SS_6 \rightarrow SS_7\}$, $MU_2 = \{SS_3 \rightarrow SS_8,\ SS_9 \rightarrow SS_{10}\}$, $MU_3 = \{SS_1 \rightarrow SS_5,\ SS_2 \rightarrow SS_3\}$.

In the second pass, we try to reduce the number of unique meaning unions in the converted document. The word forms in one document may tend to express similar word meanings. Thus, if different meaning unions share a synset, we replace them with a single meaning union containing a merged synset link. In this case, the order of replacement is based on the frequencies of the synsets in the document. For that purpose, the occurrences of each synset in the meaning unions of the document is counted, and a list of synsets with their supporting meaning unions is created. Table 2.2 shows the details of each replacement step

for the given example. The first three columns in Table 2.2 show the synsets, their support counts and supporting meaning unions.

We perform the replacement as follows:

Step 1: Select the synset with the largest support count in the list.

- If the support count of the selected synset is larger than 1, which means more than one meaning union contains this synset, then we create a new meaning union to replace them. For example, if SS_5 is selected, its supporting meaning unions, MU_1 and MU_3, are replaced by a new meaning union MU_4. This new meaning union contains one synset link (SL) composed of SS_5 and all its hyponyms/hypernyms in its supporting meaning unions; i.e., $MU_4 = \{\{SS_1, SS_4\} \rightarrow SS_5\}$. The support count of the selected synset SS_5 is reduced to 0. At the same time, the support counts of other synsets which were in MU_1 and MU_3 are reduced by one since we have removed these two meaning unions from the document.
- If the support count of the selected synset is 1, its supporting meaning union stays and the support counts of the synsets in it are reduced to 0 since we have processed this meaning union.

Step 2: Repeat Step 1 until the support count of every synset in the list becomes 0.

Since all the meaning unions sharing the same synset are replaced by one meaning union which contains only one SL, the number of unique meaning unions in the document can be reduced.

2.5.1.2 Finding Frequent Word Meaning Sequences and Collecting the Cluster Candidates

After the preprocessing step, every document in the database is coverted to a string of meaning unions. For example, we may have an example preprocessed database $D' = \{d'_1, d'_2, d'_3\}$, where

- $d'_1 =< MU_4,\ MU_2,\ MU_4 >$

Synsets	$d'_1 =< MU_1, MU_2, MU_3 >$		Step 1 (pick SS_5) $d'_1 =< MU_4, MU_2, MU_4 >$		Step 2 (pick SS_3) $d'_1 =< MU_4, MU_2, MU_4 >$	
	Support Count	Meaning Union Ids	Support Count	Meaning Union Ids	Support Count	Meaning Union Ids
SS_5	2	$MU_1,\ MU_3$	$2 \Rightarrow 0$		0	
SS_3	2	$MU_2,\ MU_3$	$2 \Rightarrow 1$	MU_2	$1 \Rightarrow 0$	
SS_2	1	MU_3	$1 \Rightarrow 0$		0	
SS_1	1	MU_3	$1 \Rightarrow 0$		0	
SS_4	1	MU_1	$1 \Rightarrow 0$		0	
SS_6	1	MU_1	$1 \Rightarrow 0$		0	
SS_7	1	MU_1	$1 \Rightarrow 0$		0	
SS_8	1	MU_2	1	MU_2	$1 \Rightarrow 0$	
SS_9	1	MU_2	1	MU_2	$1 \Rightarrow 0$	
SS_{10}	1	MU_2	1	MU_2	$1 \Rightarrow 0$	

Table 2.2: Reorganizing the meaning unions in document d'_1

  - $MU_4 = \{\{SS_4, SS_1\} \rightarrow SS_5\}$
  - $MU_2 = \{SS_3 \rightarrow SS_8,\ SS_9 \rightarrow SS_{10}\}$

- $d'_2 =< MU_5,\ MU_6 >$
  - $MU_5 = \{SS_2 \rightarrow SS_5,\ SS_7 \rightarrow SS_{11}\}$
  - $MU_6 = \{\{SS_8, SS_6\} \rightarrow SS_{13}\}$

- $d'_3 =< MU_1,\ MU_3,\ MU_7 >$
  - $MU_1 = \{SS_{12} \rightarrow SS_{13}\}$
  - $MU_3 = \{SS_{14} \rightarrow SS_{15}\}$
  - $MU_7 = \{SS_{16} \rightarrow SS_{17}\}$

In order to find frequent word meaning sequences, we use association rule miner to find frequent meaning unions (FMUs) first, then the frequent sets of 2 meaning unions. If we perform exact matching between meaning unions, we cannot find any frequent sets of 2 meaning unions in D' when the minimum support count is 2. In fact, there is a frequent

word meaning sequence in this database. If we check the synsets in every meaning union, we can find that synset SS_5 is shared by MU_4 of d'_1 and MU_5 of d'_2; synset SS_8 is shared by MU_2 of d'_1 and MU_6 of d'_2; and synset SS_{13} is shared by MU_6 of d'_2 and MU_1 of d'_3. The frequent word meaning sequence (FMS) found is $< SS_5, SS_8 >$.

In order to find the frequent word meaning sequences, we need to check and collect the counts of synsets and their hyponyms/hypernyms, i.e. SLs, instead of just matching the word meaning unions. For instance, when we check MU_2 of d'_1, there are two SLs in MU_2: $\{SS_3 \rightarrow SS_8\}$ and $\{SS_9 \rightarrow SS_{10}\}$. Let's denote them by SL_A and SL_B, respectively. We record SL_A and SL_B as unique meaning unions and increase their counts by one, respectively. When MU_6 of d'_2 is checked, we meet SS_8. Since SS_8 is in SL_A, we insert SS_{13} and SS_6 into SL_A, and increase its count by one. In the same way, when we check MU_1 of d'_3, we meet SS_{13} and insert SS_{12} into SL_A and, increase its count by one again. Table 2.3 shows the support counts of unique meaning unions (as SLs) in database D' after we checked all three documents. SL_C and SL_A are frequent meaning unions and $\{SL_C,\ SL_A\}$ is a frequent set of 2 meaning unions. The synsets in these frequent meaning unions are called frequent synsets. In this example, the frequent synsets are $SS_1,\ SS_2,\ SS_4,\ SS_5,\ SS_3,\ SS_8,\ SS_{12},\ SS_6$, and SS_{13}.

SL	Synsets with Hyponyms/Hypernyms	Support Count	Document Ids
SL_C	$\{SS_1, SS_2, SS_4\} \rightarrow SS_5$	2	1, 2
SL_A	$\{\{SS_3 \rightarrow SS_8\}, SS_{12}, SS_6\} \rightarrow SS_{13}$	3	1, 2, 3
SL_D	$SS_7 \rightarrow SS_{11}$	1	2
SL_B	$SS_9 \rightarrow SS_{10}$	1	1
SL_E	$SS_{14} \rightarrow SS_{15}$	1	3
SL_F	$SS_{16} \rightarrow SS_{17}$	1	3

Table 2.3: Support counts of the meaning unions in database D'

After finding the frequent sets of 2 meaning unions, in order to reduce each document into a compact document, we remove the meaning unions which do not contain a frequent synset, and replace each meaning union containing a frequent synset with the frequent meaning union containing that frequent synset. Thus, each resulting compact document contains only the

frequent meaning unions. In our example, the resulting database D'' is composed of compact documents d''_1, d''_2, and d''_3:

- $d''_1 =< SL_C,\ SL_A,\ SL_C >$
- $d''_2 =< SL_C,\ SL_A >$
- $d''_3 =< SL_A >$

Then, we can build the Generalized Suffix Tree (GST) for the compact documents and collect the cluster candidates the same way as in the CFWS algorithm described in Section 2.4 . By traversing the GST, the frequent meaning union sequences are found. They are used to represent the frequent word meaning sequences and serve as the labels of the cluster candidates collected. In our example case, a cluster candidate we can obtain is $cc[FMS =$ "$SL_C,\ SL_A$", $Ids = \{1,\ 2\}]$.

2.5.1.3 Combining the Cluster Candidates

The frequent word meaning sequences of a cluster candidate describe the topics those documents cover. When we combine the cluster candidates, we don't need to use the k-mismatch concept used in the CFWS algorithm. The reason is because the cluster candidates we found are general enough since word meanings were used, instead of word forms. We only need to check the overlapping between cluster candidates, as in CFWS.

2.6 Experimental Evaluation

In this section, we evaluate the performance of our text clustering algorithm in terms of the scalability of finding frequent word sequences and the accuracy of clustering. We implemented our algorithm in C++ on a SuSE Linux PC with a Celeron 500 MHz processor and 384 MB memory.

2.6.1 Data Sets

For the performance evaluation, two groups of data sets are used. One group is typical text document sets which were widely used in text clustering researches. They are different in terms of document size, cluster size, number of classes, dimension of database and document distribution. We chose this group of data sets to test our algorithms' performance on typical text documents. There are six data sets in this group. They are Re1, Re2 and Re3 from Exchanges, Organizations and People categories of the Reuters-21578 Distribution 1.0 [55], and Ce1, Ce2 and Ce3 from the CISI abstracts of Classic database [9].

Another group of data sets were prepared by us. We tried to simulate the case of using a search engine to retrieve the desired documents from a database, and we adopted the Lemur Toolkit [37] as the search engine. The English newswire corpus of the HARD track of the Text Retrieval Conference (TREC) 2004 [27] is used as the database. This corpus includes about 652,309 documents (in 1575 MB) from 8 different sources. The user queries were sent to the search engine, and the top 200 results of these queries were collected and classified as the data sets, denoted by Se1, Se2, and Se3, for our evaluation. The reason why we chose only top 200 documents is that usually users do not read more than 200 documents for a single query.

Each document of the test data sets has been pre-classified into one or more classes. This information is hidden during the clustering processes and used to evaluate the clustering quality of each clustering algorithm in terms of the accuracy. Table 2.4 summarizes the characteristics of all the data sets used for our experiments.

2.6.2 Evaluation of the Finding Frequent Word Sequences

We evaluated our method of finding frequent word sequences in terms of its scalability. The whole mining process has two steps: finding frequent 2-word sets, then building and traversing the GST. Any frequent itemset mining algorithm can be used to find frequent 2-word sets in our method. In our experiment, we adopted Apriori algorithm [1], which is the most representative frequent itemset mining algorithm. There are many other frequent

Data Set	Num. of Doc.	Num. of Classes	Min. Class Size	Max. Class Size	Avg. Class Size	Num. of Unique Terms	Avg. Doc. Length	Num. of Total Terms
Re1	340	7	28	97	53	3,368	69	24,414
Re2	807	9	20	349	98	6,488	144	116,816
Re3	797	15	20	168	62	5,466	108	86,744
Ce1	262	4	56	144	93	2,224	64	16,944
Ce2	355	4	70	146	108	2,633	65	23,070
Ce3	178	4	35	117	65	2,102	60	11,639
Se1	200	3	44	92	66	8,998	329	65,868
Se2	200	4	20	98	54	12,368	736	147,390
Se3	200	3	18	152	70	9,301	940	188,047

Table 2.4: Summary of data sets used for experiments

itemset mining algorithms proposed [29], but there isn't much difference in their performance to find only frequent 2-itemsets. The efficiency of Apriori is sensitive to the minimum support level. When the minimum support is decreased, the runtime of Apriori increases as there are more frequent itemsets.

The most time-consuming part is building and traversing the GST. As reported in [24, 62], the GST construction time can be linear with the size of the whole database, and it can be constructed incrementally as the documents are read from files. The number of unique words affects the creation and traverse times of the GST. Thus, by keeping only the frequent words in the database, the construction of the GST of large text databases becomes more feasible. In our method, the size of whole database is dramatically reduced by removing infrequent words from the documents. For example, the Re1 data set has 807 documents with 116,816 total words and 6488 unique words. Its average document size and length are 1.4 MB and 144, respectively. The Apriori algorithm finds 1049 frequent 2-word sets with 175 unique words when the minimum support is 10%. After removing the infrequent words from the documents, the total number of words is reduced to 52,792, and the average document length is reduced to 51. Documents in Re2 are about organizations. By using our method, 64 frequent word sequences are found when the minimum support is 5%. Some of

them are listed in Table 2.5, and they represent the topics of the data set very well.

Frequent Word Sequence (after stemmed)	Support Count
world bank	113
oil price	73
export quota	49
intern monetari fund	101
west germani	112
financ minist	82
ec commiss	91
cooper develop	49
european currenc	66
agricultur minist	43
tariff trade	96

Table 2.5: Some frequent word sequences in the Re2 data set

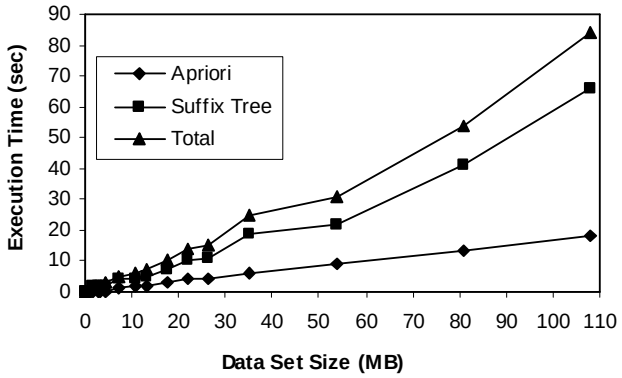


Figure 2.3: Scalability of finding frequent word sequences wrt the data set size

Even though the construction time of the GST is reduced dramatically when the compact documents are used, there is a trade-off between the construction time and the memory space requirement when the GST is large. When the average length of the compact documents is big, we may have the memory bottleneck problem as the GST size becomes larger than the available memory size. To handle this problem, many efficient in-memory suffix tree construction algorithms were proposed [16, 17, 35, 44].

Since the number of unique words in text databases is relatively large, we used a hash

table in our implementation for efficient searching of the child nodes during the construction of the GST. In order to test the scalability of our method, we increased the number of documents in the test data set by duplicating Re1, Re2 and Re3, and the execution time is plotted against the data set size in Figure 2.3. As we can see, the execution time increases linearly with the data set size.

We also tested our method for various minimum support levels, and the result is shown in Figure 2.4. As the minimum support level is increased, the execution time decreases since less frequent 2-word sets are found, and it leads to a smaller GST. From these experimental results, we can conclude that our method of finding frequent word sequences is very scalable.

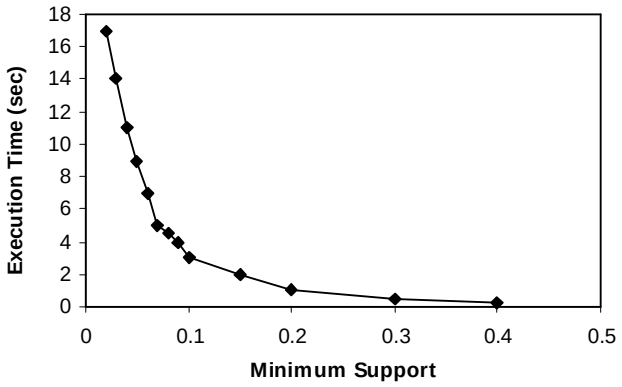


Figure 2.4: Scalability of finding frequent word sequences wrt the minimum support

2.6.3 Evaluation Method of the Text Clustering

We used the *F-measure* and *purity* values to evaluate the accuracy of our clustering algorithm. The F-measure is a harmonic combination of the *precision* and *recall* values used in information retrieval [56]. As we described how our data sets were prepared in Section 2.6.1, each cluster obtained can be considered as the result of a query, whereas each pre-classified set of documents can be considered as the desired set of documents for that query. Thus, we can calculate the recall $R(i, j)$ and precision $P(i, j)$ of each cluster j for each class i.

If n_{ij} is the number of the members of class i in cluster j, n_j is the number of the members of cluster j, and n_i is the number of the members of class i, then $R(i, j)$ and $P(i, j)$ can be

defined as follows:

$$R(i,j) = \frac{n_{ij}}{n_i} \tag{2.2}$$

$$P(i,j) = \frac{n_{ij}}{n_j} \tag{2.3}$$

The corresponding F-measure $F(i,j)$ is defined as:

$$F(i,j) = \frac{2 * R(i,j) * P(i,j)}{R(i,j) + P(i,j)} \tag{2.4}$$

Then, the F-measure for the whole clustering result is defined as

$$F = \sum_i \frac{n_i}{n} \max_j(F(i,j)) \tag{2.5}$$

where n is the total number of documents in the collection. In general, the larger the F-measure is, the better the clustering is [59].

The purity of a cluster represents the fraction of the cluster corresponding to the largest class of documents assigned to that cluster, thus the purity of a cluster j is defined as:

$$Purity(j) = \frac{1}{n_j} \max_i(n_{ij}) \tag{2.6}$$

The overall purity of the clustering is a weighted sum of the cluster purities:

$$Purity(C) = \sum_j \frac{n_j}{n} Purity(j) \tag{2.7}$$

In general, the larger the purity value is, the better the clustering is [71].

2.6.4 Comparison of CFWS and CFWMS with Other Algorithms

For a comparison with our CFWS and CFWMS, we also executed bisecting k-means and FIHC on the same data sets. We chose bisecting k-means because it has been reported to produce a better clustering result consistently compared to k-means and agglomerative hierarchical clustering algorithms [59]. FIHC is chosen because, like CFWS, it uses frequent word sets. For a fair comparison, we did not implement the bisecting k-means and FIHC algorithms by ourselves. We downloaded the CLUTO toolkit [10] to perform the bisecting k-means, and obtained FIHC version 1.0 [19] from the inventor of FIHC.

Data sets were preprocessed before they were used in our experiments. The first step is to remove the stop words from the documents. Then for CFWS, bisecting k-means and FIHC algorithms, the words are stemmed by using the Porter's suffix-stripping algorithm [59]. It is important to do the stemming since it can eliminate the minor difference between words with the identical meaning. For CFWMS, we used the WordNet ontology to convert word forms to word meanings. As a result, the dimension of text database is reduced further, and Figure 2.5 shows the changes of the dimension in test data sets.

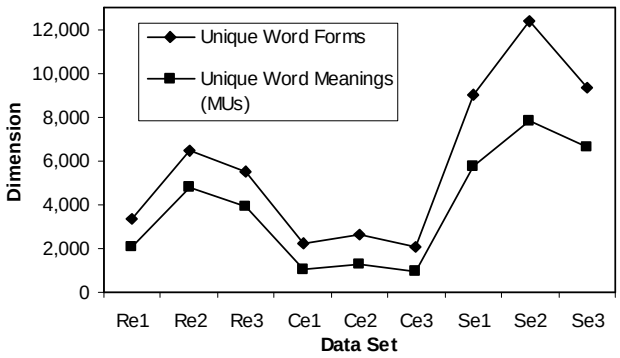


Figure 2.5: Dimensionality changes after applying WordNet

Table 2.6 shows the F-measures of four algorithms: bisecting k-means, FIHC, CFWS and CFWMS. For the bisecting k-means algorithm, we specified the desired number of clusters to be same as the number of the classes in each data set. FIHC, CFWS and CFWMS do not take the number of clusters as an input parameter, and we specified the same minimum support level for them, in the range of 5–15%, for a fair comparison. Based on the F-measures, it is clear that our CFWS and CFWMS algorithms consistently outperform two other algorithms on the sets of typical text documents as well as on the sets of search query results. For CFWS and CFWMS, the overlapping threshold value δ for the merging of clusters was 0.5, and Figure 2.6 shows the effect δ on the F-measure of CFWS. As we can see, the F-measure is not sensitive if δ is higher than 0.5.

The F-measure represents the clustering accuracy. Our two algorithms have better F-measures because we use a better model for text documents. Both bisecting k-means and FIHC use the vector space model for text documents. However, the vector space model

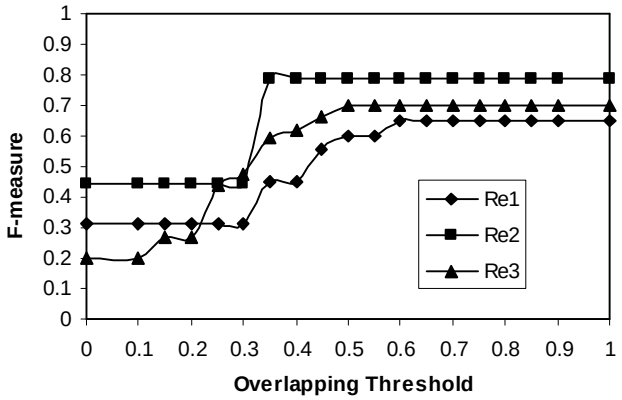


Figure 2.6: Effect of the overlapping threshold δ on the F-measure of CFWS

cannot capture the order of words, which is important in representing the context in the text document. On the contrary, our model stores the words as well as their orders, which provide more valuable information for clustering.

Data Set	Bisecting k-means	FIHC	CFWS	CFWMS
Re1	0.606	0.506	0.651	0.721
Re2	0.677	0.740	0.790	0.790
Re3	0.675	0.390	0.703	0.701
Ce1	0.430	0.506	0.541	0.550
Ce2	0.466	0.460	0.480	0.480
Ce3	0.489	0.515	0.540	0.604
Se1	0.611	0.664	0.705	0.711
Se2	0.529	0.461	0.689	0.710
Se3	0.716	0.759	0.800	0.806

Table 2.6: F-measures of the clustering algorithms

Both CFWS and FIHC use the frequent words to cluster documents, and FIHC's measurement of the closeness between clusters is similar to ours. However, FIHC uses the frequent word sets to cluster documents, whereas CFWS uses the frequent word sequences. If a word is a member of a frequent k-word sequence, it must be a member of a frequent k-word set. But a member of a frequent k-word set is not necessarily a member of a frequent k-word sequence. It is also true that a frequent k-word set is not necessarily a frequent k-word

sequence. As a result, FIHC has a higher probability of grouping unrelated documents into the same cluster. CFWMS has better F-mesures than CFWS in most cases because it can identify the same word meaning sequences represented by different word form sequences.

Data Set	Bisecting k-means	FIHC	CFWS	CFWMS
Re1	0.622	0.527	0.749	0.751
Re2	0.796	0.652	0.679	0.810
Re3	0.777	0.444	0.504	0.675
Ce1	0.640	0.605	0.620	0.635
Ce2	0.690	0.585	0.601	0.637
Ce3	0.790	0.692	0.702	0.775
Se1	0.62	0.66	0.703	0.803
Se2	0.695	0.563	0.714	0.789
Se3	0.837	0.84	0.851	0.852

Table 2.7: Purity values of the clustering algorithms

Table 2.7 shows the purity values of the four clustering algorithms. As we can see, bisecting k-means performs better than both FIHC and CFWS for almost all of the typical text document sets: Re1, Re2, Re3, Ce1, Ce2 and Ce3. The reason is that people may use different words to express the same meaning, but both FIHC and CFWS perform the exact word matching during the procedure of finding frequent word sets and sequences, respectively. Our CFWMS algorithm is designed to solve this problem. In CFWMS, we first apply an ontology, WordNet, to convert word forms to word meanings. Then, we match the word meanings instead of word forms. The frequent word meaning sequences are used as the measurement of the closeness between documents in CFWMS. In typical text documents, the same word meanings may not be expressed by the same word forms because synonyms and polysemous words are used. By using WordNet, the lexical and semantic relations between word forms and word meanings are explored. The word meanings expressed by different word forms are also captured successfully in our CFWMS algorithm. Moreover, the topics covered in the typical text documents are represented by the frequent word meaning sequences.

For the search query results (i.e., Se1, Se2 and Se3), CFWS performs better than bisect-

ing k-means and FIHC. The unique characteristics of these document sets can explain the performance results. These document sets were obtained from the retrieval lists of a search engine for user queries. We used the simple TFIDF retrieval model of the Lemur Toolkit to perform the retrieval for each query. The documents obtained from this retrieval model are more likely to have common words between them compared to the typical text documents. In other words, the topics covered by this type of document sets are much closer than those of the typical documents. For example, top 200 documents on the retrieval list for the query of "tea consumption increased in US" cover 3 topics: "tea consumption", "tea industry" and "unrelated issues, such as drinking problem, sports, etc." By specifying a lower minimum support level, we can obtain finer clusters to cover the subtopics in the data set, which are shown in Table 2.8.

class	subclasses
tea consumption	good for health; tea drinking culture; tea popularity
tea industry	tea growing issues; tea export issues; tea auction issues
unrelated issues	sports; drinking problem; others

Table 2.8: Classification of the Se1 data set

For this type of data sets, our CFWS algorithm can work better as it can group the documents into much finer clusters. The reason is that our algorithm can tell the minor differences among subtopics by recognizing frequent word sequences in the documents. By clustering the retrieved documents, our CFWS algorithm enables the web search engines to provide more accurate search results to the user. The purity values show that CFWMS can improve the quality of the clustering for both typical text documents and search query results.

2.7 Conclusion

In this chapter, first we proposed a new text document clustering algorithm named CFWS, which stands for Clustering based on Frequent Word Sequences. Unlike the traditional

vector space model, our model utilizes the sequential patterns of the words in the document. Frequent word sequences discovered from the document set can represent the topics covered by the documents very well, and the documents containing the same frequent word sequences are clustered together in our algorithm.

To facilitate the discovery of the frequent word sequences from documents, we use the Generalized Suffix Tree (GST) built on the frequent words within each document. The performance of our frequent word sequence mining algorithm is quite scalable. For very large data sets, we can adopt some of the efficient in-memory GST construction algorithms proposed [16, 17, 35, 44].

Most existing clustering algorithms do not satisfy the unique requirements of the text document clustering, such as handling high dimension and context-sensitive languages, and providing overlapped clusters and self-explanatory labels of the clusters. Our CFWS algorithm explores unique characteristics of text documents by using the frequent word sequences to reduce the high dimension of the documents and to measure the closeness between them. Our experimental results show that CFWS performs better than other clustering algorithms in terms of accuracy, especially for the fine clustering of documents in the same category, which is a very useful feature for modern web search engines.

Then, we proposed another new text document clustering algorithm named CFWMS, which stands for Clustering based on Frequent Word Meaning Sequences. CFWMS enhanced CFWS by using frequent word meaning sequences to measure the closeness between documents. Frequent word meaning sequences can capture the topics of documents more precisely than frequent word sequences. To find frequent word meaning sequences, we used the synonyms, hyponyms and hypernyms provided by the WordNet ontology to preprocess the documents. CFWMS has a better accuracy than CFWS on most of our test data sets.

Chapter 3

Text Clustering with Feature Selection By Using Statistical Data

3.1 Introduction

How to explore and utilize the huge amount of text documents is a major question in the areas of information retrieval and text mining. Document clustering is one of the most important text mining methods, which are developed to help users to effectively navigate, summarize, and organize text documents. By organizing a large amount of documents into a number of meaningful clusters, document clustering can be used to browse a collection of documents or organize the results returned by a search engine in response to a user's query. It can significantly improve the precision and recall in information retrieval systems [56], and it is an efficient way to find the nearest neighbors of a document [8]. The problem of document clustering is generally defined as follows: given a set of documents, we would like to partition them into a predetermined or an automatically derived number of clusters, such that the documents assigned to each cluster are more similar to each other than the documents assigned to the other clusters. In other words, the documents in one cluster share the same topic, and the documents in different clusters represent different topics.

In most existing document clustering algorithms, documents are represented using the vector space model [56], which treats a document as a bag of words. A major characteristic of this representation is the high dimensionality of the feature space, which imposes a big

challenge to the performance of clustering algorithms. These algorithms could not work efficiently in high dimensional feature spaces due to the inherent sparsity of the data [2]. Another problem is that not all features are important for document clustering. Some of the features may be redundant or irrelevant. Some may even misguide the clustering result, especially when there are more irrelevant features than relevant ones. In such case, selecting a subset of original features often leads to a better clustering performance [40]. Feature selection not only reduces the high dimensionality of the feature space, but also provides a better data understanding, which improves the clustering result. The selected feature set should contain sufficient or more reliable information about the original data set. For document clustering, this will be formulated into the problem of identifying the most informative words within a set of documents for clustering.

Feature selection has been widely used in supervised learning, such as text classification. It is reported that feature selection can improve the efficiency and accuracy of text classification algorithms by removing redundant and irrelevant terms from the corpus [52]. Traditional feature selection methods for classification are either supervised or unsupervised, depending on whether the class label information is required for each document. Those unsupervised feature selection methods, such as the ones using document frequency and term strength, can be easily applied to clustering [67]. But it is shown in [40] that supervised feature selection methods using the information gain [54] and χ^2 statistic can improve the clustering performance better than unsupervised methods when the class labels of documents are available for the feature selection. However, supervised feature selection methods cannot be directly applied to document clustering because usually the required class label information is not available. In [40], an iterative feature selection method is proposed to utilize supervised feature selection methods to select features iteratively and perform the text clustering at the same time.

In many previous text mining and information retrieval researches, the χ^2 term-category independence test has been widely used for the feature selection in a separate preprocessing step before the text categorization [52]. By ranking their χ^2 statistic values, features that

have strong dependency on the categories can be selected [40, 42], and this method is denoted as CHI in this chapter.

In this chapter, we extended the χ^2 independence test by introducing new statistical data that can measure whether the dependency between a term and a category is positive or negative. We also developed a new supervised feature selection method, named CHIR, which is based on the χ^2 statistic and the new term-category dependency measure. Unlike CHI, CHIR selects features having strong positive dependency on the categories. In other words, CHIR keeps only those features relevant to the categories. Furthermore, we explored CHIR in text clustering, and developed a new text clustering algorithm, named TCFS, which stands for Text Clustering with Feature Selection. TCFS iteratively performs the clustering and the supervised feature selection, such as CHIR, alternately. Thus, the whole process of TCFS is basically a learning process. While the information of the clusters is utilized to find better features (i.e., terms), the quality of the clustering result is improved by reducing the weight of irrelevant features. As the TCFS algorithm converges, both a good clustering result and an informative feature subset are obtained.

Our experimental results demonstrated that the TCFS algorithm using the CHIR feature selection method performs better than k-means, k-means with the Term Strength (TS) feature selection method [40], and TCFS with CHI in terms of the accuracy of clustering results for various real data sets.

The rest of this chapter is organized as follows. In Section 3.2, we describe the χ^2 term-category independence test and the feature selection method CHI. Then, we propose a new term-category measure and a new feature selection method CHIR. In Section 3.3, we propose a high performance text clustering algorithm TCFS, which can adopt the feature section method CHIR without knowing the class information of the documents in advance. In Section 3.4, CHIR is compared with CHI in terms of the cluster cohesiveness, and the clustering accuracy of TCFS with CHIR is compared with those of other clustering and feature selection algorithms. Section 3.5 contains some conclusions, and Section 3.6 outlines our future research topics.

3.2 Feature Selection Based on χ^2 Statistics

3.2.1 χ^2 Term-Category Independence Test

In text mining and information retrieval, we often use the χ^2 statistic to measure the degree of the dependency between a term and a specific category. This is done by comparing the observed co-occurrence frequencies in a 2-way contingency table with the frequencies expected when they are assumed to be independent. Suppose that the corpus contains n labeled documents and they fall into m categories. After the stop words removal and the stemming, distinct terms are extracted from the corpus. We use an example to explain the χ^2 term-category independence test.

	c	$\neg c$	$\sum$
w	40	80	120
$\neg w$	60	320	380
$\sum$	100	400	500

Table 3.1: A 2×2 term-category contingency table

Example 1: To analyze the relationship between a word w and a category c, we create a two-way contingency table, shown as Table 3.1. The row variable, term, has two possible values: $\{w, \neg w\}$. The column variable, category, may take either one in $\{c, \neg c\}$. Each cell at the position (i, j), where $i \in \{w, \neg w\}$ and $j \in \{c, \neg c\}$, contains the observed frequency, denoted by $O(i, j)$. For example, $O(w, c)$ is the number of documents which are in the category c and contain the term w, and $O(\neg w, \neg c)$ is the number of documents which neither belong to c nor contain w.

For the χ^2 term-category independence test, we consider the *null hypothesis* and the *alternative hypothesis*. The null hypothesis is that the two variables, term and category, are independent of each other. On the other hand, the alternative hypothesis is that there is some dependency between the two variables. To test the null hypothesis, we compare the observed frequency with the expected frequency calculated under the assumption that the null hypothesis is true. The expected frequency $E(i, j)$ can be calculated as:

$$E(i,j) = \frac{\sum_{a\in\{w,\neg w\}} O(a,j) \times \sum_{b\in\{c,\neg c\}} O(i,b)}{n} \tag{3.1}$$

Using Equation 3.1, we get $E(w,c) = 24$, $E(w,\neg c) = 96$, $E(\neg w,c) = 76$, and $E(\neg w,\neg c) = 304$. The χ^2 statistic is defined as:

$$\chi^2_{w,c} = \sum_{i\in\{w,\neg w\}} \sum_{j\in\{c,\neg c\}} \frac{(O(i,j) - E(i,j))^2}{E(i,j)} \tag{3.2}$$

Using Equation 3.2, we get $\chi^2_{x,c} = 17.61$. The degree of freedom is $(2-1)\times(2-1) = 1$ for our case. Looking up the tables of the χ^2 distribution, we get the critical value $\chi^2_{0.001} = 10.83$ for the confidence level 0.1%. Since $\chi^2_{0.001}$ is much smaller than 17.61, we reject the null hypothesis. This can be explained as the divergence between the observed frequency and the expected frequency is statistically significant. That means, it is very unlikely that the divergence is just caused by the random sampling process. Thus, we reject the null hypothesis and believe there is some dependency between w and c; i.e., the distribution of the term w is related to the category.

As shown in Equation 3.2, if the difference between the observed frequency and the expected frequency is bigger, then the χ^2 statistic becomes bigger, and the word is more informative for the category. This is the basic idea behind most previous researches on the feature selection for text categorization. The feature selection method CHI could be described as follows. For an m-ary classifier, we usually define the term-goodness of a term w as either one of:

$$\chi^2_{avg}(w) = \sum_{j=1}^{m} p(c_j)\chi^2_{w,c_j} \tag{3.3}$$

$$\chi^2_{max}(w) = \max_j\{\chi^2_{w,c_j}\} \tag{3.4}$$

where $p(c_j)$ is the probability of the documents to be in the category c_j. Then, the terms whose term-goodness measure is lower than certain threshold value would be ignored in the training of the classifier. It is reported in [52] that Equation 3.4 results better performance than Equation 3.3. In some systems, the multi-class text categorization problem is reduced to the training of a set of binary classifiers. For each binary classifier, a separate feature

selection process is performed and $\chi^2_{w,c}$ is directly used to measure the term-goodness of a term w with respect to a class c.

3.2.2 New Term-Category Dependency Measure $R_{w,c}$

In our research, we found the feature selection method CHI does not fully explore all the information provided by the χ^2 term-category independence test. We will use an example to point out where the problem is, and propose a new term-category dependency measure, denoted by $R_{w,c}$, to fix this problem.

	c	$\neg c$	$\sum$
w'	60	320	380
$\neg w'$	40	80	120
$\sum$	100	400	500

Table 3.2: Another 2×2 term-category contingency table

Example 2: Let's compare Table 3.1 and Table 3.2. Using Equations 3.1 and 3.2, we can find both tables produce the same χ^2 statistic with $\chi^2_{w,c} = \chi^2_{w',c} = 17.61$. This is interesting because the two terms, w and w', actually have quite different distributions in c and $\neg c$.

Based on Equation 3.4 and Table 3.1, we can see that there is positive dependency between w and c because $40/100 = 2/5$ of the documents in c contain w and $40/120 = 1/3$ of the documents contain w are in c. That means, w is a typical word in category c, and w is relevant to c. On the other hand, as shown in Table 3.2, it is not clear whether there is positive dependency between w' and c, because even though there are $60/100 = 3/5$ of the documents in c contain w', only $60/380 = 3/19$ of the documents containing w' are in c. In contrast, most documents in $\neg c$ contain w'. Therefore, it is hard to believe that w' is relevant to the category c. Actually, we can say that there is negative dependency between w' and c.

The second example shows that using only the χ^2 statistic based on Equation 3.4 might make many errors in estimating how much a term is relevant to a category. To address this problem, we define our criteria for the relevancy of a term w to a category c as: 1) $\chi^2_{w,c}$ should

be large, and 2) there should be some positive dependency between w and c. To evaluate the positive dependency between a term and a category, we introduce a new measure, $R_{w,c}$, defined as:

$$R_{w,c} = \frac{O(w,c)}{E(w,c)} \tag{3.5}$$

As $R_{w,c}$ is the ratio between $O(w,c)$ and $E(w,c)$, if there is no dependency between the term w and the category c (i.e., $\chi^2_{w,c}$ is not statistically significant), then $R_{w,c}$ should be close to 1. If there is positive dependency, then the observed frequency should be larger than the expected frequency, hence $R_{w,c}$ should be larger than 1. If there is negative dependency, $R_{w,c}$ should be smaller than 1. Only when $\chi^2_{w,c}$ is statistically significant and $R_{w,c}$ is larger than 1, we estimate that the term w is relevant to the category c. Using Equation 3.5, we get $R_{w,c} = 1.67$ for Table 3.1 and $R_{w',c} = 0.79$ for Table 3.2. Based on our criteria, the term w is relevant to the category c, while the term w' is irrelevant, which is a reasonable estimation.

From Equations 3.2 and 3.5, we can see the following relationship between $\chi^2_{w,c}$ and $R_{w,c}$: the farther $R_{w,c}$ is from 1, either negatively or positively, the bigger is its contribution to $\chi^2_{w,c}$. $\chi^2_{w,c}$ is a summary of the whole contingency table and just tells if there is dependency between a term and a category in distribution. But it cannot tell whether the dependency is positive or negative. On the other hand, $R_{w,c}$ tells the dependency more accurately. However, we still need to use $\chi^2_{w,c}$ to evaluate the dependency because our hypothesis test is based on the theoretical χ^2 distribution. By combining $\chi^2_{w,c}$ and $R_{w,c}$, we can provide better information about the relationship between a term and a category.

3.2.3 New Feature Selection Method CHIR

Recall that the feature selection method CHI uses Equation 3.3 or 3.4 as the term-goodness measure to select the terms to be used for text categorization. Given n labeled documents falling into m categories, the steps of CHI to select p terms are as follows after prepressing:

1. For each distinct term in the corpus, calculate its χ^2 statistic value by using Equation 3.3 or 3.4.

2. Sort the terms in descending order of their χ^2 statistic values.

3. Select the top p terms from the list.

As we mentioned in Section 3.2.1, Equation 3.4 performs better in text categorization, so we will use Equation 3.4 to calculate the χ^2 statistic value for terms. In this way, CHI selects only the terms, each of which has strong dependency on some category, no matter the dependency is positive or negative.

	c_1	c_2	c_3
w_1	d_7	d_1, d_2, d_3, d_5	
w_2	d_6, d_7	d_1, d_2, d_3, d_5	
w_3	d_6, d_7	d_1, d_2, d_5	
w_4		d_2, d_3	
w_5			d_4

Table 3.3: 7 documents in 3 categories with 5 terms

Example 3: Let's consider a set of 7 labeled documents, $\{d_1, d_2, \ldots, d_7\}$, falling into 3 categories, $\{c_1, c_2, c_3\}$, as: $c_1 = \{d_6, d_7\}$, $c_2 = \{d_1, d_2, d_3, d_5\}$ and $c_3 = \{d_4\}$. There are total 5 distinct terms, $\{w_1, w_2, \ldots, w_5\}$, in the corpus, and the details are shown in Table 3.3. By following the steps of CHI, the χ^2 statistic values of the terms are calculated as shown in Table 3.4, where the maximum χ^2 statistic value of each term is shown in bold. By listing the terms in descending order of their maximum χ^2 statistic values, we can obtain $(w_5, w_2, w_1, w_3, w_4)$. If we select the top 3 terms from this list, $\{w_5, w_2, w_1\}$ will be chosen. However, this selection has some serious problems. First, w_2 is selected because it shows strong dependency on c_3, but actually w_2 does not occur in any document in c_3. The strong dependency between w_2 and c_3, which is shown by the χ^2 statistic, is their negative dependency. Second, w_4 is not selected even though it is a good feature for c_2, as we can see that w_4 has strong dependency on c_2 in Table 3.3. This example shows that CHI method can remove the terms which are quite relevant to a category and does not provide enough detail information about the relationship between the selected terms and the corresponding

categories. To solve these problems, we propose a new feature selection method, named CHIR.

	c_1		c_2		c_3	
	$\chi^2_{w_i,c_1}$	R_{w_i,c_1}	$\chi^2_{w_i,c_2}$	R_{w_i,c_2}	$\chi^2_{w_i,c_3}$	R_{w_i,c_3}
w_1	0.630	0.700	**3.733**	1.400	2.917	0
w_2	0.467	1.167	*1.556*	1.167	**7.000**	0
w_3	*1.120*	1.400	0.058	1.050	**2.917**	0
w_4	1.120	0	**2.100**	1.750	0.467	0
w_5	0.467	0	1.556	0	**7.000**	7.000

Table 3.4: χ^2 statistic and $R_{w,c}$ values for 5 terms

Our new CHIR method enhances the CHI method by adding one step for checking the $R_{w,c}$ values. This step makes sure that χ^2 statistic value assigned to each term represents only the positive term-category dependency. In other words, CHIR selects the terms which are relevant to categories and removes the irrelevant terms. The detail steps of CHIR to select p terms are as follows:

1. For each distinct term w_i,
   (a) Calculate its $\chi^2_{w_i,c_j}$ value for each category c_j by using Equation 3.2.
   (b) Rank all $\chi^2_{w_i,c_j}$ values in descending order on a list L.
   (c) Select the category c_j which has the largest $\chi^2_{w_i,c_j}$ value, and check the corresponding R_{w_i,c_j} value (by using Equation 3.5) to determine whether its dependency is positive or negative.
      i. If the dependency is positive, assign the $\chi^2_{w_i,c_j}$ value as the χ^2 statistic value of w_i.
      ii. If the dependency is negative, remove $\chi^2_{w_i,c_j}$ from L.
   (d) Repeat Step 1c until w_i has its χ^2 statistic value assigned.
2. Sort the terms in descending order of their χ^2 statistic values on a list L'.

3. Select the top p terms from L'.

For the term w_2 in Example 3, as shown in Table 3.4, even though $\chi^2_{w_2,c_3} = 7$ is the largest among its χ^2 statistic values, the corresponding $R_{w_2,c_3} = 0$ shows that w_2 has negative dependency on c_3. This is confirmed by the fact that w_2 never occurs in c_3 (see Table 3.3). Thus, the χ^2 statistic value of w_2 should be 1.556 in our CHIR method. Similarly, another term w_3 also has a new χ^2 statistic value 1.12 instead of 2.917. As a result, the new list of terms becomes $(w_5, w_1, w_4, w_2, w_3)$, and if we select the top 3 terms, $\{w_5, w_1, w_4\}$ will be selected. In Table 3.3, we can see that these 3 terms are relevant to the corresponding categories, respectively, and they are better than the 3 terms, $\{w_5, w_2, w_1\}$, selected by CHI.

3.3 Text Clustering with Feature Selection (TCFS) Algorithm

As we discussed in Section 3.1, it is challenging to apply supervised feature selection methods directly to text clustering because of the lack of the class label information of documents. However, it is not impossible to adopt the supervised feature selection in text clustering because the clusters obtained during the clustering process can provide valuable information for the feature selection. The well-known Expectation-Maximization (EM) algorithm [14] provides us a framework to combine the text clustering and supervised feature selection methods. Based on the EM algorithm, we propose a new text clustering algorithm with feature selection, named TCFS, which iteratively performs the clustering and the supervised feature selection alternately. The whole process of TCFS is basically a learning process. While the information of the clusters is utilized to find better features (i.e., terms), the quality of the clustering result is improved by reducing the weight of irrelevant features. As the TCFS algorithm converges, both a good clustering result and an informative feature subset are obtained.

Recall that we defined the problem of text clustering as the grouping of documents with similar topics into a cluster. By using the EM algorithm, we assume that each cluster of

documents has a Gaussian distribution of terms. That means, a corpus with k clusters is considered as a mixture of k Gaussian distributions. Given the parameters and our Gaussian model, we maximize the likelihood of our data set. The maximum likelihood represents how well our Gaussian model fits the data set. In this case, the clustering criterion for our TCFS algorithm is the maximum likelihood, and a natural criterion for the feature selection also is the maximum likelihood.

For text clustering, the likelihood function $p(S|\theta)$, which represents the probability that a set S of n documents are grouped into k clusters when the parameter vector θ of the Gaussian model is given, can be written as:

$$p(S|\theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} p(c_j|\theta) p(d_i|c_j, \theta) \tag{3.6}$$

where c_j is the jth cluster, $p(c_j|\theta)$ is the prior probability of the cluster c_j for given θ, and $p(d_i|c_j, \theta)$ is the prior probability of the document d_i in the cluster c_j for given θ. In the EM framework for text clustering, the terms in documents are assumed to be conditionally independent of each other, and the likelihood function can be rewritten as:

$$p(S|\theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} (p(c_j|\theta) \prod_{w \in d_i} p(w|c_j, \theta)) \tag{3.7}$$

where $p(w|c_j, \theta)$ is the conditional probability of the term w in the cluster c_j. As we discussed in Section 3.1, not all the terms (features) are equally relevant to the clusters, so $p(w|c_j, \theta)$ can be represented as:

$$p(w|c_j, \theta) = p_r(w) p(w \text{ is relevant}|c_j, \theta) + (1 - p_r(w)) p(w \text{ is irrelevant}|c_j, \theta) \tag{3.8}$$

where $p_r(w)$ is defined as the probability that the term w is relevant to the clusters, which is determined by performing the feature selection [40]. EM produces a sequence of estimates $\{\hat{\theta}(i) \text{ and } \hat{p_r}(i), i = 0, 1, 2, \ldots\}$ by using the following two steps:

1. Expectation step (E-step): $\hat{p_r}(i+1) = E(p_r|S, \hat{\theta}(i))$

2. Maximization step (M-step): $\hat{\theta}(i+1) = \arg \max_{\theta} p(S|\theta, \hat{p_r}(i))$

In fact, the E-step is to perform the supervised feature selection by calculating the expected feature relevancy for the current clustering result given, and the M-step is to re-cluster the data set in the new feature space.

In our TCFS algorithm, we can adopt the proposed feature selection method CHIR into the framework of EM. In the E-step, we use the CHIR method to estimate the relevancy of each term to the clusters, then the probability of the term relevancy, $p_r(w)$, is set to either 1 or f, where f is a predetermined factor in (0,1). That means, if a term relevancy score calculated using the information obtained at each iteration is higher than a predefined threshold value, the term is treated as a relevant one and untouched in the feature space. Otherwise, the term is treated as irrelevant, and its weight is reduced by the factor of f; i.e., its new weight is obtained by multiplying the previous weight with f. The iterative feature selection method proposed in [40] simply removes irrelevant terms based on the relevancy score calculated at each iteration. The reason why we do not simply remove these irrelevant terms from the feature space is that the information utilized by the supervised feature selection method CHIR is not the real (i.e., final) class label information of the documents. With the convergence of EM iterations, we are getting closer to the real class label information. At the end of the iterations, we could select the terms with high relevancy scores into the desired feature subset. Since the k-means clustering algorithm is considered as an extension of EM for the hard threshold case [6], it could be used in the M-step to cluster the documents in the new feature space.

The detail steps of our TCFS algorithm are as follows:

1. Perform a clustering algorithm, such as k-means, on the data set and get initial clusters.

2. Perform a feature selection method, such as CHIR, on the data set by using the current clustering result as the class label information of the documents. The selected features (i.e., terms) remain untouched in the feature space, but the weights of the unselected features are reduced by f, where f is a predetermined factor in (0,1). (E-step)

3. Perform the clustering algorithm again on the data set in the new feature space. (M-

step)

4. Repeat Steps 2 and 3 until convergence.

3.4 Experimental Results

In this section, first the proposed feature selection method CHIR is compared with CHI in terms of cluster cohesiveness. Then, the proposed text clustering algorithm TCFS with CHIR is compared with the k-means algorithm, the k-means algorithm with the Term Strength (TS) feature selection method, and TCFS with CHI. The experimental results show that TCFS with CHIR has the best clustering performance in terms of the accuracy of the clustering result.

3.4.1 Data Sets

We used 7 test data sets from two different types of text databases, which have been widely used by the researchers in the information retrieval area. Three data sets, denoted by CISI, CACM and MED, are extracted from the CISI, CACM and MEDLINE abstracts, respectively, which are included in the Classic database [9]. Additional four data sets, denoted by EXC, ORG, PEO and TOP, are from the EXCHANGES, ORGS, PEOPLE and TOPICS category sets of the Reuters-21578 Distribution 1.0 [55].

Each document of the test data sets has been pre-classified into one unique class. But, this information was hidden during the clustering processes and just used to evaluate the clustering accuracy of each clustering algorithm. Before the experiments, the stop words removal and the stemming were performed as preprocessing steps on the data sets. Table 3.5 summarizes the characteristics of all the data sets used for our experiments.

Data Set	Num. of Doc.	Num. of Classes	Min. Class Size	Max. Class Size	Num. of Unique Terms	Avg. Doc. Length	Avg. Pairwise Similarity by cosine
CISI	148	3	24	78	1,935	67	0.04
CACM	170	5	26	51	1,260	56	0.04
MED	287	9	26	39	4,255	77	0.02
EXC	334	7	28	97	3,258	67	0.03
ORG	733	9	20	349	6,172	138	0.03
PEO	694	15	11	143	5,046	102	0.04
TOP	2279	7	23	750	10,719	113	0.03

Table 3.5: Summary of data sets

3.4.2 Evaluation Methods

3.4.2.1 An Evaluation Method of the Feature Selection

We used the cohesiveness of clusters to measure the performance of CHI and CHIR feature selection methods. The cohesiveness value of a cluster can be computed by using the weighed sum of the similarities between documents in the cluster as follows [59]:

$$Cohesiveness(C) = \frac{1}{|C|^2} \sum_{d \in C, d' \in C} cosine(d', d) = \frac{1}{|C|} \sum_{d \in C} d \bullet \frac{1}{|C|} \sum_{d \in C} d = c \bullet c = \|c\|^2 \quad (3.9)$$

where C represents the cluster, c is the centroid of the cluster, d and d' are documents in the cluster, and the *cosine* function is used to measure the pairwise similarity between documents. From Equation 3.9, we can see that the square of the length of the centroid vector is the average pairwise similarity between two documents in the cluster. This also includes the similarity of each document with itself, which is one. We applied the CHIR and CHI feature selection methods to the same clusters obtained by the k-means algorithm, respectively, and compared the cohesiveness values of each cluster to compare the two methods. A good feature selection method should eliminate irrelevant features while obtaining large cohesiveness values of the clusters.

3.4.2.2 Evaluation Methods of the Text Clustering

We used the *F-measure* and the *entropy* to evaluate the accuracy of the clustering algorithms. The F-measure is a harmonic combination of the *precision* and *recall* values used in information retrieval [56]. Since our data sets were prepared as described in Section 3.4.1, each cluster obtained can be considered as the result of a query, whereas each pre-classified set of documents can be considered as the desired set of documents for that query. Thus, we can calculate the precision $P(i, j)$ and recall $R(i, j)$ of each cluster j for each class i.

If n_i is the number of the members of class i, n_j is the number of the members of cluster j, and n_{ij} is the number of the members of class i in cluster j, then $P(i, j)$ and $R(i, j)$ can be defined as:

$$R(i, j) = \frac{n_{ij}}{n_i} \tag{3.10}$$

$$P(i, j) = \frac{n_{ij}}{n_j} \tag{3.11}$$

The corresponding F-measure $F(i, j)$ is defined as:

$$F(i, j) = \frac{2 * R(i, j) * P(i, j)}{R(i, j) + P(i, j)} \tag{3.12}$$

Then, the F-measure for the whole clustering result is defined as

$$F = \sum_i \frac{n_i}{n} \max_j (F(i, j)) \tag{3.13}$$

where n is the total number of documents in the data set. In general, the larger the F-measure is, the better the clustering result is [59].

The entropy is an external quality measure of a clustering algorithm. By comparing the clustering result with known classes, this measure shows how good it is. For each cluster of the clustering result, the class distribution of the data points is calculated first by computing the probability that a member of cluster j belongs to class i, denoted by p_{ij}. Then, the entropy of each cluster j is calculated as:

$$E_j = -\sum_i p_{ij} \log p_{ij} \tag{3.14}$$

The total entropy of all the clusters is the sum of the entropies of the clusters weighted by their sizes:

$$E = \sum_{j=1}^{k} \frac{n_j E_j}{n} \tag{3.15}$$

where n_j is the size of cluster j, n is the total number of documents, and k is the number of clusters. The smaller the entropy is, the purer the produced clusters are.

3.4.3 Comparison of the Feature Selection Methods

In order to compare our new feature selection method CHIR with CHI, the k-means algorithm was used first to cluster each test data set. Then, we applied both CHIR and CHI to the clusters of documents and checked the change of the cohesiveness value of each cluster. For our implementation, we used the vector-space model to represent documents. In this model, each document is represented by a vector of the frequencies of unique terms within the document. Each document vector is normalized to have a unit length for the comparison of documents with different lengths. In our experiment, the percentage of the selected features (i.e., terms) was varied from 5% to 90%. At each round of feature selection, the unselected terms were simply removed from the document vectors, then the document vectors are re-normalized. For comparison, the cohesiveness value of every cluster was calculated and recorded.

With fewer terms left in the feature space, the cohesiveness value of the cluster increases because sparse features are removed and documents become more similar to each other. When the feature selection method selects an appropriate feature subset, which represents the cluster better than other subsets, the cohesiveness value is larger. For example, for the cluster $C1$ of the CISI data set, when CHIR is performed with 5% of the features selected, the cohesiveness value of this cluster is 0.221. When CHI is performed, the cohesiveness value of $C1$ is 0.203. This result suggests that CHIR removes irrelevant features better than CHI.

We applied the k-means algorithm on the CISI and EXC data sets and obtained 3 and 7 clusters, respectively. Then, we performed both CHIR and CHI on these clusters with differ-

ent percentages of feature selection. The cohesiveness values of the clusters were compared, and a part of the results are shown in Figures 3.1 and 3.2. Our experimental results show that CHIR consistently outperforms CHI in terms of increasing the cohesiveness values of the clusters.

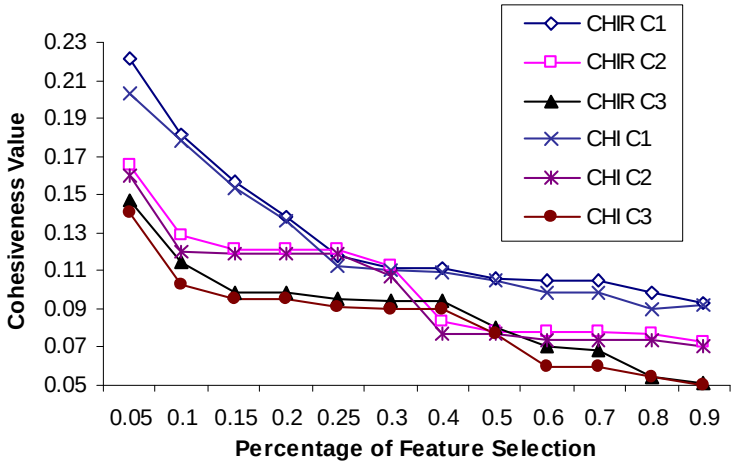


Figure 3.1: Cohesiveness values of three clusters of the CISI data set

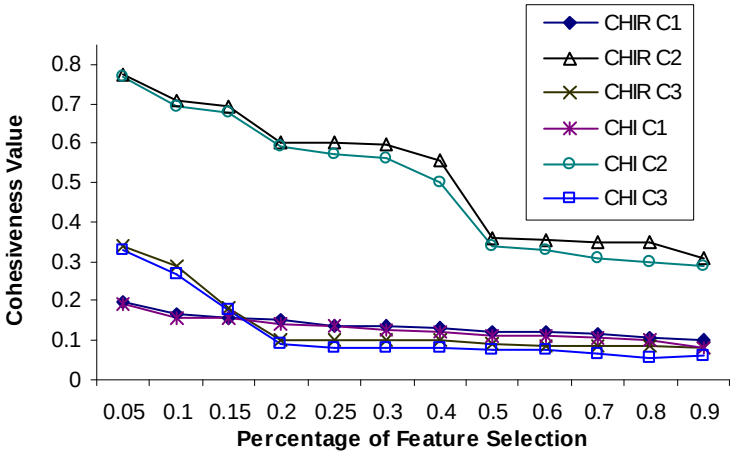


Figure 3.2: Cohesiveness values of three Clusters of the EXC data set

3.4.4 Comparison of the Clustering Algorithms

In [40], supervised and unsupervised feature selection methods are evaluated in terms of improving the clustering quality by conducting experiments in the case that the class labels

of documents are available for the feature selection. As a preprocessing step of text clustering, the Term Strength (TS) feature selection method was reported as the best among the unsupervised feature selection methods evaluated in [40].

TS was originally proposed and evaluated for the vocabulary reduction in text retrieval [65], and later applied to text categorization [67]. It is computed based on the conditional probability that a term occurs in the second halves of a pair of related documents given that it occurs in their first halves:

$$TS(w) = p(w \in d_j \mid w \in d_i), d_i \in D, d_j \in D \text{ and } similarity(d_i, d_j) \geq \delta \quad (3.16)$$

$$\approx \frac{\text{\# of pairs in which } w \text{ co-occurs in both documents}}{\text{\# of pairs in which } w \text{ occurs in the first document}} \quad (3.17)$$

where δ is the parameter to determine the related document pairs. Since we need to calculate the similarity of each document pair, the time complexity of calculating TS is quadratic of the number of documents. As the class label information is not required, TS can be used for the term reduction in text clustering. In this case, terms are ordered in descending order of their TS values, and then a certain percentage of them are selected from the top to be used for clustering.

In our experiments, we compared the clustering accuracies of TCFS with CHIR, TCFS with CHI, k-means, and k-means with TS. When k-means is combined with the feature selection method TS, TS was performed first as a preprocessing step, then k-means was applied to the data set with the selected terms. δ was set to 0.1 for TS, and the percentage of feature selection was varied in the range of [5%, 90%] for TS, CHI, and CHIR.

When we performed TCFS on the data sets, at each iteration, a certain percentage of features was selected based on the feature selection method chosen — CHIR or CHI. As described in Section 3.3, the relevancy of each term to the clusters is estimated based on the information obtained at each iteration. The probability of the term relevancy is set to either 1 or f, where f is a predetermined factor in (0,1). At each iteration, the weights of the irrelevant terms are reduced by f in the feature space. In our experiments, we set f as 0.5.

Figures 3.3, 3.4, 3.5 and 3.6 show the results of running TCFS with CHIR, TCFS with CHI, k-means, and k-means with TS on the EXC and CISI data sets. In Figures 3.3 and 3.4, we can see that TCFS with either CHIR or CHI can achieve a better F-measure than k-means with TS regardless of the percentage of feature selection. In TCFS, when more terms are removed, the clustering result is better. The performance of k-means with TS is not consistent as the percentage of feature selection changes. For example, when the top 25% terms are selected by TS, the F-measure is 0.3846, which is even lower than the case of simply conducting k-means. This result shows that TS does not always select an appropriate feature subset. In other words, TS may remove some relevant terms, while keeping some irrelevant ones. The entropy values shown in Figures 3.5 and 3.6 suggest the same result.

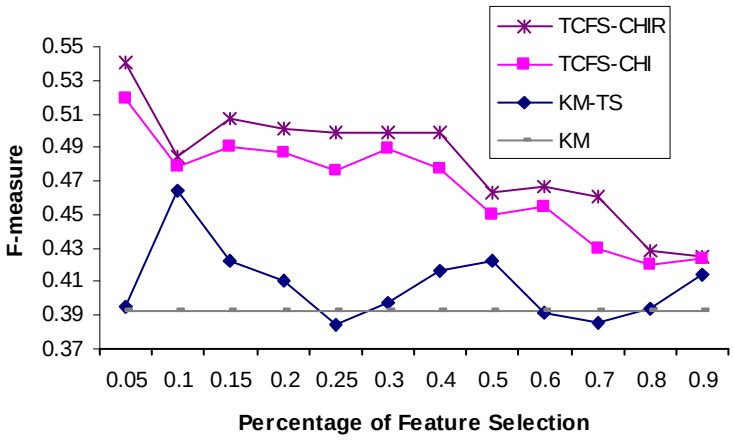


Figure 3.3: F-measure of the clusters of the EXC data set

Tables 3.6 and 3.7 show the F-measure and entropy values of the clusters obtained by running four clustering algorithms on 7 data sets. For TCFS with either CHIR or CHI, the top 15% of the terms were selected at each iteration; and the top 15% of the terms were selected once for k-means with TS. On the PEO data set, all four clustering algorithms perform equally well. This could be explained by the fact that there are only few irrelevant terms in the feature space of this data set. Except for this data set, TCFS with CHIR performs better than other three algorithms.

Our experimental results demonstrate that the class label information obtained during

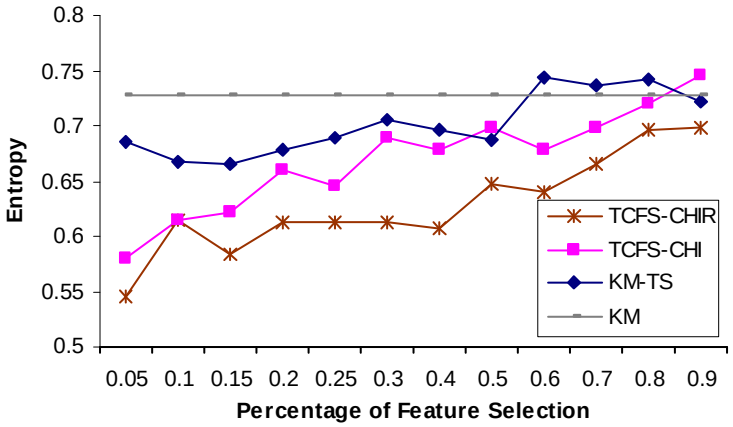


Figure 3.4: Entropy value of the clusters of the EXC data set

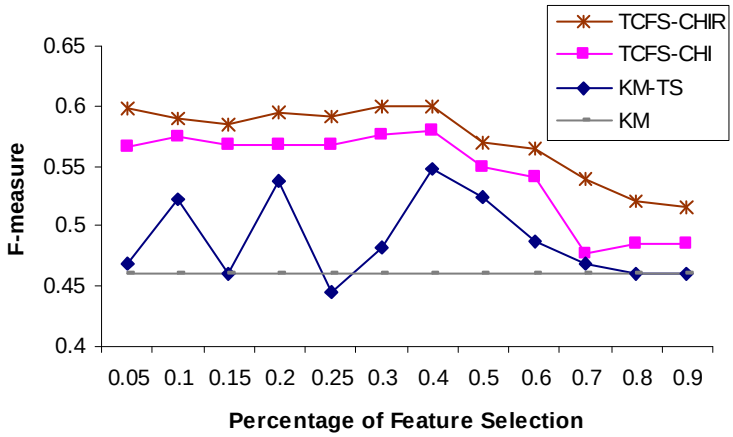


Figure 3.5: F-measure of the clusters of the CISI data set

the clustering process by performing a supervised feature selection method can be utilized to improve the clustering accuracy. Moreover, our proposed TCFS with CHIR text clustering algorithm generats much more accurate clusters than existing algorithms for different data sets.

3.5 Conclusions

In this chapter, we introduced a new term-category dependency measure, denoted by $R_{w,c}$ which can tell whether the dependency is positive or negative. Based on the χ^2 statistic and $R_{w,c}$, we proposed a new supervised feature selection method CHIR. CHIR selects the terms

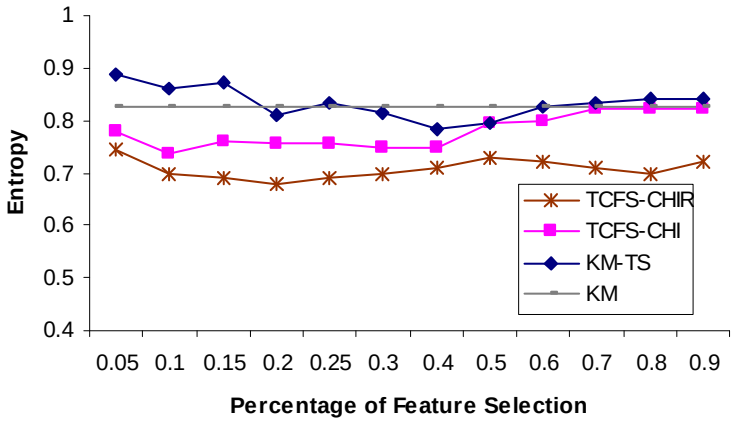


Figure 3.6: Entropy values of the clusters of the CISI data set

Data Set	KM	KM with TS	TCFS with CHI	TCFS with CHIR
CISI	0.461	0.460	0.513	0.585
CACM	0.552	0.561	0.632	0.695
MED	0.645	0.680	0.692	0.694
EXC	0.393	0.422	0.490	0.507
ORG	0.569	0.605	0.630	0.645
PEO	0.451	0.442	0.450	0.456
TOP	0.688	0.701	0.723	0.740

Table 3.6: F-measure (15% feature selection)

that are relevant to the categories by utilizing the known class label information. CHIR can be used for text categorization, text summarization, and ontology creation.

We also proposed a new text clustering algorithm TCFS which performs a supervised feature selection, such as CHIR and CHI, during the clustering process. The cluster label information obtained during the clustering process is utilized as the known class label information for the feature selection. The selected features improve the quality of clustering iteratively, and as the clustering process converges, the clustering result has higher accuracy. TCFS with CHIR has been compared with other clustering and feature selection algorithms, such as k-means, k-means with the Term Strength (TS) feature selection method, and TCFS with CHI. Our experimental results show that TCFS with CHIR has better performance

Data Set	KM	KM with TS	TCFS with CHI	TCFS with CHIR
CISI	0.824	0.873	0.793	0.690
CACM	0.755	0.740	0.723	0.710
MED	0.444	0.421	0.372	0.366
EXC	0.728	0.665	0.622	0.557
ORG	0.304	0.300	0.281	0.275
PEO	0.504	0.469	0.556	0.504
TOP	0.343	0.303	0.298	0.270

Table 3.7: Entropy values (15% feature selection)

than other algorithms in terms of the clustering accuracy for different test data sets.

3.6 Future Works

3.6.1 Building Ontology

The semantic knowledge in the form of ontology is widely used to facilitate visualization, summarization, and maintenance of large document repositories. Therefore, ontology has been acknowledged as an important type of metadata, and many researches focus on ontology building [60]. However, in many cases, the ontology building is still conducted manually, which is time-consuming and labor-intensive. Thus, semiautomatic or fully automatic ontology building becomes very attractive. In [60], it has been shown how to extract a set of candidate concept words for a domain ontology by using some feature selection techniques. Unlike the TFIDF (term frequency inverse document frequency) framework, which reflects only the frequency-based importance of a word, the statistical or information theoretical measures used in the feature selection, such as χ^2 statistics, Markov blanket [34], information gain [54], and mutual information [11], can reflect the dependency between a word and a category. In [60], experimental results show that χ^2 statistic consistently outperform other measures. We will propose a method to automatically generate concept words for ontology building by combining χ^2 statistic and $R_{w,c}$.

3.6.2 Incremental Text Categorization

The naive Bayes classifier has been very popular in text categorization [43]. Unlike other complex classifiers, such as the Support Vector Machine (SVM) [32] and the Maximum Entropy Model [50], which require a big computation overhead in training, the naive Bayes classifier directly uses the frequency information of terms to calculate its model parameters. Thus, the naive Bayes classifier is probably the best choice for incremental text categorization, because we just need to store the required frequency information and update it when new labeled documents are provided. Moreover, it has been shown that the performance of the naive Bayes classifier can be comparable to that of other sophisticated classifiers, even though it assumes the independence of features, which may be too strong. We plan to develop an enhanced multinomial naive Bayes classifier by using the χ^2 statistics and $R_{w,c}$.

Chapter 4

Text Document Clustering Based on Neighbors

4.1 Introduction

How to explore and utilize the huge amount of text documents is a major question in the areas of information retrieval and text mining. Document clustering is one of the most important text mining methods, which are developed to help users to effectively navigate, summarize, and organize text documents. By organizing a large amount of documents into a number of meaningful clusters, document clustering can be used to browse a collection of documents or organize the results returned by a search engine in response to a user's query. It can significantly improve the precision and recall in information retrieval systems [56], and it is an efficient way to find the nearest neighbors of a document [8]. The problem of document clustering is generally defined as follows: given a set of documents, we would like to partition them into a predetermined or an automatically derived number of clusters, such that the documents assigned to each cluster are more similar to each other than the documents assigned to the other clusters. In other words, the documents in one cluster share the same topic, and the documents in different clusters represent different topics.

There are two general categories of clustering methods: agglomerative hierarchical and partitioning methods. In the previous researches, both of them are applied to text clustering. Agglomerative hierarchical clustering (AHC) algorithms initially treat each document as a

cluster, use different kinds of distance functions to compute the similarity between all pairs of clusters, and then merge the closest pair [30]. This merging step is repeated until the desired number of clusters is obtained. Comparing with the bottom-up method of AHC algorithms, the family of k-means algorithms [13, 33, 36], which belong to the partitioning category, create a one-level partitioning of the documents. The k-means algorithm is based on the idea that a centroid can represent a cluster. After selecting k initial centroids, each document is assigned to a cluster based on a distance measure, then k centroids are recalculated. This step is repeated until an optimal set of k clusters are obtained based on a heuristic function.

For document clustering, Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [30] of AHC is reported to be the most accurate one in its category. Bisecting k-means is reported to outperform the k-means as well as the agglomerative approach in terms of accuracy and efficiency. In bisecting k-means algorithm, initially the whole data set is treated as a cluster. Based on a rule, it selects a cluster to split into two by using the basic k-means algorithm. This bisecting step is repeated until the desired number of clusters is obtained. Generally speaking, the partitional clustering algorithms are well-suited for the clustering of large document databases due to their relatively low computational requirements and high quality.

A key characteristic of the partitional clustering algorithms is that a global criterion function is used, whose optimization drives the entire clustering process. The goal of this criterion function is to optimize different aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations. A well-known similarity function is the *cosine* function, which is widely used in document clustering algorithms and is reported performing very well [59]. The *cosine* function can measure how similar two documents are, and when it is used in the family of k-means algorithms, a document is assigned to a cluster with the most similar cluster centroid in an effort to maximize the intra-cluster similarity.

Since the *cosine* function measures the similarity of two documents, only the pairwise similarity is considered when we determine whether a document is assigned to a cluster or not. However, when the clusters are not so well-separated, partitioning them just based on

the pairwise similarity is not good enough because some documents in different clusters may be similar to each other.

The concepts of *neighbors* and *link* are proposed in [26]. When two documents are similar enough, they are considered as *neighbors* of each other. Every document can have a set of *neighbors* in the data set for a certain similarity threshold. The *link* represents the number of *common neighbors* between two data points [26]. For example, $link(p_i, p_j)$ is the number of *common neighbors* of two data points: p_i and p_j. In [26], the *link* function is used in an agglomerative algorithm for clustering data with categorical attributes and obtained better clusters than traditional algorithms.

Each text document can be viewed as a tuple with boolean attributes, where each attribute corresponds to a unique term. An attribute value is true if the corresponding term exists in the document. Since a boolean attribute is a special case of categorical attribute, we could treat documents as data with categorical attributes. With this assumption, the concepts of *neighbors* and *link* could provide valuable information about the documents in the clustering process. We believe that the intra-cluster similarity should be measured not only between the documents and the centroid, but also between their *neighbors*. The *link* function can be used to enhance the evaluation of the closeness between documents because it takes the information of surrounding documents into consideration.

The information about *neighbors* and *link* for all documents in a data set could be represented by a *neighbor matrix* [26]. In this chapter, we propose the applications of the *neighbor matrix* along with the *cosine* function in different aspects of the k-means and bisecting k-means algorithms for clustering documents. The family of k-means algorithms have two phases: initial clustering and cluster refinement [71]. The initial clustering phase is the process of choosing a desired number (k) of initial centroids and assigning documents to their closest centroids to form initial partitions. The cluster refinement phase is the optimization process which adjusts the partitions by repeatedly calculating the new cluster centroids based on the documents assigned to them and reassigning documents.

First, we propose a new method to select the initial centroids. It is well known that

the performance of the family of k-means algorithms is very sensitive to the selection of the initial centroids [30]. It's very important that the initial centroids are distributed well enough to attract sufficient nearby, topically related documents [36]. Our selection of the initial centroids is based on the evaluation of three values: the pairwise similarity value calculated by the *cosine* function, the *link* function value, and the number of neighbors of documents in the data set. This combination helps us to find a group of initial centroids with high quality.

Second, we explore a new clustering criterion function to determine the assignment of each document to a cluster during the clustering refinement phase. This criterion function is composed of the *cosine* and *link* functions. We believe that, besides the pairwise similarity, involving documents in the neighborhood can improve the accuracy of closeness measurement between documents.

Third, we create a new heuristic function for the bisecting k-means algorithm to select a cluster to split. Unlike the k-means algorithm which splits the whole data set is split into k subclusters at each iteration step, the bisecting k-means algorithm splits only one existing cluster into 2 subclusters. Our selection of a cluster to split is based on the neighbors of the centroids instead of the size of clusters because the concept of *neighbors* provides more information about the intra-similarity of clusters.

We evaluated the performance of our proposed clustering algorithms on various real-life data sets, and the experimental results demonstrated very significant improvement in the accuracy of clustering.

The rest of this chapter is organized as follows: In Section 4.2, we will review the vector space model of documents, the *cosine* function, the concept of neighbors, the *link* function, and the k-means and the bisecting k-means algorithms. In Section 4.3, the applications of the *neighbor matrix* in the k-means and bisecting k-means algorithms are described in details. In Section 4.4, the experimental results of our clustering algorithms are compared with those of original algorithms in terms of the accuracy of clustering. Section 4.5 contains some conclusions.

4.2 Background

4.2.1 Vector Space Model of Text Documents

For most existing document clustering algorithms, documents are represented by using the vector space model [56]. In this model, each document d is considered to be a vector in the term-space and represented by the *term-frequency* (TF) vector:

$$d_{tf} = [tf_1, tf_2, \ldots, tf_m] \tag{4.1}$$

where tf_i is the frequency of the ith term in the document, and m is the dimension of the text database, which is the total number of unique terms. Normally there are several preprocessing steps, including the stop words removal and the stemming on the documents. A widely used refinement to this model is to weight each term based on its *inverse document frequency* (IDF) in the document collection. The idea is that terms appearing frequently in many documents have limited discrimination power, so they need to be deemphasized [56]. This is commonly done by multiplying the frequency of each term i by $log(N/df_i)$, where N is the total number of documents in the collection, and df_i is the number of documents that contain the ith term (i.e., document frequency). The following is the *tf-idf* representation of the document:

$$d_{tf-idf} = [tf_1 log(N/df_1), tf_2 log(N/df_2), \ldots, tf_m log(N/df_m)] \tag{4.2}$$

To account for documents of different lengths, the length of each document vector is normalized so that it is of unit length ($\|d_{df-idf}\| = 1$), and each document is a vector in the unit hypersphere. In the rest of the chapter, we assume that this normalized vector space model weighted by *tf-idf* is used to represent documents during the clustering.

Given a set C_j of documents and their corresponding vector representations, the *centroid* vector c_j is defined as

$$c_j = \frac{1}{|C_j|} \sum_{d \in C_j} d \tag{4.3}$$

where d is the document vector and $|C_j|$ is the number of documents in the set C_j. The centroid is in fact the vector obtained by averaging the weights of every term in the documents

of C_j. It should be noted that even though the document vector d is of unit length, the centroid vector c_j is not necessarily of unit length.

4.2.2 Cosine Similarity Measure

For the problem of clustering documents, there are different criterion functions available. The most commonly used is the *cosine* function [56]. For two documents d_i and d_j, the similarity between them can be calculated as:

$$cos(d_i, d_j) = \frac{d_i^t d_j}{\|d_i\| \|d_j\|} \tag{4.4}$$

Since the document vectors are of unit length, the above equation simplifies to:

$$cos(d_i, d_j) = d_i^t d_j \tag{4.5}$$

The *cosine* value is 1 when two documents are identical, and 0 if there is nothing in common between them (i.e., their document vectors are orthogonal to each other).

4.2.3 Neighbors and Link

The *neighbors* of a document d in a data set are those documents that are considered similar to it [26]. Let *sim(d_i, d_j)* be a similarity function capturing the pairwise similarity between two documents d_i and d_j, and have values between 0 and 1 with larger values indicating higher similarity. For a given threshold θ, d_i and d_j are defined as *neighbors* of each other if

$$sim(d_i, d_j) \geq \theta, \text{ with } 0 \leq \theta \leq 1. \tag{4.6}$$

Here θ is a user-defined threshold that can be used to control how similar a pair of documents must be in order to be considered as neighbors of each other. If we use the *cosine* as *sim*, *sim* is 1 for identical documents and 0 for totally dissimilar documents. When θ is set to 1, a document is constrained to be a neighbor to only other identical documents. On the other hand, 0 as the value of θ allows any arbitrary pair of documents to be neighbors. Depending on the desired similarity for the application, the user can choose an appropriate value for θ.

The information about the neighbors of every document in the data set could be represented by a *neighbor matrix*. A *neighbor matrix* for a data set of n documents is an $n \times n$ adjacency matrix M, in which entry $M[i, j]$ is 1 or 0 depending on whether documents d_i and d_j are neighbors or not [26]. The number of neighbors of a document d_i in this data set is the same as the number entries whose values are 1 in the ith of the matrix M. It could be represented as follows:

$$N(d_i) = \sum_{m=1}^{n} M[i, m] * A[m, 1] \tag{4.7}$$

where A is a $n \times 1$ matrix, in which all the entries are 1.

The *link* function, *link(d_i, d_j)*, is defined as the number of common neighbors between d_i and d_j [26]. The value of *link(d_i, d_j)* can be obtained by multiplying the ith row with the jth column of the *neighbor matrix* M:

$$link(d_i, d_j) = \sum_{m=1}^{n} M[i, m] * M[m, j]. \tag{4.8}$$

This definition indicates that if *link(d_i, d_j)* is large, then it is more probable that d_i and d_j are close enough to be in the same cluster.

Since the *cosine* considers only the features of two documents, it is a *local* approach for clustering. Involving the *link* function could be considered as a *global* approach for clustering [26], since it captures the global knowledge of neighbor documents into the relationship between individual pairs of documents. Thus, the *link* function is also a good candidate for measuring the closeness of two documents.

4.2.4 k-means and Bisecting k-means Algorithms for Document Clustering

k-means is a popular algorithm to solve the problem of clustering a data set into k clusters. If the data set contains n documents, $d_1, d_2, \ldots, d_n$, then the clustering is the optimization process of grouping them into k clusters so that the global criterion function

$$\sum_{j=1}^{k} \sum_{i=1}^{n} f(d_i, c_j) \tag{4.9}$$

is either minimized or maximized. c_j represents the centroid of cluster C_j, for $j = 1, \ldots, k$, and $f(d_i, c_j)$ is the clustering criterion function for a document d_i and a centroid c_j. When the *cosine* is used, a document d_i is assigned to the cluster with the most similar centroid c_j, and the global criterion function is maximized as a result. This optimization process is known as a NP-complete problem [23], and k-means algorithm was proposed to provide an approximate solution [28]. The steps of k-means are as follows:

1. Select k initial cluster centroids.
2. For each document of the whole data set, compute the clustering criterion function with each cluster centroid. Assign the document to its best choice. (assignment step)
3. Recalculate k centroids based on the documents assigned to them.
4. Repeat steps 2 and 3 until convergence.

The bisecting k-means [59] is a variant of the k-means algorithm. The key point of this algorithm is that only one cluster is split into two subclusters at each step. This algorithm starts with the whole data set as a single cluster, and its steps are as follows:

1. Select a cluster C_j to split based on a heuristic function.
2. Find 2 subclusters of C_j using the k-means algorithm: (bisecting step)
   (a) Select 2 initial cluster centroids.
   (b) For each document of C_j, compute the clustering criterion function with 2 cluster centroids, and assign the document to its best choice. (assignment step)
   (c) Recalculate 2 centroids based on the documents assigned to them.
   (d) Repeat steps 2b and 2c until convergence.
3. Repeat step 2 I times, and select the split that produces the clustering satisfying the global criterion function.

4. Repeat steps 1, 2 and 3 until k clusters are obtained.

I is the number of iterations for each bisecting step, which is usually specified in advance.

4.3 Applications of the Neighbor Matrix in k-means and Bisecting k-means Algorithms

4.3.1 Initial Centroids Selection Based on the Ranks

The family of k-means algorithms start with an initial partition, and documents are assigned to the clusters iteratively in order to minimize or maximize the value of the global criterion function. It is known that the clustering algorithms based on the iterative process are computationally efficient but often converge to local minima or maxima of the global criterion function. There is no guarantee that those algorithms will reach a global optimization. Since different initial partitions can lead to different final clustering results, starting with a good initial partition is one way to overcome this problem.

For document clustering, an initial partition is formed by specifying a set of k initial centroids first and then assigning each document to the closest centroid. There are three algorithms available for the selection of initial centroids: random, buckshot [13], and fractionation [13]. The random algorithm randomly chooses k documents from the data set as the initial centroids [30]. The buckshot algorithm picks $\sqrt{kn}$ documents randomly from the data set of n documents, and clusters them using a clustering algorithm. The k centroids resulting from this clustering become the initial centroids. The fractionation algorithm splits the documents into buckets of the same size, and the documents within each bucket are clustered. Then these clusters are treated as if they are individual documents, and the whole procedure is repeated until k clusters are obtained. The centroids of the resulting k clusters become the initial centroids.

In this chapter, we propose a new method to select initial centroids based on the *neighbor matrix* and the *cosine* function. Since the documents in one cluster are supposed to be more similar to each other than the documents in different clusters, a good candidate for a

initial centroid should not only be close enough to a certain group of documents but also well separated from other centroids. By setting an appropriate similarity threshold, θ, the number of neighbors of a document in the data set could be used to evaluate how many documents are close enough to it. Since both *cosine* and *link* functions can measure the similarity of two documents, here we use them together to evaluate the dissimilarity of two documents.

First, by checking the neighbor matrix of the data set, we list documents in descending order of the number of their neighbors. In order to find documents which are close enough to a certain group of documents, the top m documents are selected from this list to form a set of initial centroid candidates, denoted by S_m with $m = k + n_{plus}$, where k is the desired number of clusters and n_{plus} is the extra number of candidates selected. Since these m candidates have the most neighbors in the data set, we assume they are more likely the centers of clusters.

For example, let's consider a data set S containing 6 documents, $S = \{d_1, d_2, d_3, d_4, d_5, d_6\}$, whose *neighbor matrix* is as shown in Figure 4.1. When $\theta = 0.3$, $k = 3$ and $n_{plus} = 1$, S_m has four documents: $S_m = \{d_1, d_2, d_3, d_4\}$.

	d_1	d_2	d_3	d_4	d_5	d_6
d_1	1	1	0	1	0	0
d_2	1	1	0	1	0	0
d_3	0	0	1	1	1	0
d_4	1	1	1	1	0	0
d_5	0	0	1	0	1	0
d_6	0	0	0	0	0	1

Figure 4.1: Neighbor matrix (M) of data set S with $\theta = 0.3$

Next, we obtain the *cosine* and *link* values between every pair of documents in S_m and rank them in ascending order of the *cosine* values and the *link* values, respectively. For a pair of documents, d_i and d_j, let's define $rank_{cos(d_i,d_j)}$ be their rank based on the *cosine* values, $rank_{link(d_i,d_j)}$ be their rank based on the *link* values, and $rank_{d_i,d_j}$ be the sum of $rank_{cos(d_i,d_j)}$ and $rank_{link(d_i,d_j)}$. A pair of documents with smaller rank value has a higher rank, and 0 is

the highest rank. The ranks of document pairs are shown in Table 4.1.

d_i, d_j	cos	$rank_{cos}$	$link$	$rank_{link}$	$rank_{d_i,d_j}$
d_1, d_2	0.35	2	3	3	5
d_1, d_3	0.10	1	1	0	1
d_1, d_4	0.40	3	3	3	6
d_2, d_3	0	0	1	0	0
d_2, d_4	0.50	4	3	3	7
d_3, d_4	0.60	5	2	2	7

Table 4.1: Similarity measurement between initial centroid candidates

Initial centroids better be far from each other in order to represent the whole data set. Thus, the document pairs with high ranks could be considered good initial centroid candidates. For the selection of k initial centroids out of m candidates, there are ${}_mC_k$ possible combinations. Each combination, com_k, is a k-subset of S_m, and we calculate the rank value of each combination com_k as:

$$rank_{com_k} = \sum rank_{d_i,d_j}, \text{ for } d_i \in com_k \text{ and } d_j \in com_k \tag{4.10}$$

This equation shows that the rank of a combination is the sum of the rank values of ${}_kC_2$ pairs of initial centroid candidate documents in the combination. In this case, there are 4 combinations available and their rank values are in Table 4.2.

com_k	${}_kC_2$ pairs of document candidates	$rank_{com_k}$
$\{d_1, d_2, d_3\}$	$\{d_1, d_2\}, \{d_1, d_3\}, \{d_2, d_3\}$	6
$\{d_1, d_2, d_4\}$	$\{d_1, d_2\}, \{d_1, d_4\}, \{d_2, d_4\}$	18
$\{d_1, d_3, d_4\}$	$\{d_1, d_3\}, \{d_1, d_4\}, \{d_3, d_4\}$	14
$\{d_2, d_3, d_4\}$	$\{d_2, d_3\}, \{d_2, d_4\}, \{d_3, d_4\}$	14

Table 4.2: Ranks of the candidate sets of initial centroids

Then, we choose the combination with the highest rank as the set of initial centroids for k-means algorithm. In this example, $\{d_1, d_2, d_3\}$ is chosen since the rank value of this combination is 6, which is the highest rank among 4 different combinations. The documents

in this combination are considered to be far from each other and also close enough to a group of documents, so they can serve as the initial centroids of the k-means algorithm.

The effectiveness of this method depends on the selection of n_{plus} and the distribution of the cluster sizes. In Section 4.4.3.1, we will discuss how to select an appropriate n_{plus} to achieve the best clustering result. For those data sets having a large variation in cluster sizes, the initial centroids selected by this proposed rank-based method may not distribute over all the clusters, and some of them could be in a cluster with large size. Our experiments showed that our proposed clustering criterion function described in the following Section 4.3.2 could be adopted to improve the clustering results of those data sets.

4.3.2 Clustering Criterion Based on the Cosine and Link

For document clustering, the *cosine* function is a very popular criterion function. It measures the similarity between two documents as the correlation between the document vectors representing them. This correlation is quantified as the cosine value of the angle between the two vectors. The larger cosine value indicates that these two documents share more terms and are more similar. When the *cosine* function is adopted in the family of k-means algorithms, the correlation between each pair of a document and a centroid is evaluated during the assignment step.

However, the similarity measure based on the *cosine* function may not work well for some document data sets. Usually, the number of unique terms in a document data set is very large while the average number of unique terms in a document is much smaller. In addition, documents that cover the same topic and belong to a single cluster may have a small subset of terms within a much larger vocabulary of the topic. Here we give two examples to explain this situation. The first example is regarding the topic and subtopic relationship. A cluster with the topic of the family tree is related to a set of terms such as *parents*, *brothers* and *sisters*, *aunts* and *uncles*, etc. Some documents in this cluster may focus on brothers and sisters, and the rest covers other branches of the family tree. Thus, those documents do not contain all the relevant terms listed above.

Another example is about the usage of synonyms. Different terms are used in different documents even if they are covering the same topic. Documents in a cluster of automobile industry may not always use the same word to describe the *car*. There are many words available for the same meaning, such as *auto*, *automobile*, *vehicle*, etc. Thus, it is quite possible that a pair of documents in a cluster have few terms in common, but have connections with other documents in the same cluster, and those documents have many common terms with each of these two documents, respectively. In this case, the concept of *link* may help us identify the closeness of two documents by checking their *neighbors*. When a document d_i shares a group of terms with its *neighbors*, and a document d_j shares another group of terms with the same set of *neighbors*, even though these two documents, d_i and d_j, are not considered similar by the *cosine* function, the neighbors they share show how close they are.

Another fact is the sets of unique terms related to the topics of clusters may not have the same size because, in real life, different subjects have their own vocabularies. In a cluster involving a large vocabulary, since documents are spread out over a larger number of terms, most document pairs will share few terms, and consequently only a small percentage of document pairs would have a certain number of terms in common. Thus, when using the *cosine* function, the similarity between a document and a centroid would be much smaller because the centroid is by definition the mean vector of all the document vectors in the cluster. Splitting a cluster related to a large vocabulary will increase the value of the global criterion function. Since the cluster refinement phase of the k-means algorithm is the process of maximizing the global criterion function when the *cosine* function is used for similarity measure, it prefers to split the clusters with large vocabularies. However, this is not desirable since documents in these clusters may be strongly related to each other. On the other hand, if the global criterion function is based on the concept of *link*, which catches the information about the connections of documents in a cluster in terms of the *neighbors*, a cluster will not be split just because it has large vocabulary. If the documents in the cluster are strongly linked (i.e., sharing a large number of *neighbors*), it will not be split regardless of its vocabulary size. In other words, splitting the clusters with large vocabularies will not be favored because

it will reduce the value of the global criterion function.

However, the *link* function may not perform well as the clustering criterion function by itself. In the cluster refinement phase, if a document is assigned to the cluster whose centroid shares the largest number of *neighbors* with this document (i.e., the largest *link* function value), there is a high probability that this document is assigned to a large cluster than to a small cluster. For a fixed similarity threshold θ, the centroid of a large cluster, say c_i, has more *neighbors* than the centroid of a small cluster, say c_j. Thus, for a document d, it is quite probable that $link(d, c_i)$ is larger than $link(d, c_j)$. The worst case scenario is that the global criterion function is optimized when the whole data set is in one cluster while all the other clusters are empty.

Based on these discussions, we propose a new clustering criterion function for the family of k-means algorithms by combining the *cosine* and *link* functions as follows:

$$f(d_i, c_j) = \alpha * \frac{link(d_i, c_j)}{N_{max}} + (1 - \alpha) * cos(d_i, c_j), \text{ with } \alpha \in [0, 1] \tag{4.11}$$

where N_{max} is the largest possible value of $link(d_i, c_j)$, and α is the coefficient set by the user. For the k-means algorithm, since all the documents in the data set are involved in the whole clustering process, the largest possible value of $link(d_i, c_j)$ is the number of documents in the data set (n), which means all the documents in the data set are *neighbors* of both d_i and c_j. For the bisecting k-means algorithm, only the documents in the selected cluster are involved in each bisecting step. Thus, the largest possible value of $link(d_i, c_j)$ is the number of documents in the selected cluster, say $|C_j|$. However, for both k-means and bisecting k-means, the smallest value of $link(d_i, c_j)$ is zero, which means d_i and c_j do not have any *common neighbors*.

We use N_{max} to normalize *link* values so that the value of $link(d_i, c_j)/N_{max}$ always falls in the range of [0, 1]. With $\alpha \in [0, 1]$, the value of $f(d_i, c_j)$ is between 0 and 1 for all the cases. Equation 4.11 shows that we use the sum of weighted values of the *cosine* and *link* functions to evaluate the closeness of two documents, and a larger value of $f(d_i, c_j)$ indicates they are closer. When α is set to 0, the criterion function becomes the *cosine* function;

and it becomes the *link* function when α is 1. In Section 4.4, we will use the experimental results on test data sets to show the optimal choice of α for the best clustering result. Since the *cosine* function and the *link* function evaluate the closeness of two documents in different aspects, our new clustering criterion function is more comprehensive in measuring the closeness between documents. During the clustering process, each document is assigned iteratively to the cluster with the largest criterion function value, so that the global criterion function is maximized.

	d_1	d_2	d_3	d_4	d_5	d_6	c_1	c_2	c_3
d_1	1	1	0	1	0	0	1	0	0
d_2	1	1	0	1	0	0	0	1	0
d_3	0	0	1	1	1	0	0	0	1
d_4	1	1	1	1	0	0	1	1	1
d_5	0	0	1	0	1	0	0	0	1
d_6	0	0	0	0	0	1	0	0	0

Figure 4.2: Expanded neighbor matrix (M') of data set S with $\theta = 0.3$ and k = 3

When we calculate $link(d_i, c_j)$, we add k columns to the *neighbor matrix* M of the data set. The new matrix is a $n \times (n + k)$ matrix, denoted by M', in which an entry $M'[i, n + j]$ is 1 or 0 depending on whether a document d_i and a centroid c_j are *neighbors* or not. The *expanded neighbor matrix* for the example data set S is shown in Figure 4.2. The value of $link(d_i, c_j)$ can be obtained by multiplying the ith row with the $n + j$ column of M':

$$link(d_i, c_j) = \sum_{m=1}^{n} M'[i, m] * M'[m, n + j] \tag{4.12}$$

4.3.3 Cluster Selection Based on the Neighbors of the Centroids

For the bisecting k-means algorithm, in each bisecting step, one existing cluster is selected to be split based on a heuristic function. The basic goal of this function is to find an existing cluster with the poorest quality. A cluster with poor quality means the documents in it are not closely related to each other, and the bonds between them are weak. Therefore, our selection of a cluster to split should base on the measurement of the compactness of

clusters. In [59], the compactness of a cluster is measured by its overall similarity, the size of the cluster, or the combination of both the size and the overall similarity. It has been reported in [59] that the differences between them are small according to the final clustering result. The authors recommended to split the largest remaining cluster. However, in our experiment, we found that this method may not produce the best clustering result because the size of a cluster is not necessarily a good measurement of its compactness.

When we choose between two clusters, one is loosely bound and the other is compact, we should split the first one even if its size is smaller than that of the second one. The concept of *neighbors*, which is based on the similarity of two documents, provides more information about the compactness of a cluster than the size of the cluster. We create a new heuristic function which compares the neighbors of the centroids of remaining clusters, and our experimental results show that the performance of bisecting k-means is improved, compared to the case of splitting the largest cluster.

Since we want to measure the compactness of a cluster, only the local *neighbors* of the centroid are counted. In other words, we just count those documents that are similar to the centroid and existing in that cluster. For a cluster C_j, a $|C_j| \times 1$ matrix M'_{C_j} is created by extracting all the entries $M'[i, n+j]$ for $d_i \in C_j$. Then, the number of local *neighbors* of the centroid c_j is obtained by multipling two matrices as follows:

$$N(c_j)_{local} = \sum_{m=1}^{|C_j|} B[1, m] * M'_{C_j}[m, 1] \tag{4.13}$$

where B is a $1 \times |C_j|$ matrix with all entries are set to 1. For the same cluster size and the same similarity threshold θ, the centroid of a compact cluster should have more neighbors than that of a loose cluster. By the definition of the centroid, when the similarity threshold θ is fixed, the centroid of a large cluster tends to have more *neighbors* than that of a small cluster. Thus, we divide the number of neighbors of the centroid by the size of the cluster to get a normalized value, denoted by $V(c_j)$ for c_j, which is always in the range of [0,1]:

$$V(c_j) = N(c_j)_{local}/|C_j| \tag{4.14}$$

In the cluster selection step, we choose the cluster with the smallest V value to split.

4.4 Experimental Results

In order to show that the applications of *neighbor matrix* in k-means (KM) and bisecting k-means (BKM) algorithms can improve the quality of document clustering, we ran the modified k-means and bisecting k-means algorithms by using (1) the initial centroids selection based on the ranks, (2) the clustering criterion function based on the *cosine* and *link* functions, (3) the selection of a cluster to split based on the *neighbors* of the centroids, respectively as well as in combinations, on real-life document data sets. The clustering results were compared with the original k-means and bisecting k-means algorithms. The time and space complexities of the algorithms using the *neighbor matrix* are also discussed. We implemented all the algorithms in C++ on a SuSE Linux PC with a 500 MHz processor and 384 MB memory.

4.4.1 Data Sets

We used 13 test data sets from three different types of text databases, which have been widely used by the researchers in the information retrieval area. The first group of six data sets, denoted by CISI1, CISI2, CISI3, CISI4, CACM1 and MED1, are extracted from the CISI, CACM and MEDLINE abstracts, respectively, which are included in the Classic database [9].

The second group of four data sets, denoted by EXC1, ORG1, PEO1 and TOP1, are are from the EXCHANGES, ORGS, PEOPLE and TOPICS category sets of the Reuters-21578 Distribution 1.0 [55].

The third group of data sets is prepared by ourselves. We tried to simulate the case of using a search engine to retrieve the desired documents from a database, and we adopted the Lemur Toolkit [37] as the search engine. The English newswire corpus of the HARD track of Text Retrieval Conference (TREC) 2004 [27] is used as the database. This corpus includes about 652,309 documents (in 1575 MB) from 8 different sources, and there are 29 test queries. Among those 29 queries, HARD-306, HARD-309, HARD-314 queries were sent

to the search engine and the top 200 results of these queries were collected and classified as three data sets, denoted by SET1, SET2 and SET3, for our evaluation. The reason why we choose only top 200 documents is that usually users do not read more than 200 documents for a single query.

Each document of the test data has been pre-classified into one unique class. But, this information is hidden during the clustering processes and just used to evaluate the clustering accuracy of each clustering algorithm. Before the experiments, the stop words removal and the stemming were performed as preprocessing steps on the data sets. Table 4.3 summarizes the characteristics of all the data sets used for our experiments.

Data Set	Num. of Doc.	Num. of Classes	Min. Class Size	Max. Class Size	Num. of Unique Term	Avg. Doc. Length	Avg. Pairwise Similarity by cosine
CISI1	163	4	4	102	1,844	66	0.04
CISI2	282	4	31	92	2,371	63	0.04
CISI3	135	4	15	85	1,824	63	0.04
CISI4	148	3	24	78	1,935	67	0.04
CACM1	170	5	26	51	1,260	56	0.04
MED1	287	9	26	39	4,255	77	0.02
EXC1	334	7	28	97	3,258	67	0.03
ORG1	733	9	20	349	6,172	138	0.03
PEO1	694	15	11	143	5,046	102	0.04
TOP1	2279	7	23	750	10,719	113	0.03
SET1	200	4	18	88	9,301	940	0.06
SET2	200	3	44	92	8,998	664	0.05
SET3	200	4	10	81	12,368	1637	0.06

Table 4.3: Summary of data sets

4.4.2 Evaluation Methods of Text Clustering

We used the *F-measure* and *purity* values to evaluate the accuracy of our clustering algorithms. The F-measure is a harmonic combination of the *precision* and *recall* values used in information retrieval [56]. Since our data sets were prepared as described above, each

cluster obtained can be considered as the result of a query, whereas each pre-classified set of documents can be considered as the desired set of documents for that query. Thus, we can calculate the precision $P(i,j)$ and recall $R(i,j)$ of each cluster j for each class i.

If n_i is the number of the members of class i, n_j is the number of the members of cluster j, and n_{ij} is the number of the members of class i in cluster j, then $P(i,j)$ and $R(i,j)$ can be defined as:

$$P(i,j) = \frac{n_{ij}}{n_j} \tag{4.15}$$

$$R(i,j) = \frac{n_{ij}}{n_i} \tag{4.16}$$

The corresponding F-measure $F(i,j)$ is defined as:

$$F(i,j) = \frac{2 * P(i,j) * R(i,j)}{P(i,j) + R(i,j)} \tag{4.17}$$

Then, the F-measure for the whole clustering result is defined as:

$$F = \sum_i \frac{n_i}{n} \max_j(F(i,j)) \tag{4.18}$$

where n is the total number of documents in the collection. In general, the larger the F-measure is, the better the clustering result is [59].

The purity of a cluster represents the fraction of the cluster corresponding to the largest class of documents assigned to that cluster, thus the purity of a cluster j is defined as:

$$Purity(j) = \frac{1}{n_j} \max_i(n_{ij}) \tag{4.19}$$

The overall purity of the clustering is a weighted sum of the cluster purities:

$$Purity = \sum_j \frac{n_j}{n} Purity(j) \tag{4.20}$$

In general, the larger the purity value is, the better the clustering result is [71].

4.4.3 Clustering Results

Figures 4.3–4.6 show the F-measure values of the clustering results of all the algorithms on 13 data sets, and Tables 4.4 and 4.5 show the purity values of the clustering results. In the

original k-means (KM) and bisecting k-means (BKM) algorithms, the random algorithm is adopted for the initial centroids selection and the *cosine* function is used as the clustering criterion function. For BKM, the largest cluster is selected to split at every bisecting step, the iteration number for each bisecting step is set to 5. In the figures, *Rank* denotes that the initial centroids are selected based on the ranks of the documents; *CL* denotes that the clustering criterion function is based on the *cosine* and *link* functions; and *NB* denotes that the selection of a cluster to split is based on the neighbors of the centroids. We ran each algorithm 10 times to obtain the average result. The test results demonstrate that with the applications of *neighbor matrix* on KM and BKM, the quality of clustering is improved significantly.

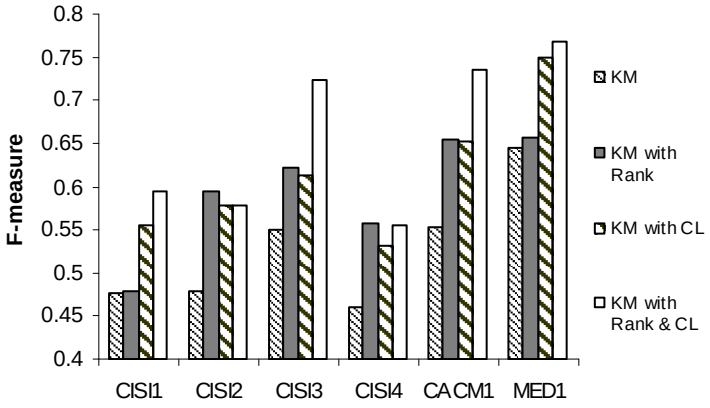


Figure 4.3: Results of k-means algorithms on Classic data sets

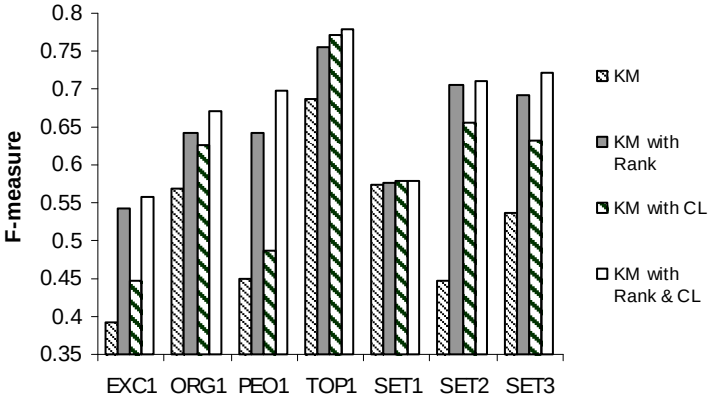


Figure 4.4: Results of k-means algorithms on Reuters and Search Result data sets

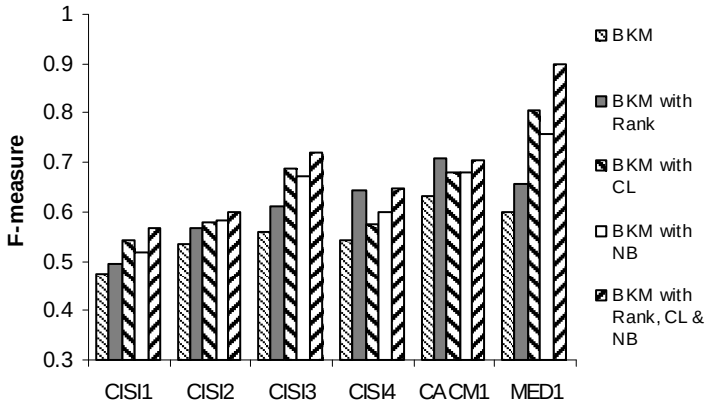


Figure 4.5: Results of bisecting k-means algorithms on Classic data sets

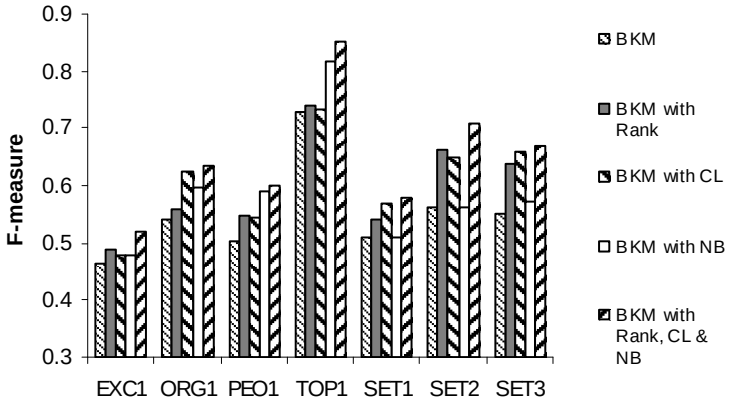


Figure 4.6: Results of bisecting k-means algorithms on Reuters and Search Result data sets

4.4.3.1 Results of Initial Centroids Selection Using the Ranks

From the test results, we can see that the initial centroids selection by by using the ranks of documents performs much better than the random selection in terms of clustering quality.

Since our rank-based method selects k centroids from $k + n_{plus}$ candidates, the setting of n_{plus} is very important. If n_{plus} is too small, our choice of initial centroids is limited to a small set of documents. Even if these documents have the largest numbers of neighbors, which indicates that they are close to a large amount of documents, they may not be distributed evenly across the whole data set. The larger n_{plus} will help us to find a better selection of initial centroids. However, when the candidates set is too big, the computation cost is high. We tried a series of n_{plus} for KM and BKM to find the balance between the clustering

Data Set	KM	KM with Rank	KM with CL	KM with Rank & CL
CISI1	0.534	0.546	0.595	0.625
CISI2	0.504	0.592	0.571	0.606
CISI3	0.760	0.822	0.785	0.778
CISI4	0.561	0.642	0.567	0.561
CACM1	0.593	0.689	0.696	0.800
MED1	0.652	0.693	0.742	0.756
EXC1	0.434	0.587	0.452	0.596
ORG1	0.711	0.744	0.727	0.769
PEO1	0.474	0.634	0.527	0.676
TOP1	0.759	0.803	0.808	0.818
SET1	0.525	0.545	0.545	0.545
SET2	0.495	0.710	0.675	0.720
SET3	0.590	0.700	0.645	0.695

Table 4.4: Purity values of k-means algorithms

quality and the computing cost. The test results show that for KM, within the range $[0, k]$, the larger n_{plus} is, the better the clustering result is. This means, we select k initial centroids from $2k$ candidates. And for BKM, the optimal range of n_{plus} is $[0, 4]$, regardless of k. This is because only 2 initial centroids are needed at each bisecting step of BKM.

Our rank-based method involves several steps, which includes the creation of the *neighbor matrix*, sorting the documents based on the number of neighbors, sorting $k+n_{plus}$ documents based on the *cosine* and *link* values, and calculating the ranks of all candidate sets of initial centroids. The time complexity of creating the *neighbor matrix* is $O(n^2)$ for the data set with n documents, and that of sorting n documents based on their neighbor numbers is $O(nlogn)$. Comparing these two steps, the time complexity of sorting $k + n_{plus}$ documents based on the similarity values and calculating the ranks of all candidate sets of initial centroids is relatively small and can be ignored.

Data Set	BKM	BKM with Rank	BKM with CL	BKM with NB	BKM with Rank, CL & NB
CISI1	0.595	0.607	0.607	0.619	0.619
CISI2	0.539	0.631	0.596	0.543	0.624
CISI3	0.755	0.815	0.763	0.756	0.807
CISI4	0.574	0.642	0.561	0.574	0.655
CACM1	0.689	0.778	0.733	0.689	0.793
MED1	0.711	0.689	0.812	0.777	0.899
EXC1	0.476	0.530	0.503	0.533	0.575
ORG1	0.727	0.754	0.753	0.749	0.754
PEO1	0.506	0.576	0.555	0.586	0.618
TOP1	0.839	0.856	0.830	0.866	0.866
SET1	0.535	0.570	0.545	0.535	0.575
SET2	0.620	0.690	0.630	0.625	0.655
SET3	0.700	0.699	0.699	0.700	0.700

Table 4.5: Purity values of bisecting k-means algorithms

4.4.3.2 Results of the Clustering Criterion Based on the Cosine and Link Functions

The first step of our clustering criterion function based on the *cosine* and *link* is to determine the neighbors of each document. When we check if two documents are *neighbors*, we need to compare their similarity value with the threshold θ (refer to equation 4.6). In order to find the right θ, we first tried the average pairwise similarity of the documents in the data set,

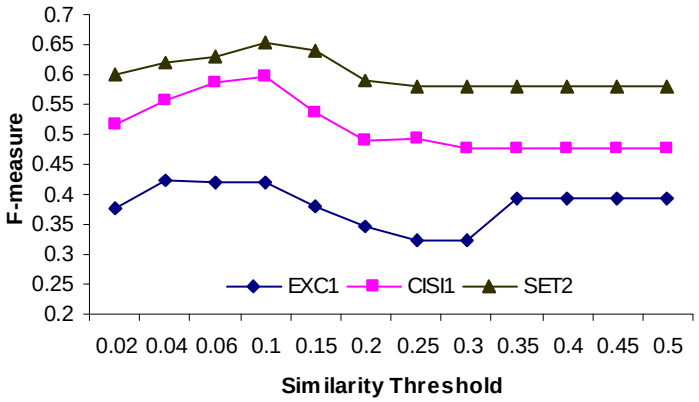


Figure 4.7: Effect of the similarity threshold θ on the F-measure of k-means with CL

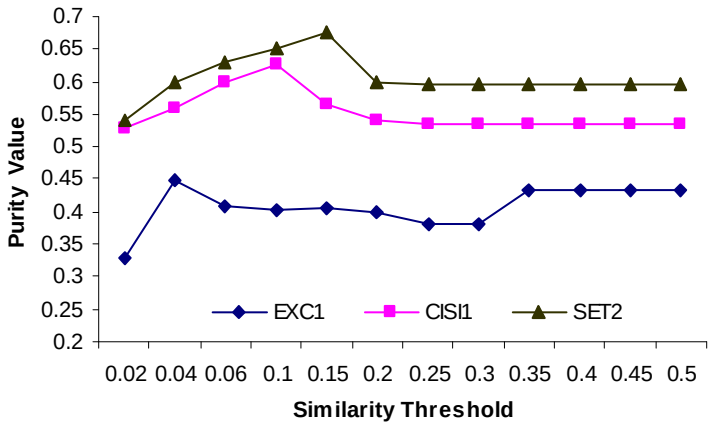


Figure 4.8: Effect of the similarity threshold θ on the purity value of k-means with CL

but some results are good and others are not. Then, we tried the threshold θ between 0.02 and 0.5, and the effect of θ on the F-measure and purity values of k-means with CL on data sets EXC1, CISI1 and SET2 is shown in Figures 4.7 and 4.8. We also performed this test on other data sets, and the results are similar. As we can see, when θ is 0.1, we can achieve the best clustering results. So, θ was set to 0.1 for all other experimental results reported in in this chapter.

In our new clustering criterion function, we use the linear combination of *cosine* and *link* functions to measure the closeness of two documents, as defined in equation 4.11. The range of the coefficient α of the *link* function is [0,1]. We tried different coefficient values for k-means with CL on data sets EXC1, CISI1 and SET2. The results shown in Figures 4.9 and 4.10 suggest that, when the coefficient is set between 0.8 and 0.95, the clustering results are better than the case of using the *cosine* alone. The tests on other data sets showed the same trend. We set the coefficient to 0.9 for other experimental results reported in this chapter. This high optimal coefficient value can be explained by the fact that the *link* value is calculated using the similarity value given by the *cosine*. In the other words, since the *link* value contains the *cosine* value already, the weight of the *cosine* in the linear combination should be smaller than that of the *link*.

The time complexity of our new clustering criterion function is determined by the computation of the *cosine* and *link* functions. For k-means algorithm with the *cosine* function

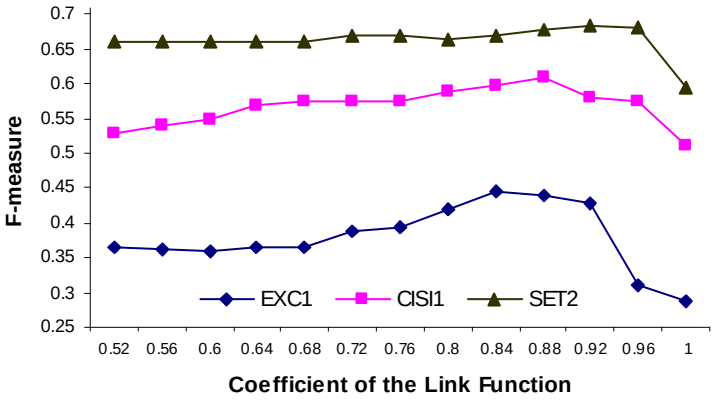


Figure 4.9: Effect of the coefficient α on the F-measure of k-means with CL

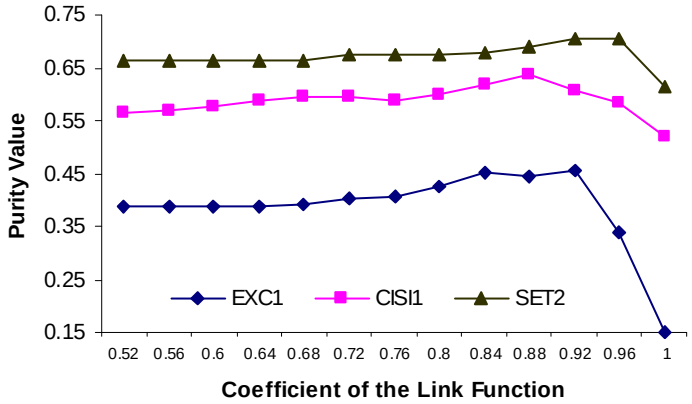


Figure 4.10: Effect of the coefficient α on the purity value of k-means with CL

alone, the time complexity could be represented as:

$$T_{cos} = FnkdLt \tag{4.21}$$

where F is a constant for the calculation of the *cosine* function, n is the number of documents in data set, k is the desired number of clusters, d is the number of unique words in the data set, L is the number of iterations of the loop in k-means, and t is the unit operation time for all basic operations. The computation of the *link* function contains three parts: creating the *neighbor matrix*, expanding the *neighbor matrix* with the columns of k centroids, and calculating the *link* value for each document with every centroid at each loop. The time complexity of creating the *neighbor matrix* could be represented as:

$$T_{matrix} = \frac{n^2 dt}{2} \tag{4.22}$$

The *neighbor matrix* is created just once before the iteration of the loop; and at each iteration, the $n \times n$ *neighbor matrix* is expanded into an $n \times (n + k)$ matrix for k centroids. So, the computation for this expansion involves only the entries in those k columns:

$$T_{expanding} = F'nkdLt \tag{4.23}$$

where F' is a constant for the calculation of judging whether two documents are *neighbors*. The calculation of *link* values for n documents with k centroids is an $n \times k$ matrix multiplication as discussed in Section 4.3.2, and could be represented as:

$$T_{link} = n^2kdLt \tag{4.24}$$

Thus, the time complexity of the new clustering criterion function is:

$$\begin{aligned} T_{CL} &= T_{matrix} + T_{expanding} + T_{link} + T_{cos} && (4.25)\\ &= \frac{n^2dt}{2} + F'nkdLt + n^2kdLt + FnkdLt && (4.26)\\ &= \frac{1 + 2kL}{2}n^2dt + (F' + F)nkdLt && (4.27) \end{aligned}$$

From the above equations, we can see that for a data set containing n documents, the time complexity of our new clustering criterion function is $O(n^2)$, which is computationally acceptable.

By adopting the new clustering criterion function to KM and BKM, both algorithms outperform the original ones on all 13 test data sets. We can conclude that this new criterion function provides a more accurate measurement of the closeness between two documents.

4.4.3.3 Results of the Cluster Selection Based on the Neighbors of the Centroids

From the results shown in Figures 4.5 and 4.6 and Table 4.5, we find the new method of selecting a cluster to split works very well on traditional document data sets—Classic and Reuters. But there is only a slight improvement on the search result data sets in terms of the F-measure, and the purity value of the clustering result is unchanged.

At each bisecting step, the original BKM splits the largest existing cluster. Our new cluster selection method is based on the measurement of the compactness of clusters by

using the neighbors of centroids. Since SET1, SET2, and SET3 data sets are simulated search results, there are more terms shared between documents in these data sets. In the other words, the documents in these data sets are more similar to each other than those in traditional data sets. Table 4.3 shows that the average pairwise similarities of these data sets (SET1, SET2, and SET3) are higher than those of others. Thus, the distribution of clusters in the search result data sets are more balanced than those in traditional data sets. This feature leads to the result that there is no big difference in the selection of a cluster to split between two methods adopted.

The time complexity of bisecting k-means with NB is not much different from that of selecting the largest cluster to split, because the cost of selecting a best cluster out of m, $(m < k)$, clusters to split based on the concept of *link* is very small.

The experimental results prove that our measurement of the compactness of clusters by using the neighbors of centroids is more accurate than just using the cluster size . When running BKM on data sets with unevenly distributed clusters, our cluster selection method is much better.

4.4.3.4 Results of the Combination of the Proposed Methods

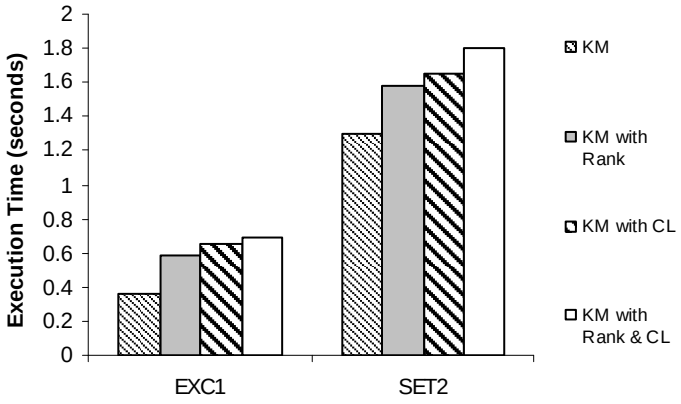


Figure 4.11: Average execution time per loop for k-means on EXC1 and SET2 data sets

We combined the three proposed methods of utilizing the *neighbor matrix* together and ran the modified algorithms on all the test data sets. The results show that the combination

achieves the best results on all the data sets. Since all of our proposed methods are based on the same *neighbor matrix*, adopting all of them into one algorithm is computationally quite reasonable. The average execution times per loop of k-means algorithms on EXC1 and SET2 data sets are shown in Figure 4.11.

For most data sets, we found that the best clustering result obtained are close to the result of using the ranks alone. It proves that the selection of initial centroids is critical for the family of k-means algorithms. For those data sets having a large variation cluster sizes, even if the selection of initial centroids is not good enough, the clustering result is improved by adopting our new clustering criterion function based on the *cosine* and *link* functions. An example of this case is CISI1 data set containing 163 documents, and its maximum class size is 102 documents and the minimum class size is 4 documents. Figure 4.3 shows that for k-means, using the ranks of the documents (to select the initial centroids) performs slightly better than the random selection as the F-measure values are 0.478 and 0.475, respectively. By adopting the new clustering criterion function, the clustering result is improved as expected (F-measure value is 0.556), and the combination of these two methods achieves much better clustering result (F-measure value is 0.5953).

4.5 Conclusions

In this chapter, we proposed three different applications of the *neighbor matrix* in k-means and bisecting k-means clustering algorithms. The *neighbor matrix* of a text database provides the number of neighbors and the link values for each document in the database. Comparing with the local information given by the *cosine* function, the *link* function provides the global view in evaluating the closeness between two documents by using the neighbor documents.

We enhanced k-means and bisecting k-means algorithms by using the ranks of documents for the selection of initial centroids, by using the linear combination of the *cosine* and *link* functions as a new criterion function for clustering, and by selecting a cluster to split based on the neighbors of centroids. All these algorithms are compared with the original k-means and bisecting k-means algorithms by running on real-life text data sets.

Our experimental results showed that the clustering quality of k-means and bisecting k-means algorithms is improved by adopting these new techniques. First, the test results proved that the selection of initial centroids is critical to the clustering quality of k-means and bisecting k-means. The initial centroids selected by our method are well distributed and close to sufficient topically related documents, so they improve the clustering quality. Second, the compactness of a cluster could be measured accurately by evaluating the neighbors of the centroid of the cluster. Third, the test results showed that our new method of measuring the closeness of two documents based on the combination of the pairwise similarity and the connections of neighbor documents performs better than using the pairwise similarity alone.

Chapter 5

Parallel Bisecting K-Means with Prediction Clustering Algorithm

5.1 Introduction

Today, every industry has a huge amount of data waiting for analysis. Data mining algorithms play important roles in this process. Since data sets are measured in gigabytes, sequential data mining algorithms cannot get the job done on a processor with limited memory. Parallel data mining algorithms can solve this problem by utilizing multiple processors with more available memory.

Data clustering [28, 31] is an important process in data mining, and the k-means algorithm [28] is one of the representative partitioning clustering algorithms [51]. The k-means algorithm creates a one-level unnested partitioning of data points by iteratively partitioning the data set. If k is the desired number of clusters, in each iteration, the data set is partitioned into k disjoint clusters. This process is continued until the specified clustering criterion function value is optimized. It works well with a variety of probability distributions [51]. Its simplicity of implementation, ease of interpretation, scalability, speed of convergence and good clustering quality make k-means very popular. However, k-means has some drawbacks, such as its sensitivity to the selection of initial centroids, convergence to suboptimal solutions and uncertainty of the number of iterations [31, 51]. A parallel k-means (PK) algorithm is proposed in [15] based on the Single Program over Multiple Data streams

(SPMD) parallel processing model and message-passing. This PK algorithm can achieve an almost linear speedup.

Bisecting k-means [59] is a variant of k-means. Instead of partitioning the data set into k clusters in each iteration, bisecting k-means algorithm splits one cluster into two subclusters at each bisecting step (by using k-means) until k clusters are obtained. As bisecting k-means is based on k-means, it keeps the merits of k-means, and also has some advantages over k-means [59]. First, bisecting k-means is more efficient when k is large. For the k-means algorithm, the computation of each iteration involves every data point of the data set and k centroids. On the other hand, in each bisecting step of bisecting k-means, only the data points of one cluster and two centroids are involved in the computation. Thus, the computation time is reduced. Second, bisecting k-means tends to produce clusters of similar sizes, while k-means is known to produce clusters of widely different sizes [59]. Moreover, bisecting k-means produces clusters with smaller entropy (i.e., purer clusters) than k-means does, as will be shown in Section 5.4.

In this chapter, we propose a new parallel bisecting k-means algorithm, named PBKP, which adopts a *prediction* step to balance the workloads of multiple processors. We implemented PBKP on a cluster of Linux workstations and analyzed its performance. Our experimental results show that the speedup of PBKP is linear with the number of processors and the number of data points, and it is more scalable than parallel k-means (PK) with respect to the dimension and the desired number of clusters.

The rest of this chapter is organized as follows: In Section 5.2, the sequential k-means (SK) and bisecting k-means (BK) algorithms are reviewed. In Section 5.3, the parallel k-means (PK) algorithm proposed in [15] and our PBKP algorithm are described in details. In Section 5.4, first the clustering qualities of SK, BK and PBKP are compared, then the speedup and scaleup of PBKP are evaluated and compared with those of PK. The applicability of bisecting k-means is also discussed. Section 5.5 contains some conclusions and a future research topic.

5.2 Sequential Algorithms

5.2.1 Sequential k-means (SK) Algorithm

k-means is a popular algorithm to solve the problem of clustering a data set into k clusters. If the data set contains n data points, $X_1, X_2, \ldots, X_n$, of dimension d, then the clustering is the optimization process of grouping n data points into k clusters so that the following global function is either minimized or maximized.

$$\sum_{j=1}^{k} \sum_{i=1}^{n} f(X_i, c_j) \tag{5.1}$$

c_j represents the centroid of cluster C_j, for $j = 1, \ldots, k$, and $f(X_i, c_j)$ is the clustering criterion function for a data point X_i and a centroid c_j. The goal of this function $f(X_i, c_j)$ is to optimize different aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations [71]. For example, the Euclidean distance function minimizes the intra-cluster dissimilarity. In that case, a data point X_i is assigned to the cluster with the closest centroid c_j, and the global function is minimized as a result. When the cosine similarity function is used, a data point X_i is assigned to the cluster with the most similar centroid c_j, and the global function is maximized as a result.

This optimization process is known as a NP-complete problem [23], and the sequential k-means (SK) algorithm was proposed to provide an approximate solution [28]. The steps of SK are as follows:

1. Select k data points as initial cluster centroids.
2. For each data point of the whole data set, compute the clustering criterion function with each centroid. Assign the data point to its best choice. (*calculation step*)
3. Recalculate k centroids based on the data points assigned to them. (*update step*)
4. Repeat steps 2 and 3 until convergence.

The computation complexity of SK is determined by the number of data points (n), the dimension of the data point (d), the desired number of clusters (k), and the number of loops

in SK (L). For a processor, we assume all the basic operations have the same unit operation time (t). At each loop, the computation complexity of the calculation step is dominated by the clustering criterion function, which has $f(n, k, d)$ operations. For the update step, recalculating the centroids needs kd operations. Thus, the time complexity of the whole k-means algorithm is:

$$T_{SK} = (f(n, k, d) + kd)Lt \tag{5.2}$$

If the clustering criterion function is the Euclidean distance function, $f(n, k, d) = 3nkd + nk + nd$ [15]; and for the cosine similarity function, $f(n, k, d) = 2nkd + nk + nd$. Under the assumption that the number of data points (n) is much larger than d and k, Equation (5.2) could be rewritten as:

$$T_{SK} = FnkdLt \tag{5.3}$$

where F is a constant for the clustering criterion function used. SK is very sensitive to the selection of initial centroids, and different initial centroids could produce different clustering results. For the same initial centroids, each run of SK on the same data set always has the same L.

5.2.2 Bisecting k-means (BK) Algorithm

The bisecting k-means (BK) [59] is a variant of the k-means algorithm. The key point of this algorithm is that only one cluster is split into two subclusters at each step. This algorithm starts with the whole data set as a single cluster, and its steps are:

1. Select a cluster C_j to split based on a rule.
2. Find 2 subclusters of C_j by using the k-means algorithm (*bisecting step*):
   (a) Select 2 data points of C_j as initial cluster centroids.
   (b) For each data point of C_j, compute the clustering criterion function with the 2 centroids, and assign the data point to its best choice. (*calculation step*)
   (c) Recalculate 2 centroids based on the data points assigned to them. (*update step*)

(d) Repeat steps 2b and 2c until convergence.

3. Repeat step 2 I times, and select the split that produces the clusters satisfying the global function.

4. Repeat steps 1, 2 and 3 until k clusters are obtained.

I is the number of iterations for each bisecting step, which is usually specified in advance. There are many different rules that we can use to determine which cluster to split, such as selecting the largest cluster at the end of the previous iteration or based on the clustering criterion function. However, it has been reported that the differences between them are small according to the final clustering result [59]. In this chapter, we always split the largest remaining cluster.

The computation complexity of BK is determined by the size of C_j at each bisecting step (n_j), the dimension of the data point (d), the desired number of clusters (k), the number of loops of k-means in each bisecting step (L), and the number of iterations for each bisecting step (I). In the bisecting step, $f(n_j, 2, d)$ operations are required for the calculation step, and $2d$ operations for the centroids updating step. Since each bisecting step produces one more cluster, total $k-1$ bisecting steps are needed to produce k clusters. Thus, the time complexity of BK can be represented as:

$$T_{BK} = (f(\overline{n_j}, 2, d) + 2d)\overline{L}I(k-1)t \tag{5.4}$$

where $\overline{n_j}$ is the average size of C_j of each bisecting step, and $\overline{L}$ is the average number of loops of k-means for each iteration of a bisecting step. Under the assumption that $\overline{n_j}$ is much larger than d and k, Equation (5.4) could be rewritten as:

$$T_{BK} = \frac{2F}{k}\overline{n_j}d\overline{L}I(k-1)t \text{ when } \overline{n_j} \leq n \tag{5.5}$$

The comparison of two Equations (5.3) and (5.5) shows that, when k is large, BK is even more efficient than k-means. The computation of each loop in k-means involves n and k, while the computation of the iteration in each bisecting step of BK involves $\overline{n_j}$ ($\overline{n_j} \leq n$) and $\frac{2(k-1)}{k} \approx 2$ (when k is large).

5.3 Design of Parallel Algorithms

5.3.1 Parallel k-means (PK) Algorithm

A parallel k-means (PK) algorithm is proposed in [15] based on the Single Program over Multiple Data streams (SPMD) parallel processing model and message-passing. PK is easy to implement and achieves a nearly linear speedup. The steps of PK are as follows:

1. Evenly distribute n data points to p processors so that each processor has n_p data points in its disk, where $n_p = n/p$.

2. Select k data points as initial centroids, and broadcast them to the p processors.

3. Each processor calculates the clustering criterion function for each of its n_p data points with each centroid of k clusters, and assigns each data point to its best choice. (*calculation step*)

4. Collect all the information needed from the p processors to update the global cluster centroids, and broadcast it to the p processors. (*update step*)

5. Repeat steps 3 and 4 until convergence.

The strategy of this algorithm is to exploit the data-parallelism of the sequential k-means (SK) algorithm. Step 2 of SK shows that the calculation of the clustering criterion function for different data points could be done at the same time without affecting the final result. Thus, by distributing n data points to p processors, these processors can execute the calculation step on n/p data points at the same time. However, the trade-off is the communication time in step 4 of PK. At the end of each loop, the information of k centroids of dimension d is collected and then broadcast to p processors for the next loop's calculation step. The communication time is determined by k and d as:

$$T_{comm} = MdkL \tag{5.6}$$

where M is the unit time required to collect and broadcast from/to p processors for a floating point number. When the implementation language and the system are determined, M is a constant. It is reported that, for most architectures, M is associated with $O(\log p)$ [12]. The cost of broadcasting the initial centroids is ignored because it is constant for the same k and d, regardless of n, and it is relatively very small compared to the costs of other steps. Based on the time complexity of SK, the time complexity of PK could be represented as:

$$T_{PK} = \frac{Fnkd}{p}Lt + MdkL \tag{5.7}$$

Figure 5.1 shows how multiple processors work together to achieve the speedup. If the step 2 of SK takes t_2 seconds, then the step 3 of PK takes t_2/p seconds. Even though the step 2 of PK takes a little longer than the step 1 of SK because of the communication cost, the difference could be ignored if the time of the calculation step is relatively long. The difference between the step 4 of PK and the step 3 of SK can be explained in the same way.

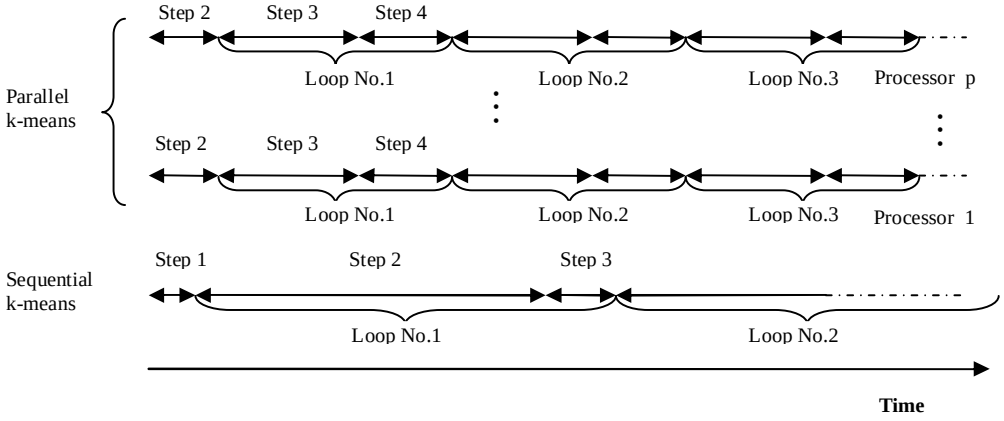


Figure 5.1: Comparison of sequential and parallel k-means algorithms

5.3.2 Parallel Bisecting k-means (PBK) Algorithm

Like the parallel k-means (PK) algorithm, we can design a parallel bisecting k-means (PBK) algorithm based on the SPMD parallel processing model and message-passing. The steps of PBK are as follows:

1. Evenly distribute n data points to p processors.

2. Select a cluster C_j to split based on a rule, and broadcast this information to all processors.

3. Find 2 subclusters of C_j using the k-means algorithm (*bisecting step*):

   (a) Select 2 data points of C_j as initial cluster centroids and broadcast them to the p_j processors that have data members of C_j.

   (b) Each processor calculates the clustering criterion function for its own data points of C_j with 2 centroids, and assigns each data point to its best choice. (*calculation step*)

   (c) Collect all information needed to update 2 centroids and broadcast it to the p_j processors participating in the bisecting. (*update step*)

   (d) Repeat steps 3b and 3c until convergence.

4. Repeat steps 2 and 3 I times, and select the split that produces the clusters satisfying the global function.

5. Repeat steps 2, 3 and 4 until k clusters are obtained.

Let's discuss the computation complexity of PBK. Only the largest remaining cluster C_j is split at each bisecting step. C_j has n_j data points which may not be evenly distributed over the p processors. For each bisecting step, each of the p_j processors can execute the calculation step on n_{jp} data points at the same time, where n_{jp} represents the number of data members of C_j allocated to a processor. Obviously, the number of data points that each processor is working on may be different. And for each processor, the value of n_{jp} for each bisecting step may change, too. For each bisecting step, the largest n_{jp} determines the time of the calculation step because, in order to start the update step, every processor must wait until all the other processors complete the calculation step. The calculation time of each bisecting step is

$$T_{comp} = \frac{2F}{k} \max(n_{jp}) d\overline{L} I t \tag{5.8}$$

At the end of each iteration, the information of 2 centroids of dimension d is collected and broadcast to the p_j processors for the calculation of next iteration. Those processors that have no data point of the selected cluster C_j do not participate in this update step. The communication time of each bisecting step can be represented as:

$$T_{comm} = 2M_j d\overline{L}I \tag{5.9}$$

where $M_j \leq M$ because $p_j \leq p$. During the $k-1$ bisecting steps, M_j is not a constant since p_j is not fixed.

The broadcasting time of initial centroids is ignored because it is relatively small compared to the calculation time of the algorithm. By combining Equations (5.8) and (5.9), the time complexity of PBK could be represented as:

$$\begin{aligned} T_{PBK} &= \frac{2F}{k}(\max(n_{1p}) + \cdots + \max(n_{(k-1)p}))d\overline{L}It + 2(M_1 + \cdots + M_{k-1})d\overline{L}I \quad (5.10) \\ &= \frac{2F}{k}\overline{\max(n_{jp})}d\overline{L}I(k-1)t + 2\overline{M_j}d\overline{L}I(k-1) \quad (5.11) \end{aligned}$$

where $\overline{\max(n_{jp})}$ is the average of $\max(n_{jp})$ of each bisecting step, and $\overline{M_j}$ is the average of M_j.

Figure 5.2 shows the drawback of this implementation. In each calculation step of PK, all data points of the data set are involved. When they are evenly distributed over the p processors, the step 3 of PK takes the same time at each processor. But for PBK, not every data point is involved in the calculation. Only those belonging to the selected cluster C_j are involved in the calculation. For PBK, the range of $\max(n_{jp})$ is $[n_j/p, n_j]$. The best scenario is that every processor has the same number of data points of C_j (i.e., $\max(n_{jp}) = n_j/p$). The worst scenario is that only one processor has all data points of C_j (i.e., $\max(n_{jp}) = n_j$). During the calculation step, some processors may be idle for different length of time, hence they may not be fully utilized.

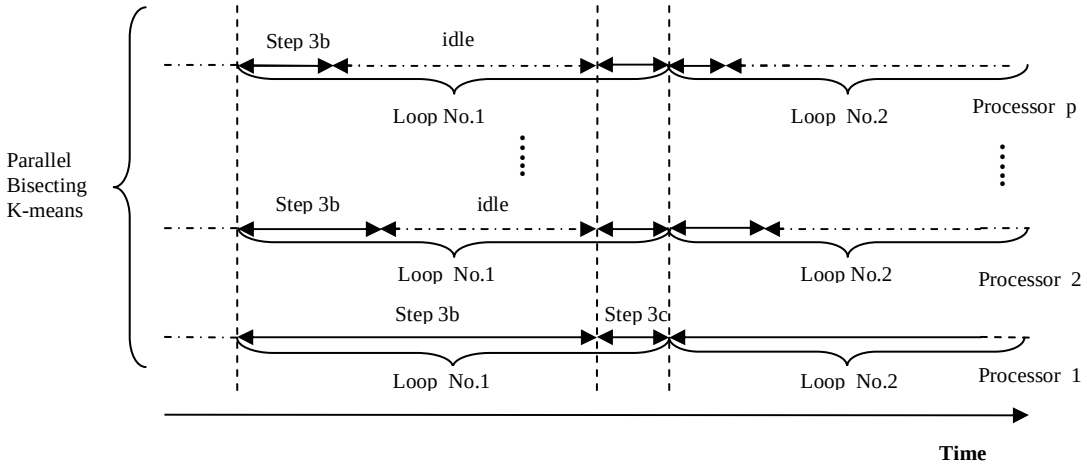


Figure 5.2: Timing diagram of PBK

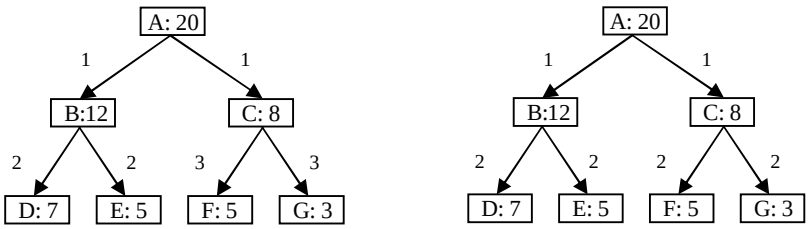
5.3.3 Parallel Bisecting k-means with Prediction (PBKP) Algorithm

In order to improve the processor utilization, we propose a new algorithm, named Parallel Bisecting k-means with Prediction (PBKP). PBKP tries to predict the future bisecting step. At each bisecting step, instead of splitting one largest cluster, it splits two largest clusters. In this way, the processor utilization is improved and the number of bisecting steps is reduced. As a result, the speedup is increased and the total execution time is shortened.

It has been reported that bisecting k-means tends to produce the clusters of similar sizes [59], where the size of a cluster is the number of data points in the cluster. When the size of a selected cluster is S, the sizes of its two subclusters are usually around $S/2$. We can take advantage of this characteristic by modifying the PBK described in Section 5.3.2. First, let's look at the example of BK shown in Figure 5.3. An initial cluster **A** has 20 data points, which is represented as a box with the cluster name and size in it. Each arrow represents a bisecting step, and the associated label shows the order of the step. Figure 5.3(a) shows the case of original BK algorithm. At the first bisecting step, cluster **A** is split into clusters **B** and **C**. Since BK tends to produce clusters of similar sizes, if $\mathbf{C}.size < \mathbf{B}.size < 2\mathbf{C}.size$, then it is quite probable that, after cluster **B** is split at the second bisecting step, cluster **C** will be split at the third bisecting step by assuming two subclusters of **B** would not be

larger than **C**.

Figure 5.3(b) shows the case that we predict cluster **C** is the one to be split at the third bisecting step and do it one step ahead. So, clusters **B** and **C** are split into two subclusters, respectively, at the second bisecting step. Here, $\mathbf{C}.size < \mathbf{B}.size < 2\mathbf{C}.size$ is a prerequisite of our prediction step. In other words, the prediction step could be performed only under this condition. This example shows that splitting the largest cluster and the second largest cluster at one bisecting step does not change the final clustering result, whereas one bisecting step is reduced.



(a) BK

(b) Modified BK with a correct prediction

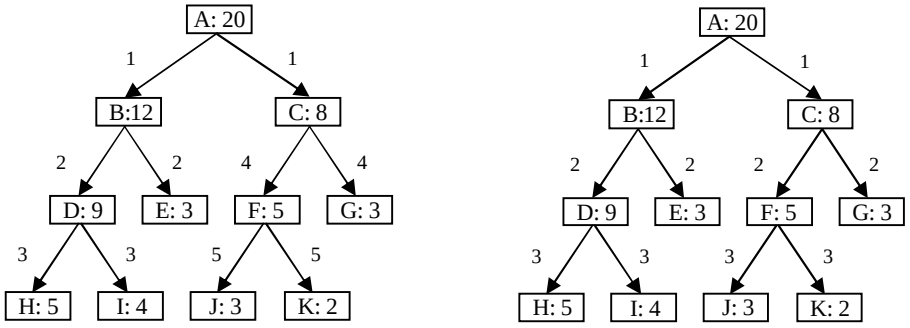
Figure 5.3: Bisecting k-means and its modification (correct prediction case)

However, there is no guarantee that our prediction is always correct. The example in Figure 5.4 shows this situation. Figure 5.4(a) shows that cluster **C** is split at the fourth bisecting step because one subcluster of cluster **B** is larger than **C**. In this case, our prediction that cluster **C** would be split at the third step is wrong. Even so, Figure 5.4(b) shows that splitting **C** at the second step with **B** does not change the final clustering result, either, while the total number of bisecting steps is reduced from 5 to 3.

There may be a case that some descendant of cluster **B** is always larger than cluster **C**, so that **C** will not be selected to split, but the chance is very slim. Our experimental results show that the quality of the final clustering of PBKP is as good as that of BK.

The key point of PBKP is to split two clusters, instead of one cluster, at each bisecting step. As more clusters are split at each bisecting step, more data points are involved in

the calculation step, and the distribution of those data points over the processors usually becomes more uniform. Thus, the processor idle time is reduced. Moreover, the reduced number of bisecting steps helps to reduce the communication cost of the parallel algorithm.



(a) BK

(b) Modified BK with a wrong prediction

Figure 5.4: Bisecting k-means and its modification (wrong prediction case)

The steps of PBKP are as follows:

1. Evenly distribute n data points to p processors.

2. Select the largest cluster C_j and the second largest cluster C'_j from the remaining clusters to split if $C_j.size < 2C'_j.size$ (*prediction step*). Otherwise, select only the largest cluster to split. Broadcast this information to all processors.

3. Find 2 subclusters of C_j and C'_j, respectively, by using the k-means algorithm (*bisecting step*):

   (a) Select 4 data points as initial cluster centroids and broadcast them to the p_j processors that have data members of C_j and C'_j.

   (b) Each processor calculates the clustering criterion function for its own data points of C_j and C'_j with 2 sets of 2 centroids, respectively, and assigns each data point to its best choice. (*calculation step*)

(c) Collect all information needed to update 4 centroids and broadcast it to the p_j processors. (*update step*)

(d) Repeat steps 3b and 3c until convergence.

4. Repeat steps 2 and 3 I times, and select the split that produces the clusters satisfying the global function.

5. Repeat steps 2, 3 and 4 until k clusters are obtained.

Since the subcluster sizes of each bisecting step are very similar in most cases, usually $C_j.size < 2C'_j.size$ holds, hence both C_j and C'_j are bisected.

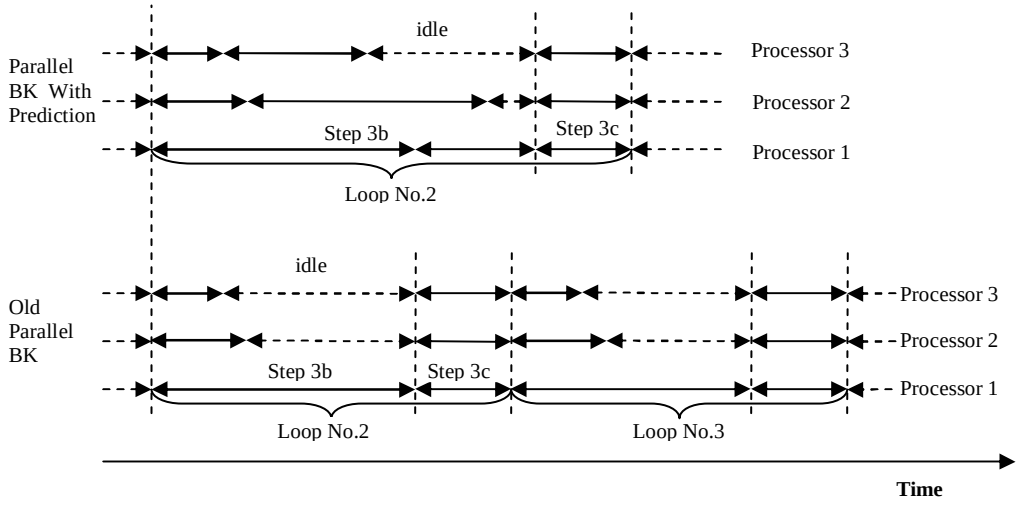


Figure 5.5: Timing diagrams of PBK and PBKP

Figure 5.5 compares the timing diagrams of PBK and PBKP. In order to simplify the comparison, we assumed I is 1. The prediction step reduces the total execution time in two ways. First, it reduces the processor idle time. Since two clusters' data points are usually more uniformly distributed over the processors than one cluster's, bisecting two clusters, one by one in two steps, may take longer time than splitting two in one step. Second, it reduces the total number of bisecting steps from $k-1$ to $k/2$.

The calculation time of each bisecting step is:

$$T_{comp} = \frac{2F}{k} \max(n_{jp} + n'_{jp}) d\overline{L} It \tag{5.12}$$

where n_{jp} and n'_{jp} represent the numbers of data members of C_j and C'_j, respectively, which are allocated to a processor. The range of $\max(n_{jp} + n'_{jp})$ is $[(n_j + n'j)/p, (n_j + n'_j)]$.

At the end of each iteration, the information of 4 centroids of dimension d is collected and broadcast to the p_j processors for the next loop's calculation step. The communication time of each iteration is:

$$T_{comm} = 4M_j d\overline{L}I \tag{5.13}$$

Thus, the time complexity of PBKP could be represented as:

$$T_{PBKP} = F\overline{\max(n_{jp} + n'_{jp})}d\overline{L}It + 2\overline{M_j}d\overline{L}Ik \tag{5.14}$$

5.4 Experimental Results

First, we evaluated the clustering qualities of k-means (SK) and bisecting k-means (BK) on data sets with different cluster size distributions to find their best applicability. Then, the clustering accuracies of new parallel bisecting k-means with prediction (PBKP) and BK are compared to show that the prediction step does not affect the quality of the clustering. Last, the speedup and scaleup of PBKP algorithm were measured to show its efficiency.

5.4.1 Experimental Setup

Our test platform is a 9-node Linux cluster system, where nodes are connected by a Fast Ethernet switch. Each node has a 800 MHz Pentium processor, 512 MB memory and a 40 GB disk drive. The implementation of all the algorithms is in Java, and the Remote Method Invocation (RMI) is used for interprocess communication. Since we focused on the efficiency of the parallel algorithms, our time measurement ignored the disk I/O times for the initial reading and distribution of the data set and for the final result saving on the disk.

We used both real-world data sets and artificial data sets in our experiments. For the quality test of sequential and parallel algorithms, four real-world data sets and three groups of artificial data sets were used in the experiments. For the performance and scalability analyses of parallel algorithms, three groups of artificial data sets were used.

The four real-world data sets used in our experiments are: king-rook-vs-king chess database (KRKOPT), letter recognition database (LETTER), optical recognition of handwritten digits (OPTDIGITS), and pen-based recognition of handwritten digits (PENDIGITS). They are obtained from the UCI Machine Learning Repository [61]. These data sets are different in terms of the dimension, the number of data points, the number of clusters, and the cluster size distribution. Each data point of the data sets has been pre-classified into a class. This information is hidden during the clustering process and used to evaluate the clustering accuracy of each clustering algorithm. Table 5.1 summarizes the characteristics of the real-world data sets.

Data Set	Number of Data Points	Number of Classes	Number of Dimensions	Min. Class Size	Max. Class Size
KRKOPT	28056	18	6	27	4553
LETTER	20000	26	16	734	813
OPTDIGITS	3823	10	64	363	414
PENDIGITS	7494	10	16	696	802

Table 5.1: Summary of real-world data sets used

The artificial data sets used in the experiments were generated by using the algorithm introduced in [45]. The clusters generated by this algorithm are well-separated, mildly truncated multivariate normal mixtures with boundaries separated by a random quantity. As such, the resulting structure could be considered to consist of natural clusters which exhibit the properties of external isolation and internal cohesion [45]. This kind of artificial data sets were widely used to test the properties of clustering algorithms [15]. The generated data sets contain distinct nonoverlapping clusters. For the given number of clusters (k), the number of data points (n) and the dimension (d), three data sets with different cluster size distributions were generated to test the efficiency of parallel clustering algorithms. Since we set the noise dimension and the percentage of outliers as zero, the generated data sets could be considered as classified data sets to evaluate the quality of clustering algorithms in terms of accuracy.

5.4.2 Quality Test of Sequential and Parallel Algorithms

The selection of initial centroids plays a very important role in k-means and bisecting k-means algorithms. Different initial centroids produce different clustering results, thus their qualities are different. Especially for k-means, the number of loops in each run depends on the choice of initial centroids. In our experiments, in order to eliminate this impact, we used the same initial centroids for both k-means and bisecting k-means.

5.4.2.1 Evaluation Method

We use the Entropy [58] to evaluate the quality of the clustering algorithm in terms of accuracy. The entropy is an external quality measure of a clustering algorithm. By comparing the clustering result with known classes, this measure shows how good it is. For each cluster of the clustering result, the class distribution of the data points is calculated first by computing the probability that a member of cluster j belongs to class i, denoted by p_{ij}. Then, the entropy of each cluster j is calculated as:

$$E_j = -\sum_i p_{ij} \log p_{ij} \tag{5.15}$$

The total entropy of all the clusters is the sum of the entropies of the clusters weighted by their sizes:

$$E = \sum_{j=1}^{k} \frac{n_j E_j}{n} \tag{5.16}$$

where n_j is the size of cluster j, n is the total number of data points, and k is the number of clusters. The smaller the entropy, the purer the produced clusters are.

5.4.2.2 Comparison of Sequential k-means (SK) and Bisecting k-means (BK) Algorithms

Three groups of artificial data sets were used to test the clustering quality of SK and BK. The first group is the data sets with the same number of data points in each cluster. The second group is the data sets that one cluster always contains 10% of the data points; whereas in the third group, one cluster always contains 60% of the data points. The remaining data

points are distributed as uniformly as possible across the other clusters. For the third group, there is big discrepancy in cluster sizes when the number of clusters is large. On the other hand, the second group has discrepancy in cluster sizes when the number of clusters is small. We ran both SK and BK on these three groups of data sets to compare the results.

For each group, we generated 4 data sets by setting $d = 8$, $n = 10000$, and varying the number of clusters (k) from 4 to 32. Thus, for each k, we generated three different cluster size distributions. Table 5.2 shows the test results. From the results, we can find that, for the same choice of initial centroids, BK works very well on the data sets of group 1, each of which has the clusters with the same number of data points. BK also performs better on the data sets of group 2 when k is small, such as 4, 8, and 16. On the rest of group 2 and all data sets of group 3, there is no big difference between the results of these two algorithms. This is because BK tends to produce the clusters of similar sizes, which justifies the prediction step of our PBKP algorithm.

There are a few methods to find good initial centroids to improve the performance of k-means [4, 7, 13]. If these methods are adopted, k-means can perform better on the data sets with different cluster sizes. However, since it is not the main issue of this chapter, we do not discuss it further.

Group No.	k	k-means	Bisecting k-means
Group 1	4	0.250	0.001
	8	0.083	0.005
	16	0.151	0.004
	32	0.110	0.015
Group 2	4	0.517	0.355
	8	0.285	0.250
	16	0.380	0.294
	32	0.284	0.285
Group 3	4	0.434	0.466
	8	0.441	0.452
	16	0.446	0.447
	32	0.436	0.437

Table 5.2: Entropies of k-means and Bisecting k-means ($d = 8, n = 10000$)

5.4.2.3 Comparison of Parallel Bisecting k-means with Prediction (PBKP) and Bisecting k-means (BK) Algorithms

In order to show that the prediction step of PBKP does not affect the quality of the clustering, we compared the clustering accuracies of PBKP and BK.

Both the real-world data sets listed in Table 5.1 and three groups of artificial data sets were used in this experiment. The three groups of artificial data sets were generated as before; and for each group, 18 data sets were obtained by varying d, n, and k parameter values.

We executed both algorithms 10 times on each data set, and the average entropies of their final clustering results obtained are shown in Tables 5.3 and 5.4.

Group 1					Group 2					Group 3				
n	d	k	BK	PBKP	n	d	k	BK	PBKP	n	d	k	BK	PBKP
10000	4	8	0.052	0.051	10000	4	8	0.209	0.210	10000	4	8	0.452	0.452
		16	0.094	0.094			16	0.331	0.330			16	0.464	0.464
		32	0.027	0.027			32	0.224	0.227			32	0.436	0.436
	8	8	0.005	0.005		8	8	0.250	0.253		8	8	0.452	0.451
		16	0.004	0.004			16	0.294	0.295			16	0.447	0.445
		32	0.015	0.015			32	0.285	0.285			32	0.437	0.439
20000	4	8	0.018	0.018	20000	4	8	0.231	0.230	20000	4	8	0.460	0.462
		16	0.134	0.135			16	0.289	0.291			16	0.448	0.449
		32	0.058	0.058			32	0.271	0.272			32	0.439	0.440
	8	8	0.001	0.001		8	8	0.209	0.209		8	8	0.452	0.453
		16	0.062	0.060			16	0.214	0.213			16	0.448	0.445
		32	0.017	0.017			32	0.263	0.264			32	0.440	0.443
40000	4	8	0.127	0.126	40000	4	8	0.209	0.210	40000	4	8	0.484	0.480
		16	0.009	0.009			16	0.264	0.265			16	0.449	0.452
		32	0.032	0.032			32	0.216	0.216			32	0.444	0.445
	8	8	0.001	0.001		8	8	0.209	0.209		8	8	0.485	0.485
		16	0.053	0.052			16	0.328	0.327			16	0.462	0.463
		32	0.102	0.101			32	0.210	0.210			32	0.442	0.441

Table 5.3: Entropies of BK and PBKP on artificial data sets

As we discussed in Section 5.3.3, the prediction of PBKP has two cases: correct prediction and wrong prediction. However, since PBKP tends to produce clusters with similar sizes as BK does, there is very little chance that the prediction step changes the clustering result. Tables 5.3 and 5.4 show that PBKP has the same level of clustering accuracy as BK, and we found they have the identical clustering result for most of the data sets used.

Data Set	k	BK	PBKP
KRKOPT	18	0.746	0.746
LETTER	26	0.695	0.696
OPTDIGITS	10	0.424	0.423
PENDIGITS	10	0.405	0.405

Table 5.4: Entropies of BK and PBKP on real-world data sets

5.4.3 Performance and Scalability Analysis of Parallel Algorithms

5.4.3.1 Speedup

If it takes t_s seconds to cluster a data set into k clusters on one processor and t_p seconds on p processors, the speedup is the ratio of t_s to t_p. By using Equations (5.3) and (5.7), the speedup of the parallel k-means (PK), denoted by S_{PK}, could be represented as:

$$S_{PK} = \frac{Fnt}{\frac{Fnt}{p} + M} \tag{5.17}$$

From Equation (5.17), we can find that

$$S_{PK} \to p \text{ when } n \gg \frac{Mp}{Ft} \tag{5.18}$$

By using Equations (5.5) and (5.11), the speedup of the parallel bisecting k-means (PBK), denoted by S_{PBK}, could be represented as:

$$S_{PBK} = \frac{F\overline{n_j}t}{F\overline{\max(n_{jp})}t + \overline{M_j}k} \tag{5.19}$$

where

$$\overline{n_j} \leq n,\ \overline{M_j} \leq M \text{ and } \overline{\max(n_{jp})} \in [\frac{\overline{n_j}}{p}, \overline{n_j}]$$

Equation (5.19) shows that

$$S_{PBK} \to \frac{\overline{n_j}}{\overline{\max(n_{jp})}} \text{ when } \overline{\max(n_{jp})} \gg \frac{\overline{M_j}k}{Ft} \tag{5.20}$$

And by using Equations (5.5) and (5.14), the speedup of the parallel bisecting k-means with prediction (PBKP), denoted by S_{PBKP}, could be represented as:

$$S_{PBKP} = \frac{F\overline{n_j}t\frac{k-1}{k}}{F\frac{\overline{\max(n_{jp}+n'_{jp})}}{2}t + \overline{M_j}k} \tag{5.21}$$

where

$$\overline{n_j} \leq n \text{ and } \overline{\max(n_{jp} + n'_{jp})} \in [\frac{\overline{n_j + n'_j}}{p}, \overline{n_j + n'_j}]$$

Equation (5.21) shows that

$$S_{PBKP} \rightarrow \frac{2\overline{n_j}}{\overline{\max(n_{jp} + n'_{jp})}} \text{ when } \frac{\overline{\max(n_{jp} + n'_{jp})}}{2} \gg \frac{\overline{M_j}k}{Ft} \tag{5.22}$$

We evaluated the speedup of PBKP as we changed the number of data points (n), the number of clusters (k) and the dimension (d), respectively. Since the parallel bisecting k-means algorithm is sensitive to the distribution of the data points of each cluster among the processors, for each data set used in the experiment, we simulated both the worst-case and the best-case distribution scenarios. For the best-case scenario, we evenly distributed the data points of each cluster to the processors. For the worst-case scenario, we allocated all the data points of each cluster to one processor. Each algorithm was tested on both cases five times, and the reported execution time is the average of those ten runs.

Figure 5.6 shows the speedup with different numbers of data points (n). We generated three data sets with fixed values of $d = 8$ and $k = 8$, and n was varied from 2^{16} to 2^{20}. As n increases, the speedup of PBKP increases almost linearly. This is because larger n leads to larger $\overline{\max(n_{jp} + n'_{jp})}$ and Equation (5.22) holds in that case. When $n = 2^{20}$, the speedup of PBKP is 6.17 when 8 processors are used.

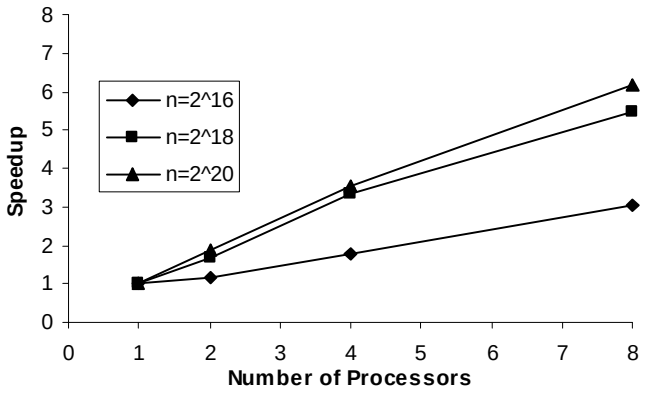


Figure 5.6: Speedup of PBKP with different n ($d = 8$, $k = 8$)

Figures 5.7 and 5.8 show the comparison of the speedups of PK, PBK and PBKP on two data sets. The speedup of PBKP is slightly smaller than that of PK, but larger than

that of PBK. The basic reason why both PBK and PBKP have smaller speedup than PK is that, for PBK and PBKP, the number of processors and their data points involved in each calculation step may be different. The difference in speedup comes from two sources. First, the ideal speedups of PBK and PBKP are smaller that that of PK:

$$\frac{\overline{n_j}}{\max(n_{jp})} \leq p \text{ and } \frac{2\overline{n_j}}{\max(n_{jp} + n'_{jp})} \leq p$$

Second, when $p \leq k$, Equation (5.18) is easier to hold than both Equation (5.20) and Equation (5.22). PBKP can achieve a better speedup than PBK because $\frac{\overline{n_j}}{\max(n_{jp})}$ is smaller than $\frac{2\overline{n_j}}{\max(n_{jp}+n'_{jp})}$ for the same distribution of data points over the processors when both Equation (5.20) and Equation (5.22) hold.

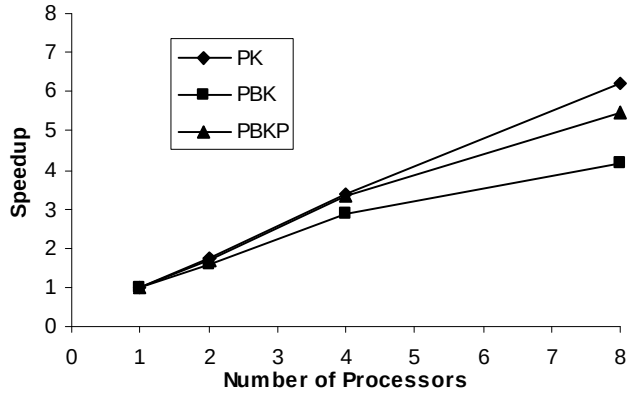


Figure 5.7: Comparison of three algorithms ($d = 8$, $k = 8$, $n = 2^{18}$)

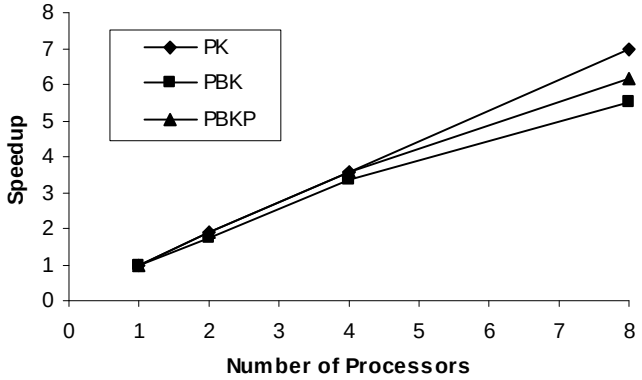


Figure 5.8: Comparison of three algorithms ($d = 8$, $k = 8$, $n = 2^{20}$)

Figure 5.9 shows the speedup of PBKP for different dimensions (d) when $k = 8$ and

$n = 2^{20}$. We can see that changing d does not affect the speedup of PBKP because d is not a factor in Equation (5.22).

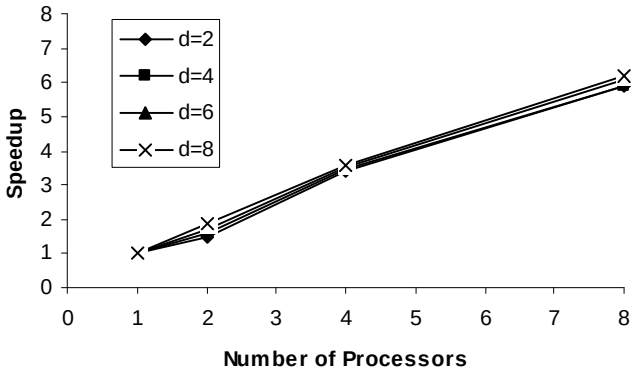


Figure 5.9: Speedup of PBKP with different d ($k = 8$, $n = 2^{20}$)

Figure 5.10 shows the speedup of PBKP for different numbers of clusters (k) when $d = 8$ and $n = 2^{20}$. Like d, changing k does not affect the speedup much because k is not a factor in Equation (5.22).

In summary, the speedup of PBKP increases as n increases. When the number of data points is fixed, the speedup does not change much as k and d increase.

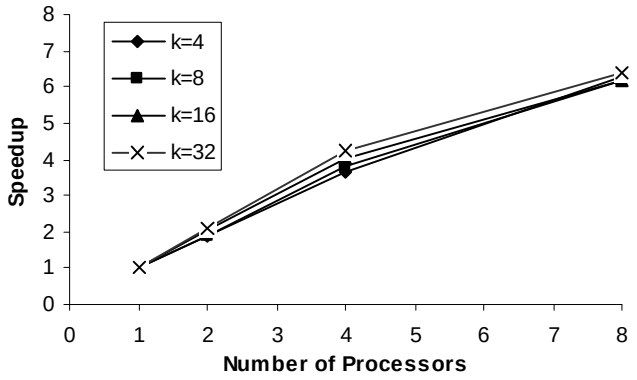


Figure 5.10: Speedup of PBKP with different k ($d = 8$, $n = 2^{20}$)

5.4.3.2 Scaleup

The scaleup is another performance measurement of parallel algorithms. If clustering a data set of size S on one processor takes t_1 seconds, and clustering a data set of size $S \times p$ on p processors takes t_p seconds, then the ratio of t_1 to t_p is the scaleup of the parallel algorithm.

Ideally, a parallel algorithm keeps the ratio not greater than 1. That is, the increase in the number of processors could balance the increase in the size of the problem. Since the size of the problem is determined by n, d and k, we evaluated the scaleup of PBKP with respect to them, one by one. The average execution times per bisecting step of PBKP are shown in Figure 5.11. In that test, the number of iterations for each bisecting step (I) was set to 5. Ideally, the execution time per bisecting step does not increase as the parameter values increase.

To test the scaleup with respect to n, we ran PBKP on the data sets with n as 2^{18}, 2^{19}, 2^{20}, and 2^{21} on 1, 2, 4, and 8 processors, respectively, while d and k were set to 8. Figure 5.11 shows that the execution time per bisecting step is almost constant when the number of processors increases with the data set size. That means, the increase in the data set size due to the increased number of data points is perfectly balanced by the increased number of processors. Thus, when we cluster a data set containing a large number of data points by using the PBKP algorithm, adding processors can reduce the total execution time.

To test the scaleup with respect to d, we ran PBKP on the data sets with d as 2, 4, 8, and 16 on 1, 2, 4, and 8 processors, respectively, while $n = 2^{20}$ and $k = 8$. The result shows that the execution time per bisecting step decreases as the number of processors increases. This is because usually d is not a dominant factor in the time complexity of the clustering criterion function. The increased number of processors can easily offset the effect of increased d on the execution time of each bisecting step.

To test the scaleup with respect to k, we ran PBKP on the data sets with k as 4, 8, 16 and 32 on 1, 2, 4, and 8 processors, respectively, while $n = 2^{20}$ and $d = 8$. The execution time per bisecting step drops quickly as k increases. This is because $\max(n_{jp} + n'_{jp})$ changes at each bisecting step. When k becomes larger, the size of remaining clusters becomes smaller for fixed n, and so does $\max(n_{jp}+n'_{jp})$. The total execution time of PBKP is mainly determined by $\max(n_{jp} + n'_{jp})$ of each bisecting step, thus the average execution time per bisecting step becomes smaller when k increases.

For a comparison, we also evaluated the scaleup of the parallel k-means (PK) algorithm. In Figure 5.12, the average execution time per loop of PK is shown for different cases. By

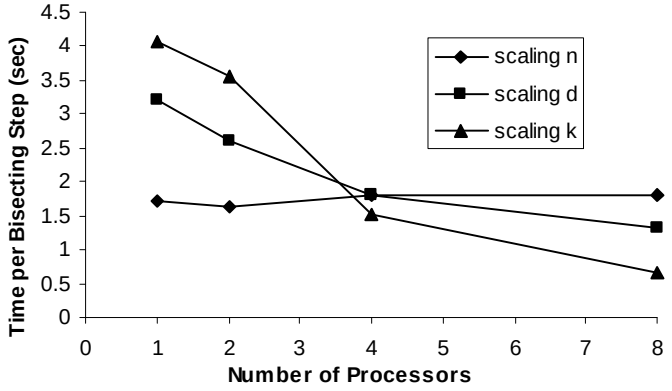


Figure 5.11: Scaleup of PBKP

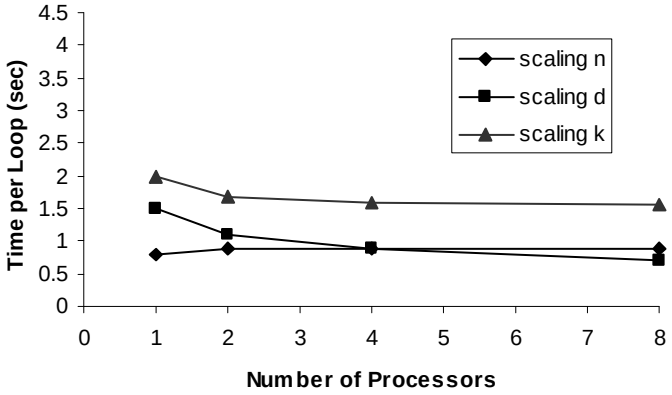


Figure 5.12: Scaleup of PK

using Equation (5.3) and Equation (5.7), we can represent the scaleup with respect to n as:

$$Scaleup_{PK} = \frac{FnkdLt}{\frac{Fnpkd}{p}Lt + MdkL} \tag{5.23}$$

From Equation (5.23), we can find that

$$Scaleup_{PK} \rightarrow 1 \text{ when } n \gg \frac{M}{Ft} \tag{5.24}$$

In Figure 5.12, we can see the scaleup of PK with respect to n approaches to 1 when n is large enough to satisfy the condition of Equation (5.24). Similarly, the scaleup with respect to k is close to 1, but slightly higher. The scaleup with respect to d is also low because, like the case of PBKP, d is not a dominant factor in the time complexity of the clustering criterion function.

Comparing PBKP and PK, PK has better scaleup with respect to the number of data points (n), but PBKP has better scaleup with respect to the dimension (d) and the desired number of clusters (k).

5.5 Conclusions and Future work

In this chapter, we proposed a new parallel bisecting k-means algorithm, named PBKP, for message-passing multiprocessor systems. PBKP predicts the future bisecting step, such that two largest clusters are split at each bisecting step, instead of just one. As a result, processors are better utilized and the number of bisecting steps is reduced. This prediction step is based on the fact that bisecting k-means (BK) tends to produce the clusters of similar sizes and, in that case, the clustering accuracy is usually high. We showed that bisecting k-means produces clusters with smaller entropy (i.e., purer clusters) than k-means does.

We implemented PBKP on a cluster of Linux workstations and analyzed its performance. Our experimental results show that the speedup of PBKP is linear with the number of processors and the number of data points, while its clustering accuracy is as good as that of BK. Compared to the parallel k-means (PK) algorithm [15], PBKP has better scaleup with respect to the dimension (d) and the number of clusters (k). As k increases, PBKP reduces the average execution time per bisecting step, while PK keeps its average loop execution time almost constant.

We also tested PK and PBKP on a collection of text documents which are processed in the vector space model [57]. The speedups of these parallel algorithms were not linear because they were designed based on the assumption that the number of data points (n) is much larger than the dimension of the data points (d) and the desired number of clusters (k). However, it is not the case of a typical text database because its dimensionality in the vector space model increases when the number of text documents increases. This is because the number of unique words (d) in documents usually increases with the number of documents (n). With the rapid growth of WWW, there are many huge text databases to be processed, so there is a clear need of scalable text clustering algorithms. We are developing an efficient text clustering algorithm by considering the unique characteristics of text databases.

Chapter 6

Conclusion

In this dissertation, a set of new scalable and high performance text clustering algorithms is presented. Comprehensive performance studies were conducted on all the proposed algorithms. The experimental results show that our clustering algorithms are scalable and have much better clustering accuracy than existing algorithms. This research is summarized as follows:

- The proposed frequent word sequence mining algorithm is implemented and tested. The testing results show that this algorithm is quite scalable as the database size increases. The new algorithm CFWS, which is proposed to cluster documents by measuring the closeness of documents based on the sharing of frequent word sequences, is implemented. The new algorithm CFWMS, which is proposed to extend CFWS by adopting Ontology into clustering, is completed. The CFWS and CFWMS algorithms are compared with bisecting k-means [59] and FIHC [21] algorithms with respect to the accuracy of text clustering. For experiments, we used the Reuters-21578 text collection and a corpus of the Text Retrieval Conference (TREC) [27]. Our experimental results show that the frequent word sequences mined from our test database and the clusters obtained are useful in improving the precision of text retrieval.

- A new statistic data $R_{w,c}$ is proposed to evaluate the term-category dependency more precisely than χ^2 statistic. This data measures whether the dependency between a term and a category is positive or negative. A new supervised feature selection method based on the family of χ^2 statistical data (CHIR) is proposed and implemented. Unlike

the feature selection method CHI, CHIR selects features which have strong positive dependence on categories. In other words, CHIR keep only the features which are relevant to categories. Furthermore, a new text clustering algorithm TCFS is proposed to involve CHIR into the text clustering process. The cluster label information obtained during the clustering process is utilized as the known class label information for the feature selection. The selected features improve the quality of clustering iteratively, and as the clustering process converges, the clustering result has higher accuracy. TCFS with CHIR has been implemented and compared with existing clustering and feature selection algorithms, such as k-means, k-means with the Term Strength (TS) feature selection method, and TCFS with CHI. Our experimental results show that TCFS with CHIR has better performance than other algorithms in terms of the clustering accuracy for different test data sets.

- The information of *neighbor matrix* is applied to the family of k-means algorithm in three aspects. First, a new document closeness measurement function (CL) is proposed. This new measurement combines global information with pair-wise similarity measurement. Second, a new initial centroids selection method (Rank) is studied. This new method selects the initial centroids based on the rank of documents. Third, a new heuristic function for selecting clusters to spilt (NB) for bisecting k-means is proposed. Our experimental results showed that the clustering quality of k-means and bisecting k-means algorithms is improved by adopting these new techniques. First, the test results proved that the selection of initial centroids is critical to the clustering quality of k-means and bisecting k-means. The initial centroids selected by our method, which are well distributed and close to sufficient topically related documents, improve the clustering quality. Second, the compactness of a cluster could be measured accurately by evaluating the neighbors of the centroid of the cluster. Third, the test results showed that our new method of measuring the closeness of two documents based on the combination of the pairwise similarity and the connections of neighbor documents performs better than using the pairwise similarity alone.

- A new parallel bisecting k-means algorithm, PBKP, is proposed for message-passing multiprocessor systems. PBKP predicts the future bisecting step, such that two largest

clusters are split at each bisecting step, instead of just one. As a result, processors are better utilized and the number of bisecting steps is reduced. This prediction step is based on the fact that bisecting k-means (BK) tends to produce the clusters of similar sizes and, in that case, the clustering accuracy is usually high. PBKP is implemented on a cluster of Linux workstations and its performance is analyzed. The experimental results show that the speedup of PBKP is linear with the number of processors and data points, while its clustering accuracy is as good as that of BK. Compared to the parallel k-means (PK) algorithm [15], PBKP has better scaleup with respect to the number of clusters (k). As k increases, PBKP reduces the average execution time per bisecting step, while PK keeps its average loop execution time almost constant.

Bibliography

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. of the 20th VLDB Conf., 1994, pp. 487–499.

[2] C. C. Aggrawal and P. S. Yu, "Finding Generalized Projected Clusters in High Dimensional Spaces," Proc. of ACM SIGMOD Int'l Conf. on Management of Data, 2000, pp. 70–81.

[3] J. Allan, "HARD Track Overview in TREC 2003 High Accuracy Retrieval from Documents," Proc. of the 12th Text Retrieval Conference, 2003, pp. 24–37.

[4] M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press, 1973.

[5] F. Beil, M. Ester, and X. Xu, "Frequent Term-based Text Clustering," Proc. of ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 2002, pp. 436–442.

[6] L. Bottou and Y. Bengio, "Convergence Properties of the k-means Algorithms", Advances in Neural Information Processing Systems, Vol.7, 1994, pp. 585–592.

[7] P. Bradley and U. Fayyad, "Refining Initial Points for K-Means Clustering," *Proc. of the 15th Int'l Conf. on Machine Learning – ICML'98*, 1998, pp. 91–99.

[8] Chris Buckley and Alan F.Lewit, *Optimizations of inverted vector searches, SIGIR'85,* 1985, pp. 97 – 110.

[9] Classic data set, ftp://ftp.cs.cornell.edu/pub/smart/.

[10] A Clustering Toolkit, Release 2.1.1, http://www.cs.umn.edu/∼karypis/cluto/.

[11] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley, 1991.

[12] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken, "LogP: A Practical Model of Parallel Computation," *Communications of the ACM*, Vol. 39, No. 11, 1996, pp. 78–85.

[13] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, "Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections," Proc. of Annual ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.

[14] A.P. Dempster, N.M. Laird and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", Journal of the Royal Stat. Society, 39, pp. 1—38.

[15] I. S. Dhillon and D. S. Modha, "A Data-Clustering Algorithm on Distributed Memory Multiprocessors," in *Large-Scale Parallel Data Mining*, LNCS 1759, Springer-Verlag, 2000.

[16] M. Farach, "Optimal Suffix Tree Construction with Large Alphabets," Proc. of the 38th Annual Symp. on Foundation of Computer Science, 1997, pp. 137–143.

[17] M. Farach, P. Ferragina and S. Muthukrishnan, "Overcoming the Memory Bottleneck in Suffix Tree Construction," Proc. of the 39th Symp. on Foundations of Computer Science, 1998, pp. 174–185.

[18] W. Frawley, G. Piatetsky-Shapiro and C. Matheus, "Knowledge Discovery in Databases: An Overview", *AI Magazine*, Fall 1992, pp. 213-228.

[19] Frequent Itemset-based Hierarchical Clustering (FIHC) Software, http://www.cs.sfu.edu.ca/~ddm/dmsoft/Clustering/fihc_index.html.

[20] George Forman, "Feature Selection: We've Barely Scratched the Surface", IEEE Intelligent Systems, November 2005.

[21] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical Document Clustering Using Frequent Itemsets," Proc. of SIAM Int'l Conf. on Data Mining, 2003.

[22] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical Document Clustering," *The Encyclopedia of Data Warehousing and Mining*, John Wang (ed.), Idea Group, 2005.

[23] M. R. Garey, D. S. Johnson, and H. S. Witsenhausen, "Complexity of the Generalized Lloyd-Max Problem," *IEEE Trans. on Information Theory*, Vol. 28, No. 2, 1982, pp. 256–257.

[24] R. Giegerich and S. Kurtz, "From Ukkonen to McCreight and Weiner: A Unifying View of Linear-time Suffix Tree Construction," Algorithmica, Vol. 19, No. 3, 1997, pp. 331–353.

[25] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[26] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes", *Information Systems*, Vol. 25, No.5, 2000, pp. 345 – 366.

[27] High Accuracy Retrieval from Documents (HARD) Track of Text Retrieval Conference, 2004.

[28] J. A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, 1975.

[29] J. D. Holt and S. M. Chung, "Multipass Algorithms for Mining Association Rules in Text Databases," Knowledge and Information Systems, Vol. 3, No. 2, Springer-Verlag, 2001, pp. 168–183.

[30] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, 1988.

[31] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, Vol. 31, No. 3, 1999, pp. 264–323.

[32] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," Proc. of the 10th. European Conf. on Machine Learning, 1998, pp. 137–142.

[33] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley-Interscience, 1990.

[34] D. Koller and M. Sahami, "Toward Optimal Feature Selection," Proc. of Int'l Conf. on Machine Learning, 1996, pp. 284–292.

[35] S. Kurtz, "Reducing the Space Requirement of Suffix Trees," Software: Practice and Experience, Vol. 29, No. 13, 1999, pp. 1149–1171.

[36] B. Larsen, C. Aone, "Fast and Effective Text Mining Using Linear-time Document Clustering," Proc. of ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 1999, pp. 16-22.

[37] The Lemur Toolkit for Language Modeling and Information Retrieval, http://www-2.cs.cmu.edu/~lemur/.

[38] M. Dash and Huan Liu, "Feature Selection for Classification", International Journal of Intelligent Data Analysis, 1997, 1(3), pp. 131–156

[39] M. Dash and Huan Liu, "Feature Selection for Clustering", Proc. of PAKDD-00, 2000, pp. 110–121.

[40] B. Liu, C. W. Chin, and H. T. Ng, "Mining Topic-Specific Concepts and Definitions on the Web," Proc. of the 12th Int'l conf. on World Wide Web, 2003, pp. 251–260.

[41] Tao Liu, Shengping Liu, Zheng Chen and Weiying Ma, "An Evaluation on Feature Selection for Text Clustering", Proc. of the 20th International Conference on Machine Learning, 2003.

[42] C. Manning and H. Schutze, *Foundations of Statistical Natural Language processing*, MIT Press, 1999.

[43] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," Proc. of AAAI-98 Workshop on Learning for Text Categorization, 1998.

[44] E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," Journal of ACM, Vol. 23, No. 2, 1976, pp. 262–272.

[45] G. Milligan, "An Algorithm for Creating Artificial Test Clusters," *Psychometrika*, Vol. 50, No. 1, 1985, pp. 123–127.

[46] H. Ahonen-Myka, "Finding All Maximal Frequent Sequences in Text," Proc. of ICML-99 Workshop on Marchine Learning in Text Data Analysis, 1999, pp. 11–17.

[47] H. Ahonen-Myka, "Discovery of Frequent Word Sequences in Text," Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining, 2002, pp. 16–19.

[48] A. Doucet and H. Ahonen-Myka, "Non-contiguous Word Sequences for Information Retrieval," Proc. of 42nd Annual Meeting of the Association for Computational Linguistics (ACL-2004) Workshop on Multiword Expressions and Integrating Processing, 2004, pp. 88–95.

[49] M. Nelson, "Fast String Searching with Suffix Trees," Dr. Dobb's Journal, August, 1996.

[50] K. Nigam, J. Lafferty, and A. McCallum, "Using Maximum Entropy for Text Classification," Proc. of IJCAI-99 Workshop on Maching Learning for Information Filtering, 1999, pp. 61–67.

[51] C. Ordonez and E. Omiecinski, "Efficient Disk-Based K-Means Clustering for Relational Databases," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 16, No. 8, 2004, pp. 909–921.

[52] Yiming Yang and Jan O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization", Proc. of International Conference on Machine Learning, 1997, pp. 412–420.

[53] G.H. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem", Proc. of International Conference on Machine Learning, 1994, pp. 121–129.

[54] J. R. Quinlan, "Induction of Decision Trees," Machine Learning, Vol. 1, 1986, pp. 81–106.

[55] Reuters-21578 Distribution 1.0, http://www.daviddlewis.com/resources/testcollections/reuters21578.

[56] C. J. van Rijsbergen, *Information Retrieval*, 2nd edition, Buttersworth, London, 1979.

[57] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1988.

[58] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, July and October, 1948, pp. 379–423 and pp. 623–656.

[59] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," KDD-2000 Workshop on Text Mining, 2000.

[60] Y. Seo, A.Ankolekar, and K.Sycara, "Feature Selection for Extracting Semantically Rich Words", Tech. Report CMU-RI-TR-04-18, Robotics Institute, Carnegie Mellon University, 2004.

[61] UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html.

[62] E. Ukkonen, "On-line Construction of Suffix Trees," Algorithmica, Vol. 14, 1994, pp. 249–260.

[63] K. Wang, C. Xu, and B. Liu, "Clustering Transactions Using Large Items," Proc. of the 8th Int'l Conf. on Information and Knowledge Management, 1999, pp. 483–490.

[64] P. Weiner, "Linear Pattern Matching Algorithms," Proc. of the 14th Annual Symp. on Foundation of Computer Science, 1973, pp. 1–11.

[65] W.J. Wilbur and K. Sirotkin, "The Automatic Identification of Stop Words", Journal of Information Science, Vol.18,1992, pp. 45–55.

[66] C. Fellbaum (Ed.), *WordNet: An Electronic Lexical Database*, MIT Press, May, 1998.

[67] Yiming Yang, "Noise Reduction in a Statistical Approach to Text Categorization", Proc. of SIGIR'95, 1995, pp. 256–263.

[68] Charu C. Aggrawal and Philip S. Yu, "Finding Generalized Projected Clusters in High Dimensional Spaces", Proc. of SIGMOD'00, 2000, pp. 70–81.

[69] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp, "Fast and Intuitive Clustering of Web Documents," Proc. of the 3rd Int'l Conf. on Knowledge Discovery and Data Mining, 1997, pp. 287–290.

[70] O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration," Proc. of Annual ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 46–54,

[71] Y. Zhao and G. Karypis, "Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering," Machine Learning, Vol. 55. No. 3, 2004, pp. 311–331.