Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 64 (2008) 381–404
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Text document clustering based on frequent word meaning sequences ☆

Yanjun Li [a], Soon M. Chung [b,*], John D. Holt [b]

[a] Department of Computer and Information Sciences, Fordham University, Bronx, NY 10458, USA
[b] Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435, USA

Received 7 December 2006; received in revised form 11 July 2007; accepted 9 August 2007
Available online 30 August 2007

Abstract

Most of existing text clustering algorithms use the vector space model, which treats documents as bags of words. Thus, word sequences in the documents are ignored, while the meaning of natural languages strongly depends on them. In this paper, we propose two new text clustering algorithms, named Clustering based on Frequent Word Sequences (CFWS) and Clustering based on Frequent Word Meaning Sequences (CFWMS). A word is the word form showing in the document, and a word meaning is the concept expressed by synonymous word forms. A word (meaning) sequence is frequent if it occurs in more than certain percentage of the documents in the text database. The frequent word (meaning) sequences can provide compact and valuable information about those text documents. For experiments, we used the Reuters-21578 text collection, CISI documents of the Classic data set [Classic data set, ftp://ftp.cs.cornell.edu/pub/smart/], and a corpus of the Text Retrieval Conference (TREC) [High Accuracy Retrieval from Documents (HARD) Track of Text Retrieval Conference, 2004]. Our experimental results show that CFWS and CFWMS have much better clustering accuracy than Bisecting *k*-means (BKM) [M. Steinbach, G. Karypis, V. Kumar, A Comparison of Document Clustering Techniques, KDD-2000 Workshop on Text Mining, 2000], a modified bisecting *k*-means using background knowledge (BBK) [A. Hotho, S. Staab, G. Stumme, Ontologies improve text document clustering, in: Proceedings of the 3rd IEEE International Conference on Data Mining, 2003, pp. 541–544] and Frequent Itemset-based Hierarchical Clustering (FIHC) [B.C.M. Fung, K. Wang, M. Ester, Hierarchical document clustering using frequent itemsets, in: Proceedings of SIAM International Conference on Data Mining, 2003] algorithms.

Keywords: Text documents; Clustering; Frequent word sequences; Frequent word meaning sequences; Web search; WordNet

1. Introduction

Nowadays in every industry, almost all the documents on papers have their electronic copies. This is because the electronic format provides safer storage and occupies much smaller space. Also, the electronic

☆ This research was supported in part by AFRL/Wright Brothers Institute (WBI).
* Corresponding author. Tel.: +1 937 775 5119; fax: +1 937 775 5133.
E-mail address: soon.chung@wright.edu (S.M. Chung).

0169-023X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2007.08.001

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 important for the search engine.

In order to increase the precision of the retrieval result, many methods have been proposed [2]. 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 high similarity between them, but they are dissimilar to documents in other clusters [13]. It is different from text classification because there is no training stage by using labeled documents.

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

- Finding a suitable model to represent the document is a non-trivial issue. Most of the text documents are written in a human 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.
- The high dimension of text documents should be reduced. Usually there are about 200–1000 unique words in a document such as a newspaper article. In order to efficiently process a huge text database like WWW, the text clustering algorithm should have a way to reduce the high dimension.
- Overlapping between document clusters should be allowed because a document can cover several topics. For example, morning news may have information about a war followed by information about the popularity of teas in US.
- 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.
- 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 number of clusters to the clustering algorithm, 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 showing in the documents. A ''word meaning'' is the lexicalized concept that a word form can be used to express [11]. The key features of our CFWS and CFWMS algorithms are: they treat the text document as a sequence of words (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 algorithms to find the frequent word (meaning) sequences from a text database.

Finding frequent itemsets is an important data mining 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 did not adopt the vector space model. In our algorithms, each document is reduced to a compact document by keeping only the frequent words (meanings). In the compact document, we keep the sequential occurrence of words (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 describe their content for the user. The difference between CFWS and CFWMS is: in CFWMS, we convert words into word meanings by exploring the synonymy, polysemy, and hyponymy/hypernymy relationships between words by using an ontology — WordNet [11]. We found CFWMS is more accurate than CFWS, and both of them are more accurate than bisecting k-means (BKM) [33], a modified bisecting k-means using background knowledge (BBK) [19] and Frequent Itemset-based Hierarchical Clustering (FIHC) [12] algorithms.

The rest of this paper is organized as follows: Section 2 introduces the related work on text clustering, mining of frequent word sequences, and the application of background knowledge to text clustering. Section 3 describes the method to find frequent word sequences from a text database in details, and Section 4 describes the CFWS clustering algorithm. Section 5 describes the CFWMS clustering algorithm based on CFWS and an ontology, named WordNet [11]. The experimental results of CFWS and CFWMS as well as their performance comparison with other clustering algorithms are presented in Section 6. Section 7 contains some conclusions.

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 [8]. 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 [5,22,24], 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. These two categories are compared in [33].

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [8] of AHC is reported to be the most accurate one in its category. Bisecting k-means algorithm, combining the strengths of partitioning and hierarchical clustering methods, is reported to outperform the basic k-means as well as the agglomerative approach in terms of accuracy and efficiency. In bisecting k-means algorithm, initially the whole database 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. 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 [35], they proposed a new criterion for clustering transactions using frequent itemsets, instead of using a distance function. The FTC algorithm introduced in [3] used the shared frequent word sets between documents to measure their closeness in text clustering. The Frequent Itemset-based Hierarchical Clustering (FIHC) algorithm proposed in [12] 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 clusters 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 stopwords 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 paper, we propose a new model to represent the document. The sequential relationship between words (meanings) 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. In [37], the idea of using word sequences (phrases) for text clustering was proposed; and then the Suffix Tree Clustering (STC) based on this idea was proposed in [38]. 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. And STC just performs the word form matching, which ignores the semantic and lexical relationships between words.

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 [7,28,29] 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.

Due to the usage of synonyms, many phrases having the same meanings may not be recognized during the clustering process. However, there are only few methods proposed to utilize the semantic relationships between words in text clustering. In [6], the Universal Networking Language (UNL) is used for the semantic representation of sentences, and the clustering is performed by using a neural network, called Kohonen's self-organizing feature maps. The algorithms proposed in [19,32] use the vector space model, apply the background knowledge from WordNet in the preprocessing of the text documents, and perform the bisecting

k-means algorithm for clustering. The modified bisecting k-means using the background knowledge [19], which is denoted as BBK algorithm in this paper, enriches the text representation by adding synonyms and up to five levels of hypernyms for all nouns in the documents. In [32], they studied the impact of tagging the documents with the part-of-speech tags and adding all hypernyms to the information available for each document. The techniques proposed in [19,32] add all available information to the representation of the text documents, so that the dimension of the text database is increased, and additional noise is added by the incorrect senses retrieved from WordNet.

Our CFWMS algorithm treats a document as a string of word meanings instead of a bag of words, and uses frequent word meaning sequences to cluster the documents. Since nouns and verbs are important members of meaningful word sequences in natural language, the word forms of nouns and verbs are converted into word meanings in the preprocessing step of CFWMS. In this way, a document is treated as a sequence of word meanings which represent nouns and verbs, and only the frequent word meaning sequences contribute to the final clustering result. Since one word meaning may represent several word forms, and infrequent word meanings are removed from the documents, this technique can reduce the dimension of the text databases for the clustering process.

3. Finding frequent word sequences

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 $d = \langle w_1, w_2, w_3, \ldots \rangle$, 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. In the rest of this paper, all the words in frequent 2-word sets will be called *frequent words*.

An ordered sequence of two or more words is called a *word sequence*. A word sequence S is represented as $\langle w_1, w_2, \ldots \rangle$. A *frequent word sequence* is denoted by FS in this paper. For example, $FS = \langle w_1, w_2, w_3, w_4 \rangle$, 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 [38] 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 $FS = \langle w_1, w_2, \ldots, w_k \rangle$, and it has two frequent subsequences of length $k-1$, which are $\langle w_1, w_2, \ldots, w_{k-1} \rangle$ and $\langle w_2, w_3, \ldots, w_k \rangle$. For example, $FS = \langle he, play, basketball \rangle$ has two frequent subsequences of length 2, which are $\langle he, play \rangle$ and $\langle play, basketball \rangle$. Please note that $\langle he, basketball \rangle$ 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.*

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.

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 $k \geqslant 2$. 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, $k \geqslant 2$, must be in WS.

After finding the frequent 2-word sets, we remove all the words in the documents that are not in WS. After the removal, the resulting documents are called *compact documents*. Let us consider an 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 $WS = \{\text{young, boys, play, basketball}\}$. After removing those words not in WS, the database D becomes $D' = \{d'_1, d'_2, d'_3\}$ as follows, where the removed words are shown in parentheses.

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

In 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'.

3.2.2. Building a generalized suffix tree (GST)

Our goal is to find the frequent word sequences of the database. We adopted the suffix tree [36], a well-known data structure for sequence pattern matching, to find all the 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 the GST, we can find all the frequent word sequences of the 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 paper, 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 a non-empty substring of a string. 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 document id set of the node. These document ids are used to check the multiple occurrences of a sequence in the same document.

Fig. 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 it 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 the label (i.e., $string_L$) 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 the label of each node, we can get the information about all the frequent word sequences in the database. In our example shown above, we have seven nodes in the GST, and the details are given in Table 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, as the minimum length of the word sequence is set to 2, 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 [28], 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

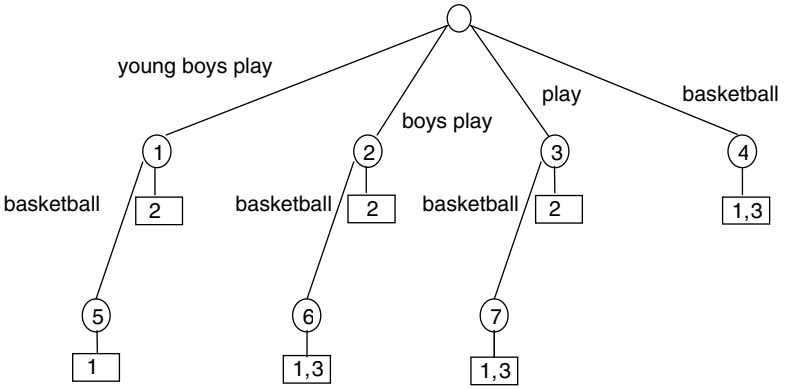

Fig. 1. Generalized suffix tree for the compact documents with seven nodes.

Table 1
Word sequences associated with the nodes in Fig. 1

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

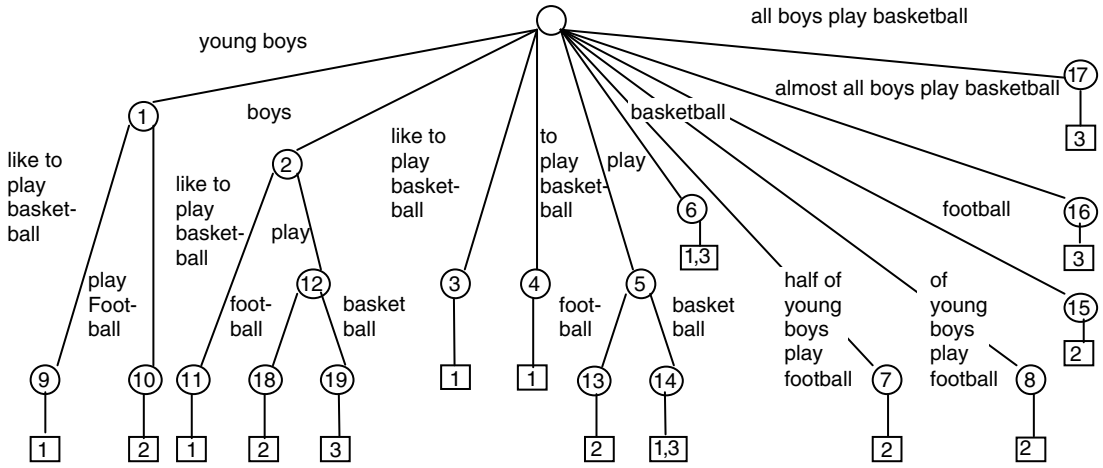

Fig. 2. Suffix tree for the original documents with 16 nodes.

the content of documents d_1 and d_3. 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 [38], 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 Fig. 2 shows the suffix tree for the example database D. Comparing this suffix tree of the STC algorithm with our suffix tree shown in Fig. 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.

4. Clustering based on frequent word sequences (CFWS)

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.

4.2. CFWS algorithm

Our CFWS algorithm has three steps: building a GST to find frequent word sequences, merging cluster candidates based on the k-mismatch concept, and then combining the overlapping clusters to obtain the final clusters.

4.2.1. Finding frequent word sequences and collecting the cluster candidates

We use the method explained in Section 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.

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

4.2.2. Merging the cluster candidates based on the k-mismatch concept

The *FS* of a cluster candidate describes what the cluster candidate is about. For example, from $FS_2 =$ "boys play", we know that cc_2 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. For that purpose, 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 the sequential patterns.

The original k-mismatch concept is about characters and strings: 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 in t is a $|p|$-substring of t that matches $(|p| - k)$ characters of p. That is, it matches p with k mismatches. In our case, the given pattern p and text t are sequences of words instead of strings of characters. A k-mismatch of p in t is a $|p|$-subsequence of t that matches $(|p| - k)$ words of p. Since the *FS* of a cluster candidate can represent its topic, we believe the closeness between two cluster candidates could be measured by their *FS*s. The idea is that two cluster candidates with k-mismatched *FS*s may cover similar topics. For example, we can say that the topic covered by cc_2 with $FS_2 =$ "boys play" is similar to the topic covered by cc_4 with $FS_4 =$ "play basketball". So, we can merge them to have a bigger cluster, which covers the topic about sports.

First, we check the mismatches between the frequent word sequences found by building the GST of the document collection. We are interested in 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_i, it becomes the shorter pattern FS_j. 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: $\{|FS_i| = |FS_j| - k \mid FS_i =$ "boys play"; $FS_j =$ "boys play basketball"$\}$;
  By inserting a word "basketball" in FS_i, $FS_i = FS_j$;
- Deletion: $\{|FS_i| = |FS_j| + k \mid FS_i =$ "boys play"; $FS_j =$ "play"$\}$;
  By deleting a word "boys" from FS_i, $FS_i = FS_j$;
- 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", $FS_i = FS_j$;

Starting with the longest frequent word sequence, denoted by FS_L, we adopt the Landau–Vishkin (LV) algorithm [23] to find all the k-mismatched patterns for FS_L for a given k. The reason of starting with FS_L is that cluster candidates with long *FS*s more likely cover subtopics, while those with short *FS*s more likely cover general topics. Our goal is to merge the cluster candidates with subtopics into clusters with more general topics.

Second, we merge $cc_L[FS_L, Ids_L]$ with all $cc_k[FS_k, Ids_k]$, where each FS_k is a k-mismatch pattern of FS_L, into a new cluster. Then, cc_L and all cc_k are removed from the set of all initial cluster candidates, S_{cc}. 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. In our experiments of clustering text documents, we found the clustering results are better when k is set to 1. These two steps are repeated until S_{cc} becomes empty.

4.2.3. Combining the overlapping clusters

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

$$O(C_i, C_j) = \frac{Ids_i \cap Ids_j}{Ids_i \cup Ids_j} \quad (1)$$

If $O(C_i,C_j)$ is larger than the specified overlap 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. This step is controlled by two parameters: the overlap threshold value δ and the optional user-specified number of final clusters *KCluster*. If the user specifies *KCluster*, two clusters with the highest overlap value are merged repeatedly until the number of clusters becomes *KCluster*. Otherwise, two clusters can be merged only when their overlap is larger than δ. The experimental result given in Section 6.4 shows that, when *KCluster* is not specified, our CFWS algorithm performs the best when $\delta = 0.5$, and the clustering accuracy is not sensitive to δ if δ is higher than 0.5.

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 ''unrelated 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 frequent words, each of which is a member of a frequent 2-word set.
2. Reduce each document d_i, $1 \leqslant i \leqslant n$, 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 a set of document ids Ids_j, create a cluster candidate $cc_j[FS_j, Ids_j]$.
5. Merge similar cluster candidates into clusters based on the k-mismatch concept for a given k.
6. Combine clusters based on the overlap threshold δ and the optional user-specified number of final clusters.

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 [11]. We believe that 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.

5.1. Term definitions

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 paper, a *synset* is denoted by SS, e.g. $SS_1 = \{\text{car, auto}\}$. 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. When the word form matching is performed to find frequent words in a text database, 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 4 other

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 hypernym of {vehicle} is {conveyance, transport}. 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. If we use "→" to represent this relationship, these three synsets could be linked as {car, railcar} → {vehicle} → {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*). All the synset members of a synset link are said to *belong* to the synset link, and the synset link is said to *contain* these synsets, which is denoted as $SS_i \in SL_j$.

In CFWMS, we want to convert word forms in the documents to the word meanings they express, so that documents containing synonyms, hyponyms, and hypernyms all contribute to the support count of the same word meaning. In this way, the number of documents containing "car", "auto" and "vehicle" are all counted as the support count of one word meaning. After the conversion, each text document is treated as a sequence of word meanings, then a text document d can be viewed as $d = \langle SS_1, SS_2, SS_3, \ldots \rangle$. 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 paper.

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. The synset link members are said to *belong* to the meaning union, and the meaning union is said to *contain* these synset links, which is denoted as $SL_j \in MU_h$. For a synset SS_i, if $SS_i \in SL_j$ and $SL_j \in MU_h$, then we say SS_i *belongs* to MU_h, and MU_h *contains* SS_i, which is denoted as $SS_i \in MU_h$. For example, for a word form "box" in a document, there is a meaning union $MU_1 = \{SL_1, SL_2\}$, where $SL_1 = \{SS_1 \rightarrow SS_2\}$, $SL_2 = \{SS_3 \rightarrow SS_4\}, SS_1 = \{\text{box}\}$, $SS_2 = \{\text{container}\}$, $SS_3 = \{\text{box, loge}\}$, and $SS_4 = \{\text{compartment}\}$. We expect that one of the synsets belonging to MU_1 is the real word meaning expressed by the word form "box" in this document.

In CFWMS, the word forms in each document are converted into meaning unions, like $d' = \langle MU_1, MU_2, MU_3, \ldots \rangle$. In this case, a set of meaning unions is frequent if its support is at least the user-specified minimum support, and a frequent set of k meaning unions is the one containing k meaning unions. In the rest of this paper, each meaning union appearing in any frequent set of 2 meaning unions is called a *frequent meaning union*. A frequent word meaning sequence could also be represented as a sequence of frequent meaning unions, like $FMS = \langle MU_1, MU_2, \ldots \rangle$. The synsets belong to the frequent meaning unions are called *frequent synsets*.

5.2. CFWMS algorithm

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

5.2.1. 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 [11], 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. 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 $d_1 = \langle w_1, w_2, w_3 \rangle$ can be converted to a sequence of meaning unions as $d_1' = \langle MU_1, MU_2, MU_3 \rangle$, 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. After the retrieval in the first pass, many meaning unions have more than one synset link since WordNet assigns more than one word meaning to a word form. Multiple synset links introduce noise to the database, and affect the accuracy of the clustering. Thus, we need to estimate the real word meaning for each word form. In natural language, the word meaning of a word form is determined by its lexical environment; in other words, the words around it. And the word forms in one document may tend to express similar word meanings. Our estimation strategy is to keep those synset links containing the synsets with high frequencies in the document. 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 support counts of the synsets in the document. The support count of a synset in a document is defined as the number of meaning unions containing the synset in the document. For the replacement of meaning unions, 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 shows the details of each replacement step for the given example. The first three columns in Table 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.

Table 2
Reorganizing the meaning unions in document d_1'

Synsets	$d_1' = \langle MU_1, MU_2, MU_3 \rangle$		Step 1 (pick SS_5) $d_1' = \langle MU_4, MU_2, MU_4 \rangle$		Step 2 (pick SS_3) $d_1' = \langle MU_4, MU_2, MU_4 \rangle$	
	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	$1 \Rightarrow 0$	

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.

5.2.2. Finding frequent word meaning sequences and collecting the cluster candidates

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

- $d'_1 = \langle MU_4, MU_2, MU_4 \rangle$
  - $MU_4 = \{\{SS_4, SS_1\} \rightarrow SS_5\}$
  - $MU_2 = \{SS_3 \rightarrow SS_8, SS_9 \rightarrow SS_{10}\}$

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

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

In order to find frequent word meaning sequences, we use an association rule miner to find the frequent sets of two meaning unions first, and then the frequent synsets. If we perform exact matching between meaning unions, we cannot find any frequent sets of two 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 $\langle SS_5, SS_8 \rangle$.

In order to find the frequent word meaning sequences, we need to check and collect the counts of synset links, instead of just matching the word meaning unions. The support count of a synset link is defined as the number of meaning unions which contain a synset that belongs to the synset link. For instance, when we check MU_2 of d'_1, there are two synset links in MU_2: $\{SS_3 \rightarrow SS_8\}$ and $\{SS_9 \rightarrow SS_{10}\}$. Let us denote them by SL_A and SL_B, respectively. Since a synset link could be treated as a meaning union containing only one synset link, 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 the count of SL_A by one again. Table 3 shows the support counts of unique meaning unions (as synset links) in database D' after we checked all three documents. In this example, $\{SL_C, SL_A\}$ is a frequent set of 2 meaning unions, SL_C and SL_A are frequent meaning unions, and the set of frequent synsets is $\{SS_1, SS_2, SS_4, SS_5, SS_3, SS_8, SS_{12}, SS_6, SS_{13}\}$.

Table 3
Support counts of the meaning unions in database D'

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

After finding the frequent synsets, 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:

- $d''_1 = \langle SL_C,\ SL_A,\ SL_C \rangle$
- $d''_2 = \langle SL_C,\ SL_A \rangle$
- $d''_3 = \langle SL_A \rangle$

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 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, a cluster candidate we obtain is $cc[FMS = \langle SL_C, SL_A \rangle, Ids = \{1,2\}]$.

5.2.3. Combining the cluster candidates

The frequent word meaning sequence of a cluster candidate describes the topic those documents cover. In CFWMS, we do not merge the cluster candidates based on the k-mismatch concept. The reason is because the cluster candidates we found are general enough since word meanings were used, instead of word forms. We only combine the cluster candidates based on the overlap threshold δ and the optional user-specified number of final clusters, as described in the CFWS algorithm.

Our text clustering algorithm CFWMS is summarized as follows:

1. Given a collection of text documents $D = \{d_1, d_2, d_3, \ldots, d_n\}$, preprocess each document using WordNet:
   (a) By retrieving multiple synsets and hypernyms from WordNet, convert each word form into a meaning union (MU).
   (b) In order to reduce the number of unique MUs in each document, if different MUs contain the same synset, replace them with a MU containing the merged synset link.

   After steps 1a and 1b, we get $D' = \{d'_1, d'_2, d'_3, \ldots, d'_n\}$, where each d'_i is composed of MUs, like $d'_i = \langle MU_1, MU_2, MU_3, \ldots \rangle$.
2. Find frequent sets of 2 MUs of D' based on the user-specified minimum support. Instead of counting MUs, we check the support counts of SLs belonging to MUs. Obtain the set of all frequent synsets, each of which belongs to a frequent set of 2 MUs.
3. Reduce each document d'_i, $1 \leqslant i \leqslant n$, into a compact document d''_i by removing every MU that does not contain a frequent synset, and replace every MU with a frequent MU if both contain the same frequent synset.
4. Insert each compact document into the GST.
5. Using the depth-first traversing, visit every node in the GST. If a node j has a frequent word meaning sequence FMS_j with a set of document ids Ids_j, create a cluster candidate $cc_j[FMS_j, Ids_j]$.
6. Combine the overlapping cluster candidates based on the overlap threshold δ and the optional user-specified number of final clusters.

6. Experimental evaluation

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

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 nine data sets in this group. They are Re1, Re2, Re3, Re4 and Re5 from the EXCHANGES, ORGS, PEOPLE and TOPICS category sets of the Reuters-21578 Distribution 1.0; and Ce1, Ce2, Ce3 and Ce4 from the CISI, CRAN and CACM abstracts of Classic database [4].

Another group of data sets are 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 [25] as the search engine. The English newswire corpus of the HARD track of the Text Retrieval Conference 2004 [16] is used as the database. This corpus includes about 652,309 documents (in 1575 MB) from eight different sources, and there are 29 test queries. Among those 29 queries, HARD-306, HARD-309, and HARD-314 queries were sent to the search engine, and the top 200 results of them were collected and classified as three 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 4 summarizes the characteristics of all the data sets used for our experiments.

6.2. Evaluation of the finding frequent word sequences

We evaluated our method of finding frequent word sequences in terms of its scalability. Before the mining, preprocessing steps including the stop-words removal and the stemming were performed on the data sets. 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 itemset mining algorithms proposed [18], but there is not 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 [14,34], 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 construction and traverse times of the GST. Thus, by keeping only the frequent words which are members of frequent 2-word sets in the database, the construction of the GST of large 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 Re2 data set has 807 documents with 116,816 total words and 6488 unique words. Its average document size and length are 1.85 KB and 144, respectively. The Apriori algorithm finds 1049 frequent 2-word sets with 175 unique words when the minimum support is 10%. After removing infrequent words from the documents,

Table 4
Summary of data sets used for experiments

Data Set	Num. of doc.	Num. of class	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	3368	69	24,414
Re2	807	9	20	349	98	6488	144	116,816
Re3	797	15	20	168	62	5466	108	86,744
Re4	1761	31	20	349	57	8793	110	195,433
Re5	4443	39	20	801	114	13,420	113	504,386
Ce1	262	4	56	144	93	2224	64	16,944
Ce2	355	4	70	146	108	2633	65	23,070
Ce3	178	4	35	117	65	2102	60	11,639
Ce4	6064	3	1400	3204	2021	10,677	52	315,736
Se1	200	3	44	92	66	8998	329	65,868
Se2	200	4	20	98	54	12,368	736	147,390
Se3	200	3	18	152	70	9301	940	188,047

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 5, and they represent the topics of the data set very well.

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 [9,10,21,27].

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 Fig. 3, without including the time to read the data file initially. As we can see, the execution time increases linearly with the data set size.

We also tested our method for various minimum support levels on the Re2 data set, and the result is shown in Fig. 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.

6.3. Evaluation method of the 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 [31]. Since our

Table 5
Some frequent word sequences in the Re2 data set

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

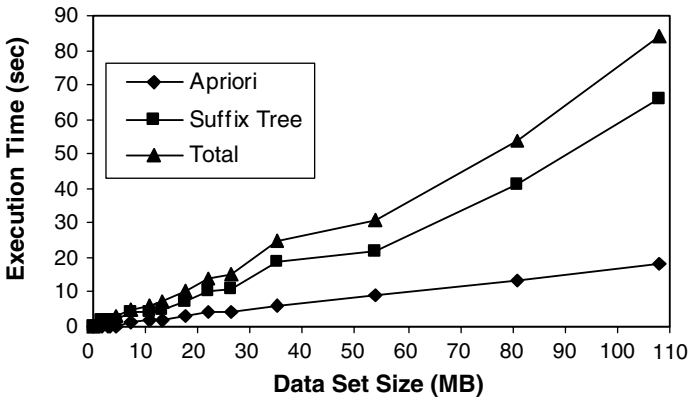

Fig. 3. Scalability of finding frequent word sequences with respect to the data set size.

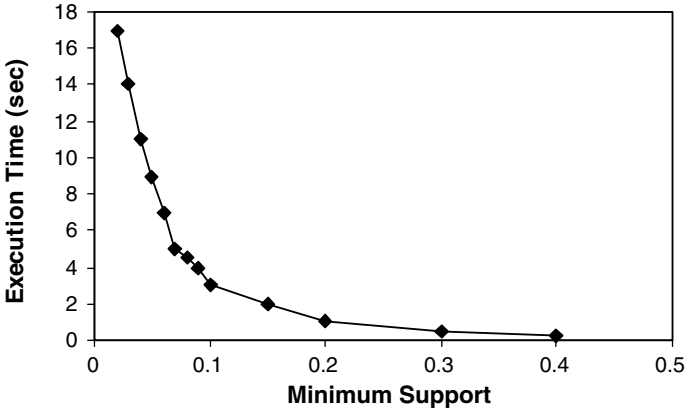

Fig. 4. Scalability of finding frequent word sequences with respect to the minimum support.

data sets were prepared as described in Section 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 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

$$P(i,j) = \frac{n_{ij}}{n_j} \tag{2}$$

$$R(i,j) = \frac{n_{ij}}{n_i} \tag{3}$$

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

$$F(i,j) = \frac{2 * P(i,j) * R(i,j)}{P(i,j) + R(i,j)} \tag{4}$$

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

$$F = \sum_i \frac{n_i}{n} \max_j (F(i,j)) \tag{5}$$

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 [33].

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

$$\text{Purity}(j) = \frac{1}{n_j} \max_i (n_{ij}) \tag{6}$$

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

$$\text{Purity} = \sum_j \frac{n_j}{n} \text{Purity}(j) \tag{7}$$

In general, the larger the purity value is, the better the clustering result is [39].

6.4. Comparison of CFWS with other algorithms

For a comparison with our CFWS, we also executed bisecting k-means (BKM) and FIHC on the same data sets. We chose BKM because it has been reported to produce a better clustering result consistently compared to k-means and agglomerative hierarchical clustering algorithms [33]. FIHC is chosen because, like CFWS, it

uses frequent word sets. For a fair comparison, we did not implement BKM and FIHC algorithms by ourselves. We downloaded the CLUTO toolkit [20] to perform BKM, and obtained FIHC version 1.0 [17] from the inventor of FIHC.

Data sets were preprocessed before they were used in our experiments. First, we removed the stop-words from the documents. Then, the words were stemmed by using the Porter's suffix-stripping algorithm [33]. It is important to do the stemming since it can eliminate the minor difference between words with the identical meaning.

Fig. 5 shows the average *F*-measures of three algorithms: BKM, FIHC and CFWS. For all three algorithms, we specified the desired number of final clusters, denoted by *KCluster*, to be the same as the number of classes in each data set. We executed BKM 11 times on each data set with randomly selected initial centroids and calculated the average *F*-measure. FIHC and CFWS were also executed 11 times on each data set; and for each run, we specified the same minimum support level, in the range of 5–15%, for both algorithms so that the comparison would be fair. Based on the average *F*-measures, it is clear that our CFWS algorithm consistently outperforms two other algorithms on the sets of typical text documents as well as on the sets of search query results. Fig. 6 shows the effect δ on the *F*-measure of CFWS when *KCluster* is not specified. As we can see, the *F*-measure is not sensitive to δ if δ is higher than 0.5.

The *F*-measure represents the accuracy of clustering. Our CFWS algorithm has better *F*-measures because we use a better model for text documents. Both BKM and FIHC use the vector space model for text documents. However, the vector space model cannot capture the order of words, which is important in representing

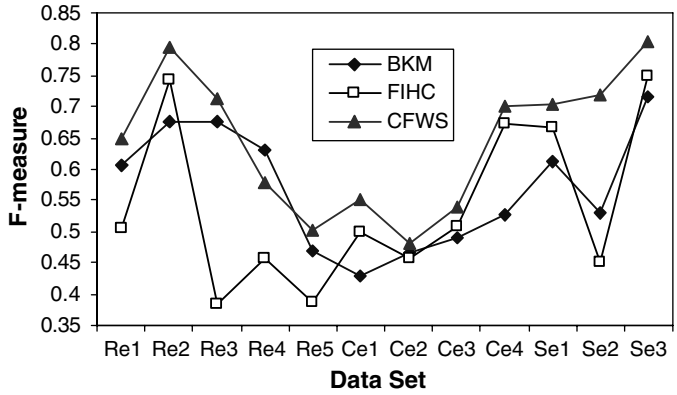

Fig. 5. *F*-measures of CFWS, BKM and FIHC.

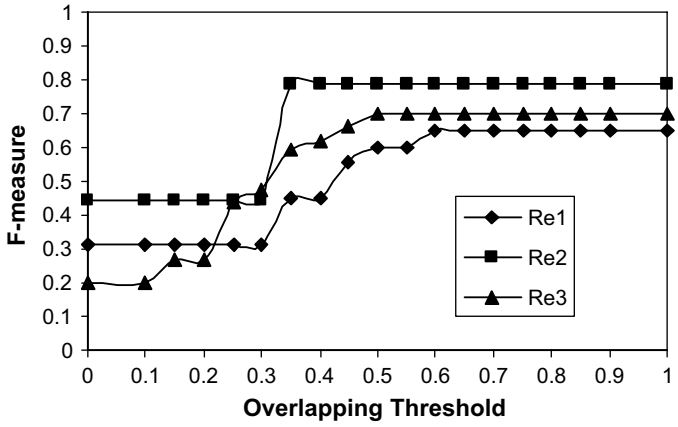

Fig. 6. Effect of the overlapping threshold δ on the *F*-measure of CFWS.

the context in the text document. On the contrary, our model represents the words as well as their orders, which provide more valuable information for clustering.

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.

Table 6 shows the F-measures of FIHC and CFWS on Re1, Ce1 and Se1 data sets when *KCluster* is not specified and the minimum support levels are 5%, 10% and 15%, respectively. We can see that CFWS produces more accurate clusters at different minimum support levels.

Fig. 7 shows the average purity values of the three clustering algorithms when the *KCluster* value specified is the same as the number of classes in each data set. As we can see, BKM performs better than both FIHC and CFWS for almost all of the typical text document sets: Re1, Re2, Re3, Re4, Re5, Ce1, Ce2, Ce3 and Ce4. 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, hypernyms, and polysemous words are used. By using WordNet, the lexical and semantic

Table 6
F-measures of FIHC and CFWS with different minimum support levels

Data set	Minimum support (%)	*F*-measure	
		FIHC	CFWS
Re1	5	0.361	0.551
	10	0.678	0.708
	15	0.578	0.641
Ce1	5	0.503	0.571
	10	0.501	0.516
	15	0.410	0.508
Se1	5	0.504	0.629
	10	0.685	0.754
	15	0.589	0.691

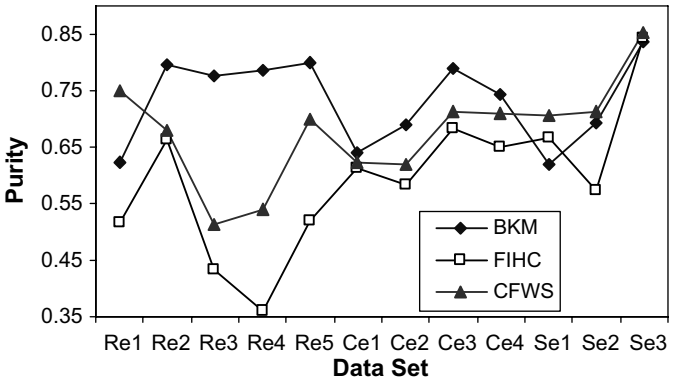

Fig. 7. Purity values of CFWS, BKM and FIHC.

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. The experimental results of CFWMS are shown in the next section.

For the search query results (i.e., Se1, Se2 and Se3), CFWS performs better than BKM 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 [25] to retrieve the documents for each query. The retrieved documents are more likely to have common words between them compared to the set of typical text documents. In other words, the topics covered by this type of document sets are much closer than those of the typical text document sets. For example, top 200 documents on the retrieval list for the query of "tea consumption increased in US" cover three 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 7. The class labels are the frequent word sequences we found during the clustering process. The words in the frequent word sequences are in the original form before the stemming. 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 between 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.

6.5. Comparison of CFWMS with other algorithms

For a comparison with our CFWMS algorithm, we implemented the modified bisecting k-means using background knowledge (BBK) algorithm proposed in [19]. BBK uses the vector space model and enriches the text representation by adding synonyms and up to five levels of hypernyms for all nouns based on the context in the document. For both CFWMS and BBK, WordNet was used as the ontology.

The stop-words were removed as the first step of preprocessing. However, since WordNet provides the morphological capability, we did not perform the stemming for CFWMS and BBK.

In CFWMS, the knowledge from WordNet is used to convert word forms to word meanings. As a result, the dimension of text database is reduced further than the case of CFWS, and Fig. 8 shows the changes of the dimension in test data sets. On the other hand, BBK adds synsets to the vector space model of the database, hence the dimension of the text database is increased after the preprocessing step.

For a fair comparison, we specified the desired number of clusters, *KCluster*, to be the same as the number of classes in each data set. We executed BBK 11 times on each data set with randomly selected initial centroids. CFWS and CFWMS were also executed 11 times on each data set with the same minimum support level, in the range of 5–15%. The performances of CFWS, CFWMS, and BBK are compared in Figs. 9 and 10. CFWMS has better average F-measures than that of CFWS in most cases because it can identify the same word meaning sequences represented by different word form sequences. The average purity values

Table 7
Classification of the Se1 data set

Class	Label	Subclass	Subclass label
1	"Tea consumption"	1.1	"Tea culture"
		1.2	"Tea health"
		1.3	"Tea popularity"
2	"Tea industry"	2.1	"Tea grow", "Tea produce"
		2.2	"Tea export"
		2.3	"Tea auction", "Tea broker"
3	Unrelated issues	3.1	"Cricket tea" (sport)
		3.2	"Drink problem"
		3.3	Others

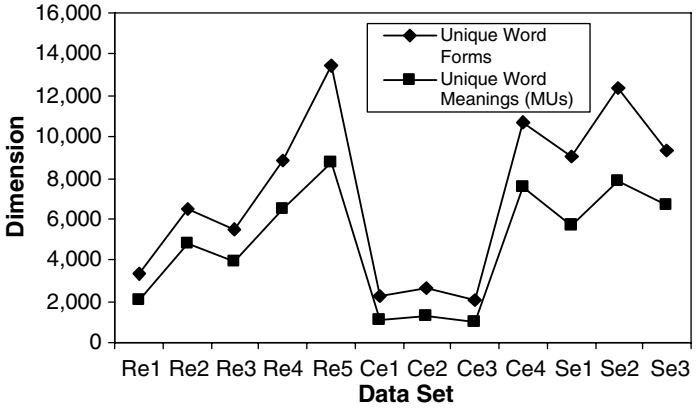

Fig. 8. Dimensionality changes after applying WordNet in CFWMS.

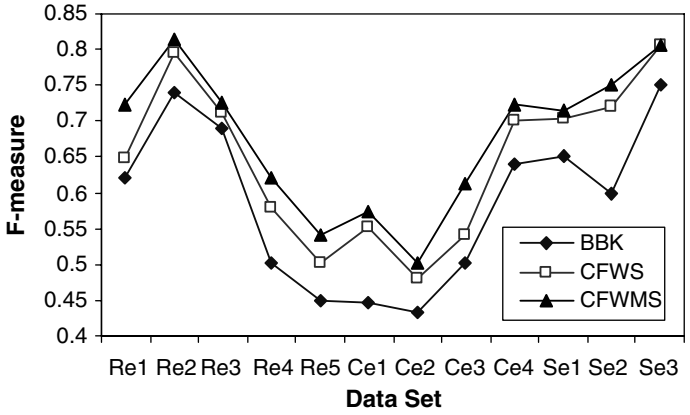

Fig. 9. *F*-measures of CFWMS, CFWS and BBK.

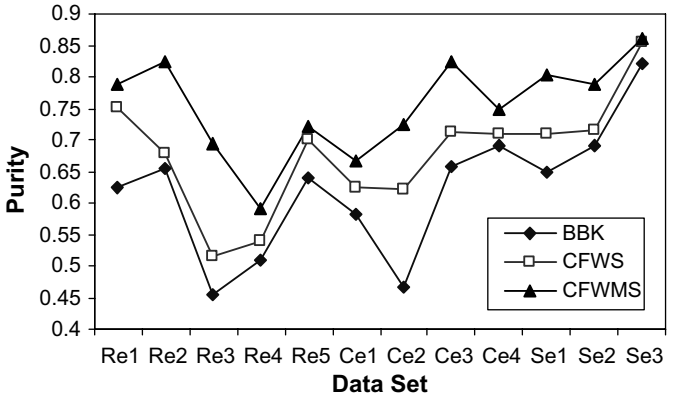

Fig. 10. Purity values of CFWMS, CFWS and BBK.

show that CFWMS can improve the clustering quality for both typical text documents and search query results.

This comparison shows that our CFWMS outperforms BBK on all three groups of test data sets. CFWMS utilizes the frequent sequences of word meanings in the clustering process, which can achieve more accurate

clustering result than the vector space model of BBK. Moreover, our method of converting word forms into word meanings is better than the one used in BBK. Both CFWMS and BBK retrieve the hypernyms of word forms, and the noise created during this process could affect the accuracy of clustering. CFWMS retrieves only up to two levels of hypernyms, while BBK retrieves up to five levels, so CFWS introduces less noise into the text database than BBK.

7. Conclusion

In this paper, 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 [9,10,21,27].

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 bisecting k-means (BKM) [33] and Frequent Itemset-based Hierarchical Clustering (FIHC) [12] 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 and hyponyms/hypernyms provided by the WordNet ontology to preprocess the documents. CFWMS has a better accuracy than CFWS on most of our test data sets. Also, CFWMS outperforms the modified bisecting k-means using background knowledge (BBK) [19], which also uses an ontology to explore the lexical relationships between words.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th VLDB Conference, 1994, pp. 487–499.
[2] J. Allan, HARD track overview in TREC 2003 high accuracy retrieval from documents, in: Proceedings of the 12th Text Retrieval Conference, 2003, pp. 24–37.
[3] F. Beil, M. Ester, X. Xu, Frequent term-based text clustering, in: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 436–442.
[4] Classic data set. <ftp://ftp.cs.cornell.edu/pub/smart/>.
[5] D.R. Cutting, D.R. Karger, J.O. Pedersen, J.W. Tukey, Scatter/gather: a cluster-based approach to browsing large document collections, in: Proceedings of Annual ACM SIGIR Conference on Research and Development in Information Retrieval, 1992, pp. 318–329.
[6] B. Choudhary, P. Bhattacharyya, Text clustering using semantics, in: Proceedings of the 11th International World Wide Web Conference, 2002.
[7] A. Doucet, H. Ahonen-Myka, Non-contiguous word sequences for information retrieval, in: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-2004) Workshop on Multiword Expressions and Integrating Processing, 2004, pp. 88–95.
[8] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs, 1988.
[9] M. Farach, Optimal suffix tree construction with large alphabets, in: Proceedings of the 38th Annual Symposium on Foundation of Computer Science, 1997, pp. 137–143.

[10] M. Farach, P. Ferragina, S. Muthukrishnan, Overcoming the memory bottleneck in suffix tree construction, in: Proceedings of the 39th Symposium on Foundations of Computer Science, 1998, pp. 174–185.
[11] C. Fellbaum (Ed.), WordNet: An Electronic Lexical Database, MIT Press, 1998.
[12] B.C.M. Fung, K. Wang, M. Ester, Hierarchical document clustering using frequent itemsets, in: Proceedings of SIAM International Conference on Data Mining, 2003.
[13] B.C.M. Fung, K. Wang, M. Ester, Hierarchical document clustering, in: John Wang (Ed.), The Encyclopedia of Data Warehousing and Mining, Idea Group, 2005.
[14] R. Giegerich, S. Kurtz, From Ukkonen to McCreight and Weiner: a unifying view of linear-time suffix tree construction, Algorithmica 19 (3) (1997) 331–353.
[16] High Accuracy Retrieval from Documents (HARD) Track of Text Retrieval Conference, 2004.
[17] Frequent Itemset-based Hierarchical Clustering (FIHC) Software, <http://www.cs.sfu.edu.ca/~ddm/dmsoft/Clustering/products/fihcDistribution.zip>.
[18] J.D. Holt, S.M. Chung, Multipass algorithms for mining association rules in text databases, Knowledge and Information Systems, Springer-Verlag, 2001, 3(2); pp. 168–183.
[19] A. Hotho, S. Staab, G. Stumme, Ontologies improve text document clustering, in: Proceedings of the 3rd IEEE International Conference on Data Mining, 2003, pp. 541–544.
[20] G. Karypis, CLUTO–A Clustering Toolkit, Release 2.1.1, Department of Computer Science, University of Minnesota. <http://www.cs.umn.edu/~karypis/cluoto/>.
[21] S. Kurtz, Reducing the space requirement of suffix trees, Software: Practice and Experience 29 (13) (1999) 1149–1171.
[22] L. Kaufman, P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, 1990.
[23] G.M. Landau, U. Vishkin, Fast parallel and serial approximate string matching, Journal of Algorithms 10 (2) (1989) 157–169.
[24] B. Larsen, C. Aone, Fast and effective text mining using linear-time document clustering, in: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999, pp. 16–22.
[25] The Lemur Toolkit for Language Modeling and Information Retrieval. <http://www-2.cs.cmu.edu/~lemur/>.
[27] E.M. McCreight, A space-economical suffix tree construction algorithm, Journal of ACM 23 (2) (1976) 262–272.
[28] H. Ahonen-Myka, Finding all maximal frequent sequences in text, in: Proceedings of ICML-99 Workshop on Marchine Learning in Text Data Analysis, 1999, pp. 11–17.
[29] H. Ahonen-Myka, Discovery of frequent word sequences in text, in: Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining, 2002, pp. 16–19.
[31] C.J. van Rijsbergen, Information Retrieval, 2nd ed., Buttersworth, London, 1979.
[32] J. Sedding, D. Kazakov, WordNet-based text document clustering, in: Proceedings of COLING-2004 Workshop on Robust Methods in Analysis of Natural Language Data, 2004.
[33] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, KDD-2000 Workshop on Text Mining, 2000.
[34] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (1994) 249–260.
[35] K. Wang, C. Xu, B. Liu, Clustering transactions using large items, in: Proceedings of the 8th International Conference on Information and Knowledge Management, 1999, pp. 483–490.
[36] P. Weiner, Linear pattern matching algorithms, in: Proceedings of the 14th Annual Symposium on Foundation of Computer Science, 1973, pp. 1–11.
[37] O. Zamir, O. Etzioni, O. Madani, R.M. Karp, Fast and intuitive clustering of web documents, in: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997, pp. 287–290.
[38] O. Zamir, O. Etzioni, Web document clustering: a feasibility demonstration, in: Proceedings of Annual ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, pp. 46–54.
[39] Y. Zhao, G. Karypis, Empirical and theoretical comparisons of selected criterion functions for document clustering, Machine Learning 55 (3) (2004) 311–331.

Yanjun Li received the B.S. degree in Economics from the University of International Business and Economics, Beijing, P.R. China, in 1993, the B.S. degree in Computer Science from Franklin University, Columbus, Ohio, in 2001, the M.S. degree in Computer Science and the Ph.D. degree in Computer Science and Engineering from Wright State University, Dayton, Ohio, in 2003 and 2007, respectively. She is currently an assistant professor in the department of Computer and Information Sciences at Fordham University, Bronx, New York.
Her research interests include data mining and knowledge discovery, text mining, ontology, information retrieval, bioinformatics, and parallel and distributed computing.

Soon M. Chung received the B.S. degree in Electronic Engineering from Seoul National University, Korea, in 1979, the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology, Korea, in 1981, and the Ph.D. degree in Computer Engineering from Syracuse University, Syracuse, New York, in 1990. He is currently a professor in the department of Computer Science and Engineering at Wright State University, Dayton, Ohio. His research interests include database, data mining, Grid computing, text mining, XML, and parallel and distributed processing.

John D. Holt is a software engineer at LexisNexis, Dayton, Ohio, and is a Senior Architect for the Risk and Information Analytics Systems, which includes data mining, modeling, and full text search engines. He received the Ph.D. degree in Computer Science and Engineering at Wright State University, Dayton, Ohio, in 2003.