Hierarchical Document Clustering Using Frequent Itemsets

Benjamin C.M. Fung* Ke Wang† Martin Ester‡

Abstract

A major challenge in document clustering is the extremely high dimensionality. For example, the vocabulary for a document set can easily be thousands of words. On the other hand, each document often contains a small fraction of words in the vocabulary. These features require special handlings. Another requirement is hierarchical clustering where clustered documents can be browsed according to the increasing specificity of topics. In this paper, we propose to use the notion of frequent itemsets, which comes from association rule mining, for document clustering. The intuition of our clustering criterion is that each cluster is identified by some common words, called frequent itemsets, for the documents in the cluster. Frequent itemsets are also used to produce a hierarchical topic tree for clusters. By focusing on frequent items, the dimensionality of the document set is drastically reduced. We show that this method outperforms best existing methods in terms of both clustering accuracy and scalability.

Keywords: document clustering, text documents, frequent itemsets.

1 Introduction

Document clustering has been studied intensively because of its wide applicability in areas such as web mining, search engines, information retrieval, and topological analysis. Unlike in document classification, in document clustering no labeled documents are provided. Although standard clustering techniques such as k-means can be applied to document clustering, they usually do not satisfy the special requirements for clustering documents: high dimensionality, high volume of data, ease for browsing, and meaningful cluster labels. In addition, many existing document clustering algorithms require the user to specify the number of clusters as an input parameter and are not robust enough to handle different types of document sets in a real-world environment. For example, in some document sets the cluster size varies from few to thousands of documents. This variation tremendously reduces the clustering accuracy for some of the state-of-the art algorithms.

In this paper, we propose a novel approach, *Frequent Itemset-based Hierarchical Clustering (FIHC)*, for document clustering based on the idea of *frequent itemsets* proposed by Agrawal et. al [1]. The intuition of our clustering criterion is that there are some frequent itemsets for each cluster (topic) in the document set, and different clusters share few frequent itemsets. A frequent itemset is a set of words that occur together in some minimum fraction of documents in a cluster. Therefore, a frequent itemset describes something common to many documents in a cluster. We use frequent itemsets to construct clusters and to organize clusters into a topic hierarchy. Here are the features of our approach.

- *Reduced dimensionality.* We use only the frequent items that occur in some minimum fraction of documents in document vectors, which drastically reduces the dimensionality of the document set. Experiments show that clustering with reduced dimensionality is significantly more efficient and scalable. This decision is consistent with the study from linguistics (Longman Lancaster Corpus) that only 3000 words are required to cover 80% of the written text in English [13, 18] and the result is coherent with the Zipf's law [21] and the findings in Mladenic et al. [12] and Yang et al. [20].

- *High clustering accuracy.* Experimental results show that the proposed approach FIHC outperforms best documents clustering algorithms in terms of accuracy. It is robust even when applied to large and complicated document sets.

- *Number of clusters as an optional input parameter.* Many existing clustering algorithms require the user to specify the desired number of clusters as an input parameter. FIHC treats it only as an optional input parameter. Close to optimal clustering quality can be achieved even when this value is unknown.

- *Easy to browse with meaningful cluster description.*

*Simon Fraser University, BC, Canada, bfung@cs.sfu.ca

†Simon Fraser University, BC, Canada, wangk@cs.sfu.ca. Supported in part by a research grant from the Natural Science and Engineering Research Council of Canada and by a research grant from Networks of Centres of Excellence/Institute for Robotics and Intelligent Systems.

‡Simon Fraser University, BC, Canada, ester@cs.sfu.ca

The topic tree provides a sensible structure for browsing documents. Each cluster in the tree has a set of frequent itemsets as its description. Users may utilize them to navigate the topic tree.

The outline of this paper is as follows. Section 2 briefly discusses some well-known clustering algorithms and some common preprocessing steps. Sections 3 and 4 present our algorithm in two stages, cluster construction and tree building, with a running example. Section 5 shows the experimental results and the comparison with other algorithms. Section 6 provides an analysis on our method. We conclude the paper and outline future directions of research in section 7.

2 Related Work

Hierarchical and partitioning methods are two major categories of clustering algorithms. One popular approach in document clustering is agglomerative hierarchical clustering [5]. Algorithms in this family follow a similar template: Compute the similarity between all pairs of clusters and then merge the most similar pair. Different agglomerative algorithms may employ different similarity measuring schemes. Recently, Steinbach et al. [14] shows that UPGMA [5, 9] is the most accurate one in its category. K-means and its variants [4, 9, 10] represent the category of partitioning clustering algorithms. [14] illustrates that one of the variants, bisecting k-means, outperforms basic k-means as well as the agglomerative approach in terms of accuracy and efficiency. The bisecting k-means algorithm first selects a cluster to split. Then it utilizes basic k-means to form two sub-clusters and repeats until the desired number of clusters is reached.

Wang et al. [17] introduces a new criterion for clustering transactions using frequent itemsets. In principle, this method can also be applied to document clustering by treating a document as a transaction; however, the method does not create a hierarchy for browsing. The HFTC proposed by Beil et al. [2] attempts to address the special requirements in document clustering using the notion of frequent itemsets. HFTC greedily picks up the next frequent itemset (representing the next cluster) to minimize the overlapping of the documents that contain both the itemset and some remaining itemsets. The clustering result depends on the order of picking up itemsets, which in turn depends on the greedy heuristic used. In our approach, we do not follow a sequential order of selecting clusters. Rather, we assign documents to the best clusters with all clusters available. Experiments show that this strategy produces better clusters and is more scalable.

Similar to most document clustering algorithms, our method employs several preprocessing steps including stop words removal and stemming on the document set. Each document is represented by a vector of frequencies of remaining items within the document. As an extra preprocessing step, many document clustering algorithms would replace the actual term frequency of an item by the weighted frequency, i.e., *term frequency* × *inverse document frequency* (TF×IDF), in the document vector. The idea is that if an item is too common across different documents, then it would have little discriminating power, and vice versa [16]. Note that our algorithm applies TF×IDF after stemming; therefore, each document is actually represented by a vector of weighted frequencies. The effect of TF×IDF on our algorithm will be explained in Section 3.2.

3 Constructing Clusters

The agglomerative or partitioning methods are "document-centered" in that the pairwise similarity between documents plays a central role in constructing a cluster. Our method is "cluster-centered" in that we measure the "cohesiveness" of a cluster directly, using frequent itemsets: the documents under the same topic are expected to share more common itemsets than those under different topics. First, we introduce some definitions.

A *global frequent itemset* is a set of items that appear together in more than a minimum fraction of the whole document set. A *minimum global support*, in a percentage of all documents, can be specified for this purpose. We borrowed an algorithm presented by Agrawal et al. [1] for finding global frequent itemsets. Note that itemsets are found based on word presence, not on TF and IDF. A *global frequent item* refers to an item that belongs to some global frequent itemset. A global frequent itemset containing k items is called a *global frequent k-itemset*. The *global support* of an itemset is the percentage of documents containing the itemset. A global frequent item is *cluster frequent* in a cluster C_i if the item is contained in some minimum fraction of documents in C_i. A *minimum cluster support*, in a percentage of of the documents in C_i, can be specified for this purpose. The *cluster support* of an item in C_i is the percentage of the documents in C_i that contain the item.

Our method constructs clusters in two steps: constructing initial clusters, then making initial clusters disjoint.

3.1 Constructing Initial Clusters. For each global frequent itemset, we construct an initial cluster to contain all the documents that contain this itemset. Initial clusters are not disjoint because one document may contain several global frequent itemsets. We will remove the overlapping of clusters in Section 3.2. One property of initial clusters is that all documents in a cluster contain all the items in the global frequent itemset that defines the cluster, that is, these items are mandatory for the cluster. We use this defining global frequent itemset as the *cluster label* to identify the cluster. Cluster labels serve another purpose: their set containment relationship establishes a hierarchical structure in the tree construction stage.

3.2 Making Clusters Disjoint. This step makes clusters disjoint: for each document, we identify the "best" initial cluster and keep the document only in the best initial cluster. Suppose that $Score(C_i \leftarrow doc_j)$ measures the goodness of a cluster C_i for a document doc_j. For each doc_j, we remove doc_j from all the initial clusters C_i that contain doc_j but one for which $Score(C_i \leftarrow doc_j)$ is maximized. If there are more than one C_i that maximizes $Score(C_i \leftarrow doc_j)$, choose the one that has the most number of items in the cluster label. After this step, each document belongs to exactly one cluster. Notice that this step preserves the early property that the items in the cluster label are mandatory for a cluster.

Now, we define the score function $Score(C_i \leftarrow doc_j)$. Intuitively, a cluster C_i is "good" for a document doc_j if there are many global frequent items in doc_j that appear in "many" documents in C_i. The "many" is qualified by being cluster frequent in C_i. Precisely, the following score measures the goodness of an initial cluster C_i for a document doc_j.

(3.1)
$$Score(C_i \leftarrow doc_j) = [\sum_x n(x) * cluster\_support(x)] - [\sum_{x'} n(x') * global\_support(x')]$$

where x represents a global frequent item in doc_j and the item is also cluster frequent in C_i; x' represents a global frequent item in doc_j that is not cluster frequent in C_i; $n(x)$ and $n(x')$ are the weighted frequency of x and x' in the feature vector of doc_j. $n(x)$ and $n(x')$ are defined by the TF×IDF of item x and x', as discussed in Section 2. For better understandability, in our running example we use simply the term frequency (TF) of x and x', i.e., the number of occurrences in a document, without applying TF×IDF.

The first term of the score function rewards cluster C_i if a global frequent item x in doc_j is cluster frequent in C_i. In order to capture the importance (weight) of item x in different clusters, we multiply the frequency of x in doc_j by its cluster support in C_i. The second term of the function penalizes cluster C_i if a global frequent item x' in doc_j is not cluster frequent in C_i. The frequency of x' is multiplied by its global support which can be viewed as the importance of x' in the entire document set. This part encapsulates the concept of dissimilarity into the score.

Doc. name	**Feature vector** (flow, form, layer, patient, result, treatment)
cisi.1	(0 1 0 0 0 0)
cran.1	(1 1 1 0 0 0)
cran.2	(2 0 1 0 0 0)
cran.3	(2 1 2 0 3 0)
cran.4	(2 0 3 0 0 0)
cran.5	(1 0 2 0 0 0)
med.1	(0 0 0 8 1 2)
med.2	(0 1 0 4 3 1)
med.3	(0 0 0 3 0 2)
med.4	(0 0 0 6 3 3)
med.5	(0 1 0 4 0 0)
med.6	(0 0 0 9 1 1)

Table 1: Document set

Global frequent itemset	**Global support**
{flow}	42%
{form}	42%
{layer}	42%
{patient}	50%
{result}	42%
{treatment}	42%
{flow, layer}	42%
{patient, treatment}	42%

Table 2: Global frequent itemsets
(minimum global support = 35%)

Example: Consider the twelve documents in Table 1. They are selected from the document set in [3] and their document names indicate their topics. Each document is represented by a feature vector. Table 2 contains all the global frequent k-itemsets with their global supports. The initial clusters of this example are shown in Table 3. To find the most suitable cluster for document $med.6$, for example, we need to calculate its scores against each initial cluster that contains the document:

$$Score(C(patient) \leftarrow med.6) = 9*1 + 1*0.83 - 1*0.42 = 9.41$$

Cluster (label)	Documents in cluster	Cluster frequent items & cluster supports (CS)
C(flow)	cran.1, cran.2, cran.3, cran.4, cran.5	{flow, CS=100%}, {layer, CS=100%}
C(form)	cisi.1, cran.1, cran.3, med.2, med.5	{form, CS=100%}
C(layer)	cran.1, cran.2, cran.3, cran.4, cran.5	{layer, CS=100%}, {flow, CS=100%}
C(patient)	med.1, med.2, med.3, med.4, med.5, med.6	{patient, CS=100%}, {treatment, CS=83%}
C(result)	cran.3, med.1, med.2, med.4, med.6	{result, CS=100%}, {patient, CS=80%}, {treatment, CS=80%}
C(treatment)	med.1, med.2, med.3, med.4, med.6	{treatment, CS=100%}, {patient, CS=100%}, {result, CS=80%}
C(flow, layer)	cran.1, cran.2, cran.3, cran.4, cran.5	{flow, CS=100%}, {layer, CS=100%}
C(patient, treatment)	med.1, med.2, med.3, med.4, med.6	{patient, CS=100%}, {treatment, CS=100%}, {result, CS=80%}

Table 3: Initial clusters
(minimum cluster support = 70%)

$Score(C(treatment) \leftarrow med.6) = 10.8$
$Score(C(result) \leftarrow med.6) = 10.6$
$Score(C(patient, treatment) \leftarrow med.6) = 10.8$

We use $Score(C(patient) \leftarrow med.6)$ to explain the calculation. The global frequent items in $med.6$ are "patient", "result", and "treatment". Their frequencies in the feature vector are 9, 1, and 1 respectively. "patient" and "treatment" are cluster frequent in cluster $C(patient)$; hence these two items appear in the rewarding part of the function and their frequencies are multiplied by their corresponding cluster supports 1 and 0.83 respectively. "result" is not cluster frequent in cluster $C(patient)$; therefore, it appears in the penalty part and its frequency is multiplied by its global support 0.42.

Both clusters $C(treatment)$ and $C(patient, treatment)$ get the same highest score. Document $med.6$ is assigned to $C(patient, treatment)$, which has a larger number of items in its cluster label, i.e., a cluster with a more specific topic. After repeating the above computation for each document, Table 4 shows the disjoint clusters. Ignore the third column at this moment. ■

Cluster (label)	Documents in cluster	Cluster frequent items & cluster supports (CS)
C(flow)	cran.1, cran.2, cran.3, cran.4, cran.5	{flow, CS=100%}, {layer, CS=100%}
C(form)	cisi.1	{form, CS=100%}
C(layer)		none
C(patient)	med.5	{patient, CS=100%}, {treatment, CS=83%}
C(result)		none
C(treatment)		{treatment, CS=100%}, {patient, CS=100%}, {result, CS=80%}
C(flow, layer)		none
C(patient, treatment)	med.1, med.2, med.3, med.4, med.6	{patient, CS=100%}, {treatment, CS=100%}, {result, CS=80%}

Table 4: Disjoint clusters

We like to point out some important differences between the cluster label and the set of cluster frequent items associated with a cluster. The cluster label is a set of mandatory items in the cluster in that every document in the cluster must contain all the items in the cluster label. We use the cluster label to construct an initial cluster and to identify the cluster. On the other hand, a cluster frequent item is required to appear in some minimum fraction of documents in the cluster. We use the cluster frequent items as the topic description of the cluster.

Since some documents are removed from initial clusters, we need to recompute the cluster frequent items for each cluster to reflect the updated clustering. While re-computing the cluster frequent items of a cluster C_i, we also include all the documents from all "descendants" of C_i. A cluster is a descendant of C_i if its cluster label is a superset of the cluster label of C_i. The rationale is that descendants are likely to be subtopics of a parent; therefore, it is sensible to include them.

Example: The third column in Table 4 reflects the updated cluster frequent items in the disjoint clusters. The potential descendant of cluster $C(patient)$ is cluster $C(patient, treatment)$. While recomputing the cluster frequent items of $C(patient)$, we would consider all the documents in both $C(patient, treatment)$ and

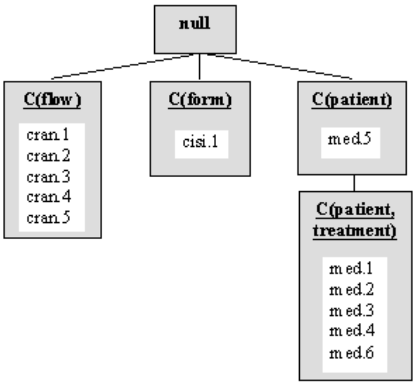

Figure 1: Cluster tree built from table 4

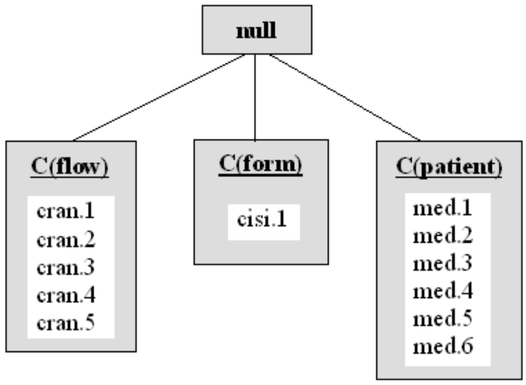

Figure 2: Cluster tree after child pruning

$C(patient)$. The cluster support of the item "treatment" in cluster $C(patient)$ is 83% because five out of the six documents contain this item. ∎

4 Building the Cluster Tree

The set of clusters produced by the previous stage can be viewed as a set of topics and subtopics in the document set. In this section, a cluster (topic) tree is constructed based on the similarity among clusters. In case a tree contains too many clusters, two pruning methods are applied to efficiently shorten and narrow a tree by merging similar clusters together.

4.1 Tree Construction. In this section, we explain how to construct a hierarchical cluster tree. The resulting tree has two objectives: to form a foundation for pruning and to provide a natural structure for browsing. In the cluster tree, each cluster (except the cluster with the empty cluster label) has exactly one parent. The topic of a parent cluster is more general than the topic of a child cluster and they are "similar" to a certain degree.

Recall that each cluster uses one global frequent k-itemset as its cluster label. Such clusters are called *k-clusters* below. In the cluster tree, the root node appears at level 0, which corresponds to the cluster with the cluster label "null", and collects the unclustered documents. In the actual user interface, the unclustered documents can be put in a cluster marked "miscellaneous" at level 1. The 1-clusters appear in level 1 of the tree, and so forth for every level. The depth of the tree is equal to the maximum size of global frequent itemsets.

We build the cluster tree bottom-up by choosing a parent at level $k-1$ for each cluster at level k. For each k-cluster C_i at level k, we first identify all potential parents that are $(k-1)$-clusters and have the cluster label being a subset of C_i's cluster label. There are at most k such potential parents. The next step is to choose the "best" among potential parents. The criterion for selecting the best is similar to choosing the best cluster for a document in Section 3.2. We first merge all the documents in the subtree of C_i into a single conceptual document $doc(C_i)$, which is done incrementally in the bottom-up tree construction, and then compute the score of $doc(C_i)$ against each potential parent. The potential parent with the highest score would become the parent of C_i. All leaf clusters that contain no document can be removed.

Example: Consider the clusters in Table 4. We start to build the tree from 2-clusters (i.e., clusters with 2-itemsets as the cluster label). Cluster $C(flow, layer)$ is removed since it is an empty leaf node. Next, we select a parent for $C(patient, treatment)$. The potential parents are $C(patient)$ and $C(treatment)$. $C(patient)$ gets a higher score and becomes the parent of $C(patient, treatment)$. Figure 1 depicts the resulting cluster tree. ∎

4.2 Tree Pruning. A cluster tree can be broad and deep, especially when a small minimum global support is used. Therefore, it is likely that documents of the same topic are distributed over several small clusters, which would lead to poor clustering accuracy. The aim of tree pruning is to merge similar clusters in order to produce a natural topic hierarchy for browsing and to increase the clustering accuracy. Before introducing the pruning methods, we will first define the *inter-cluster similarity*, which is a key notion for merging clusters.

To measure the inter-cluster similarity between two clusters C_a and C_b, we measure the similarity of C_b to C_a, and the similarity of C_a to C_b. The idea is to treat

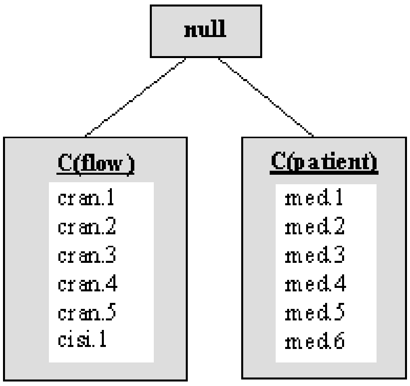

Figure 3: Cluster tree after child pruning and sibling merging

one cluster as a conceptual document (by combining all the documents in the cluster) and measure its score against the other cluster using our score function defined in Section 3.2. The only difference is that the score has to be normalized to remove the effect of varying document size. Formally, the similarity of C_j to C_i is defined as:

$$(4.2)\qquad Sim(C_i \leftarrow C_j) = \frac{Score(C_i \leftarrow doc(C_j))}{\sum_x n(x) + \sum_{x'} n(x')} + 1$$

where C_i and C_j are two clusters; $doc(C_j)$ stands for combining all the documents in the subtree of C_j into a single document; x represents a global frequent item in $doc(C_j)$ that is also cluster frequent in C_i; x' represents a global frequent item in $doc(C_j)$ that is not cluster frequent in C_i; $n(x)$ is the weighted frequency of x in the feature vector of $doc(C_j)$; $n(x')$ is the weighted frequency of x' in the feature vector of $doc(C_j)$.

To explain the normalization by $\sum_x n(x) + \sum_{x'} n(x')$, notice that the global support and cluster support in $Score$ function are always between 0 and 1. Thus, the maximum value of $Score$ is $\sum_x n(x)$ and the minimum value of $Score$ is $-\sum_{x'} n(x')$. After normalizing $Score$ by $\sum_x n(x) + \sum_{x'} n(x')$, the normalized value is within the range of [-1,1]. To avoid negative similarity values, we add the term +1. As a result, the range of Sim function is [0,2].

We define the inter-cluster similarity between C_a and C_b as the geometric mean of the two normalized scores $Sim(C_a \leftarrow C_b)$ and $Sim(C_b \leftarrow C_a)$:

(4.3)
$$Inter\_Sim(C_a \leftrightarrow C_b) = [Sim(C_a \leftarrow C_b) * Sim(C_b \leftarrow C_a)]^{\frac{1}{2}}$$

C_a and C_b are two clusters including their descendant clusters.

The advantage of the geometric mean is that two clusters are considered to be similar only if both values $Sim(C_a \leftarrow C_b)$ and $Sim(C_b \leftarrow C_a)$ are high. Given that the range of Sim function is [0,2], the range of $Inter\_Sim$ function is also [0,2]. Higher values imply higher similarity between two clusters. An $Inter\_Sim$ value below 1 implies the weight of dissimilar items has exceeded the weight of similar items. Hence, the $Inter\_Sim$ value of 1 serves as a good threshold in distinguishing whether two clusters are similar. We now present two pruning methods.

Child Pruning. The objective of child pruning is to efficiently shorten a tree by replacing child clusters with their parent. The pruning criterion is based on the inter-cluster similarity between a parent and its child. A child is pruned only if it is similar to its parent. The rationale is that if a subtopic (e.g. tennis ball) is very similar to its parent topic (e.g. tennis), the subtopic is probably too specific and can be removed.

The procedure is to scan the tree in the bottom-up order. For each non-leaf node, we calculate $Inter\_Sim$ between the node and each of its children, and prune the child cluster if $Inter\_Sim$ is above 1. If a cluster is pruned, its children become the children of their grandparent. Notice that child pruning is only applicable to level 2 and below since the root (at level 0) collects only unclustered documents.

Example: Consider figure 1. To determine whether cluster $C(patient, treatment)$ should be pruned, the inter-cluster similarity between $C(patient)$ and $C(patient, treatment)$ is calculated as follows:

$$Sim(C(patient) \leftarrow C(patient, treatment)) = (30*1+9*0.83-1*0.42-8*0.42)/48+1 = 1.70$$
$$Sim(C(patient, treatment) \leftarrow C(patient)) = (34*1+8*0.8+9*1-2*0.42)/53+1 = 1.92$$
$$Inter\_Sim(C(patient) \leftarrow C(patient, treatment)) = (1.70*1.92)^{\frac{1}{2}} = 1.81$$

To calculate $Sim(C(patient) \leftarrow C(patient, treatment))$, we combine all the documents in cluster $C(patient, treatment)$ by adding up their feature vectors. The summed feature vector is (0, 1, 0, 30, 8, 9). Then we calculate the score of this combined document against $C(patient)$ and normalize the score by the sum of the frequencies which is 48. $Sim(C(patient, treatment) \leftarrow C(patient))$ is computed using the same method. Since the inter-cluster similarity is above 1, cluster $C(parent, treatment)$ is pruned. See Figure 2. ■

Cluster pair (C_i, C_j)	**Sim** $(C_j \leftarrow C_i)$	**Sim** $(C_i \leftarrow C_j)$	**Inter_Sim** $(C_i \leftrightarrow C_j)$
C(flow) & C(form)	0.71	0.58	0.64
C(flow) & C(patient)	0.58	0.54	0.56
C(form) & C(patient)	0.58	0.58	0.58

Table 5: Inter-cluster similarity calculation

Data Set	**# of Docs**	**# of Classes**	**Class Size**	**# of Terms**
Classic4	7094	4	1033 – 3203	12009
Hitech	2301	6	116 – 603	13170
Re0	1504	13	11 – 608	2886
Reuters	8649	65	1 – 3725	16641
Wap	1560	20	5 – 341	8460

Table 6: Summary descriptions of data sets

Sibling Merging. Since child pruning only applies to level 2 and below, there are often many clusters at level 1. Sibling merging will merge similar clusters at level 1. Each time, we calculate the $Inter\_Sim$ for each pair of clusters at level 1 and merge the cluster pair that has the highest $Inter\_Sim$. By merging two clusters, the children of the two clusters become the children of the merged cluster. This procedure is repeated for the remaining clusters at level 1 until the user-specified number of clusters is reached. If the user does not specify the number of cluster, the algorithm will terminate when all cluster pairs at level 1 have $Inter\_Sim$ below or equal to 1. The pairwise comparison ensures that only similar clusters are merged.

The above sibling merging is similar to the pairwise merging in the agglomerative clustering method. The difference is that the agglomerative method applies the merging to *all* clusters, which often becomes the bottleneck to scalability. In our algorithm, the merging is applied to only the clusters at level 1, which are limited by the number of global frequent items. For clusters at a lower level, this is not necessary because child pruning has merged similar children into their parent. The idea is to apply the child pruning that does not require the expensive pairwise search when the number of clusters is large (at lower levels), and to apply the elaborated sibling merging only when the number of clusters is small (at level 1). This approach is more scalable than the agglomerative clustering.

Example: Consider the tree in Figure 2. Sibling merging computes the $Inter\_Sim$ for each pair of clusters at the level 1 as in Table 5. If the user has not specified the desired number of clusters, FIHC would terminate and return the tree as in Figure 2. Suppose the user specifies the number of clusters as 2. The algorithm would prune one cluster at level 1 based on the inter-cluster similarity among clusters $C(flow)$, $C(form)$, and $C(patient)$. Since $C(flow)$ and $C(form)$ is the pair with the highest $Inter\_Sim$, the smaller cluster $C(form)$ would merge with the larger cluster $C(flow)$. Figure 3 depicts the resulting tree. ∎

5 Experimental Evaluation

This section presents the experimental evaluation of the proposed method (FIHC) by comparing its results with several popular document clustering algorithms, agglomerative UPGMA [5, 9], bisecting k-means [5, 9, 14], and HFTC [2]. We make use of the CLUTO-2.0 Clustering Toolkit [8] to generate the results of UPGMA and bisecting k-means. For HFTC, we obtained the original Java program from the author and then compiled the program into Windows native code to avoid the overhead of the Java Virtual Machine. All algorithms, except HFTC, employ TF×IDF as a preprocessing step. HFTC applies its own preprocessing technique, the term frequency variance selection. The produced clustering results are evaluated by the same method and criterion to ensure fair comparison across all algorithms.

5.1 Data Sets.

Five data sets widely used in document clustering research [14, 2] were used for our evaluation. They are heterogeneous in terms of document size, cluster size, number of classes, and document distribution. Each document has been pre-classified into a single topic, called *natural class* below. This information is utilized by the evaluation method for measuring the accuracy of the clustering result. During the cluster construction, this information is hidden from all clustering algorithms.

Here are the five data sets. *Classic4* is combined from the four classes CACM, CISI, CRAN, and MED abstracts [3]. *Hitech* and *Wap* are originally from the San Jose Mercury newspaper articles [15] and the Yahoo! subject hierarchy web pages [19], respectively. *Reuters* and *Re0* were extracted from newspaper articles [11]. For *Reuters*, we only use the articles that are uniquely assigned to exactly one topic. All of these data sets, except *Reuters*, can be obtained from [8].

5.2 Evaluation Method. A commonly used external measurement, the *F-measure* [10, 14], is employed to evaluate the accuracy of the produced clustering solutions. It is a standard evaluation method for both flat and hierarchical clustering structures. Suppose that each cluster is treated as if it were the result of a query and each natural class is treated as if it were the relevant set of documents for a query. The *recall*, *precision*, and *F-measure* for natural class K_i and cluster C_j are calculated as follows:

$$Recall(K_i, C_j) = \frac{n_{ij}}{|K_i|} \tag{5.4}$$

$$Precision(K_i, C_j) = \frac{n_{ij}}{|C_j|} \tag{5.5}$$

(5.6)
$$F(K_i, C_j) = \frac{2 * Recall(K_i, C_j) * Precision(K_i, C_j)}{Recall(K_i, C_j) + Precision(K_i, C_j)}$$

where n_{ij} is the number of members of natural class K_i in cluster C_j. Intuitively, $F(K_i, C_j)$ measures the quality of cluster C_j in describing the natural class K_i, by the harmonic mean of *Recall* and *Precision* for the "query results" C_j with respect to the "relevant documents" K_i. While computing $F(K_i, C_j)$ in a hierarchical structure, all the documents in the subtree of C_j are considered as the documents in C_j.

The success of capturing a natural class K_i is measured by using the "best" cluster C_j for K_i, i.e., C_j maximizes $F(K_i, C_j)$. We measure the quality of a clustering result C using the weighted sum of such maximum F-measures for all natural classes. This measure is called the *overall F-measure* of C, denoted $F(C)$:

$$F(C) = \sum_{K_i \in K} \frac{|K_i|}{|D|} max_{C_j \in C}\{F(K_i, C_j)\} \tag{5.7}$$

where K denotes all natural classes; C denotes all clusters at all levels; $|K_i|$ denotes the number of documents in natural class K_i; and $|D|$ denotes the total number of documents in the data set. The range of $F(C)$ is [0,1]. A larger $F(C)$ value indicates a higher accuracy of clustering.

5.3 Experimental Results. We evaluated our algorithm, FIHC, and its competitors in terms of F-measure, sensitivity to parameters, efficiency and scalability. Recent research in [14] shows that UPGMA and bisecting k-means are the most accurate clustering algorithms in their categories. We also compared FIHC with another frequent itemset-based algorithm, HFTC [2].

Data Set	# of Clusters	Overall F-measure			
		FIHC	UPGMA	Bi kmeans	HFTC
Classic4 (4)	3	0.62*	×	0.59	n/a
	15	0.52*	×	0.46	n/a
	30	0.52*	×	0.43	n/a
	60	0.51*	×	0.27	n/a
	Avg.	0.54	×	0.44	0.61*
Hitech (6)	3	0.45	0.33	0.54*	n/a
	15	0.42	0.33	0.44*	n/a
	30	0.41	0.47*	0.29	n/a
	60	0.41*	0.40	0.21	n/a
	Avg.	0.42*	0.38	0.37	0.37
Re0 (13)	3	0.53*	0.36	0.34	n/a
	15	0.45	0.47*	0.38	n/a
	30	0.43*	0.42	0.38	n/a
	60	0.38*	0.34	0.28	n/a
	Avg.	0.45*	0.40	0.34	0.43
Reuters (65)	3	0.58*	×	0.48	n/a
	15	0.61*	×	0.42	n/a
	30	0.61*	×	0.35	n/a
	60	0.60*	×	0.30	n/a
	Avg.	0.60*	×	0.39	0.49
Wap (20)	3	0.40*	0.39	0.40*	n/a
	15	0.56	0.49	0.57*	n/a
	30	0.57	0.58*	0.44	n/a
	60	0.55	0.59*	0.37	n/a
	Avg.	0.52*	0.51	0.45	0.35

Table 7: F-measure comparison of our FIHC method and the other four methods on five data sets
× = not scalable to run * = best competitor

Accuracy. Table 7 shows the F-measure values for all four algorithms with different user-specified numbers of clusters. Since HFTC does not take the number of clusters as an input parameter, we use the same minimum support from 3% to 6% for both HFTC and our algorithm in each data set to ensure fair comparison. Our algorithm FIHC apparently outperforms all other algorithms in accuracy for most number of clusters. Although UPGMA and bisecting k-means perform slightly better than FIHC in several cases, we argue that the exact number of clusters in a document set is usually unknown in real world clustering problem, and FIHC is robust enough to produce consistently high quality clusters for a wide range number of clusters. This fact is reflected by taking the average of the F-measure values over the different numbers of clusters. Due to the pairwise

Classic	
CRANFIELD **class**	**Best FIHC cluster**
aerodynamic, aircraft, **angle**, **boundary**, **effect**, **flow**, ft, height, **layer**, maximum, measurement, **number**, **present**, **pressure**, shape, **speed**, system, stream, **theory**, value	**angle**, approximate, body, **boundary**, calculate, condition, distribution, **effect**, equation, experiment, **flow**, investigation, **layer**, machine, method, **number**, **present**, **pressure**, **speed**, surface, **theory**, velocity

Table 8: Comparison on class/cluster frequent items (minimum cluster support = 35%)

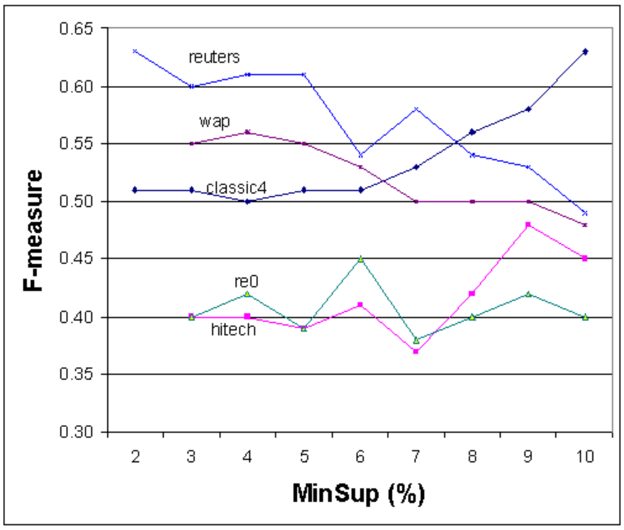

Figure 4: Sensitivity to MinSup without pre-specifying # of clusters on five data sets

similarity comparison in agglomerative algorithms, UPGMA is not scalable for large data sets. Hence, some experiment results could not be generated for UPGMA.

While the F-measure provides a yardstick of clustering accuracy in the "document retrieval" interpretation, we like to provide some intuition that our clusters capture the natural classes. One possibility is to compare the frequent items from a natural class K_i with the description, i.e., cluster frequent items, of the best cluster C_j (i.e., C_j maximizes $F(K_i, C_j)$ for K_i). We use the *CRANFIELD* class from *Classic4* [3] to illustrate this comparison. The *CRANFIELD* documents are summaries for papers on the aeronautical system. Table 8 shows the two sets of frequent items. The items in the left column are generated from the *CRANFIELD* class. The items in the right column are generated from the best cluster. We observed that many items (in bold face) in the two columns are overlapping. This suggests that the description of the cluster reflects the topic of the natural class. Interestingly, the description also capture some keywords that are not shown in the natural class column but are definitely reasonable to appear under this topic. For example, the items "body", "machine", "surface", and "velocity" are related to aeronautical system. Our algorithm also misses some items, such as "aerodynamic" and "aircraft". About 50% of the frequent items in the *CRANFIELD* class are captured by the best cluster as the description. There is a similar capturing rate for other three natural classes in *Classic4*.

Sensitivity to Parameters. Our algorithm, FIHC, allows for two input parameters: KClusters and MinSup.

1. KClusters is the number of clusters at level 1 of the tree and is an optional input. Table 7 has already shown that FIHC produces consistently high quality clusters for a wide range of KClusters.

2. MinSup is the minimum support for global frequent itemset generation with a default value 5%. Figure 4 depicts the F-measure values of FIHC with respect to MinSup without pre-specifying a value for KClusters. Unlike many other clustering methods where the clustering accuracy is very sensitive to input parameters, we observe that high clustering accuracy is fairly consistent while MinSup is set between 3% and 9%. A general guidance from numerous experiments is: if a data set contains less than 5000 documents, MinSup should be set between 5% and 9%; otherwise, MinSup should be set between 3% and 5%. We would like to emphasize that, in practice, the end user does not need to specify MinSup. Determining MinSup can be made a part of the clustering algorithm. For example, preliminary runs can be done on a sample of the document set to determine an appropriate MinSup, and then the whole document set is clustered using the determined MinSup.

Another parameter is the minimum cluster support, which determines if an item is cluster frequent. Experiments show that a value around 25% always yields a good result in different document sets, provided that there are at least a few hundreds of documents. Thus, this is not an input parameter.

Efficiency and Scalability. The largest data set,

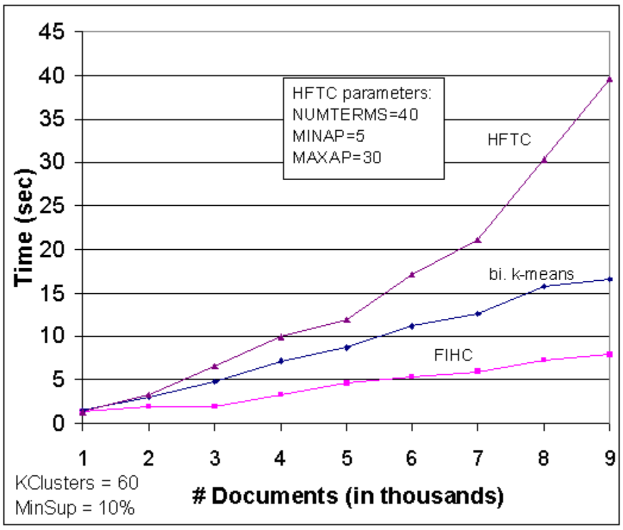

Figure 5: Comparison on efficiency with the *Reuters* document set

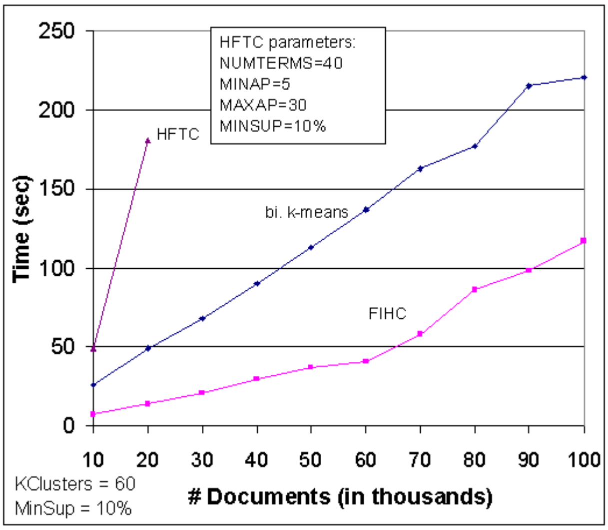

Figure 6: Comparison on efficiency with the scale-up *Reuters* document set

Reuters, is chosen to examine the efficiency and scalability of our algorithm on a Pentium III 667 MHz PC. Figure 5 compares the runtime of our algorithm only with bisecting k-means and HFTC. UPGMA is excluded again because it is not scalable. The MinSup of HFTC and our algorithm is set to 10% to ensure that the accuracy of all produced clustering solutions is approximately the same. The efficiency of HFTC is comparable with other algorithms in the first 5000 documents, but its runtime grows rapidly while there are 6000 or more documents. Our algorithm FIHC runs twice faster than the best competitor, bisecting k-means. We conclude that FIHC is significantly more efficient than other algorithms.

To create a larger data set for examining the scalability of FIHC, we duplicated the files in *Reuters* until we get 100,000 documents. Figure 6 once again illustrates that our algorithm runs approximately twice faster than bisecting k-means in this scaled up document set. Figure 7 depicts the runtime with respect to the number of documents. The whole process completes within two minutes while UPGMA and HFTC cannot even produce a clustering solution. It demonstrates that FIHC is a very scalable method. The figure also shows that Apriori and the clustering are the most time-consuming stages in FIHC, while the runtime of tree building and pruning usually completes in a split of a second. The efficiency of Apriori is very sensitive to the input parameter MinSup. Consequently, the runtime of FIHC increases while the MinSup decreases. Nevertheless, many scalable and efficient frequent itemset mining algorithms have been proposed [6, 7] and can be employed to further improve the efficiency of our method. In the clustering stage, most time is spent on constructing initial clusters, and its runtime is linear with respect to the number of documents.

6 Discussions

6.1 Tree Structure vs Browsing. Most existing agglomerative and divisive hierarchical clustering methods, e.g., bisecting k-means, generate relatively deep hierarchies. However, deep hierarchy may not be suitable for browsing. Suppose that a user makes an incorrect selection while navigating the hierarchy. She may not notice her mistake until she browses into the deeper portion of the hierarchy. Our hierarchy is relatively flat. A flat hierarchy reduces the number of navigation steps which in turn decreases the chance for a user to make mistakes. However, if a hierarchy is too flat, a parent topic may contain too many subtopics and it would increase the time and difficulty for the user to locate her target. Thus, a balance between depth and width of the tree is essential for browsing. Given a reasonable MinSup from 3% to 9%, our cluster tree usually has two to four levels in our experimental results. In general, the number of levels depends on the given documents.

Another frequent itemset-based method, HFTC, also provides a relatively flat hierarchy and its lattice structure is suitable for browsing. The resulting hierarchy usually contains many clusters at the first level. As a result, documents in the same natural class are likely to be distributed into different branches of the hierar-

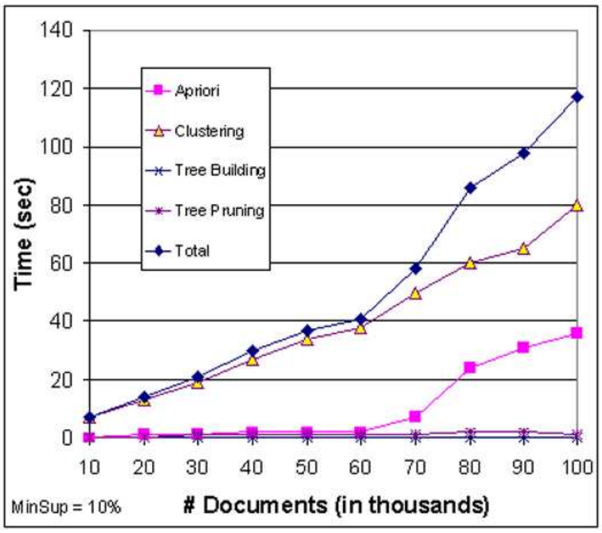

Figure 7: Scalability of our FIHC method with the scale-up *Reuters* document set

chy which decreases the overall clustering accuracy. Our sibling merging method resolves this problem by joining similar clusters at the first level of the tree.

6.2 Complexity Analysis. Our method involves four phases: finding global frequent itemsets, initial clustering, tree construction, and pruning. The problem of finding frequent itemsets has been studied intensively in the data mining literature. In the initial clustering phase, the document feature vectors are scanned twice, once for constructing initial clusters and once for making clusters disjoint. Since an initial cluster labeled by a global frequent itemset f contains $global\_support(f)$ documents, this step makes $\sum_{f \in F} global\_support(f)$ document-to-cluster assignments and score calculations. This amount of work is no more than the support counting in mining global frequent itemsets. In the tree construction, all empty clusters with a maximal cluster label are first removed. The remaining number of clusters is no more than, often much smaller than, the number of documents.

The tree construction is essentially linear in the number of remaining clusters because finding a parent for a k-cluster requires to examine at most k clusters at level $k-1$ (which are subsets of the k-cluster), where k (i.e., the height of the topic hierarchy) is usually very small. Again, this part is no more expensive than the Apriori subset pruning in [1]. Child pruning makes only one scan of clusters, and sibling merging is performed only at the first level of the tree. As mentioned in Section 4.2, the philosophy is to apply the efficient child pruning when the number of clusters is large and to apply the elaborated sibling merging only when the number of clusters is small. This approach is more scalable than the agglomerative clustering.

7 Conclusions

Most traditional clustering methods do not satisfy the special requirements for document clustering, such as high dimensionality, high volume, and ease of browsing with meaningful cluster labels. In this paper, we present a new approach to address these issues. The novelty of this approach is that it exploits frequent itemsets for defining a cluster, organizing the cluster hierarchy, and reducing the dimensionality of document sets. The experimental results show that our approach outperforms its competitors in terms of accuracy, efficiency, and scalability.

Acknowledgment

The initial phase of this work benefited considerably from extensive discussions with Leo Chen and Linda Wu.

References

[1] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12-15 1994.

[2] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD)'2002*, Edmonton, Alberta, Canada, 2002. http://www.cs.sfu.ca/˜ ester/publications.html.

[3] Classic. ftp://ftp.cs.cornell.edu/pub/smart/.

[4] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[5] R. C. Dubes and A. K. Jain. *Algorithms for Clustering Data*. Prentice Hall College Div, Englewood Cliffs, NJ, March 1998.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*, Dallas, Texas, USA, May 2000.

[7] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *ACM SIGKDD Explorations*, 2(1):58–64, July 2000.

[8] G. Karypis. Cluto 2.0 clustering toolkit, April 2002. http://www-users.cs.umn.edu/˜ karypis/cluto/.

[9] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, March 1990.

[10] B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. *KDD'99*, pages 16–22, 1999.

[11] D. D. Lewis. Reuters. http://www.research.att.com/˜ lewis/.

[12] D. Mladenic and M. Grobelnik. Feature selection for unbalanced class distribution and naive bayes. In *International Conference on Machine Learning*, 1999.

[13] I. S. P. Nation and J. Coady. Vocabulary and reading. In *Carter and McCarthy (Eds.) Vocabulary and Language Teaching*. Longman, London, 1988.

[14] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. *KDD Workshop on Text Mining'00*, 2000.

[15] Text REtrival Conference TIPSTER, 1999. http://trec.nist.gov/.

[16] C. J. van Rijsbergen. *Information Retrieval*. Dept. of Computer Science, University of Glasgow, Butterworth, London, 2 edition, 1979.

[17] K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *Proc. CIKM'99*, pages 483–490, 1999.

[18] R. Waring. Second language vocabulary acquisition, linguistic context and vocabulary task design. *Summary of a paper presented at The British Council Conference in St Andrews Scotland*, September 1995.

[19] Yahoo! http://www.yahoo.com/.

[20] M. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning*, 1997.

[21] G. K. Zipf. Selective studies and the principle of relative frequency in language, 1932.