Clustering Transactions Using Large Items

Ke Wang Chu Xu Bing Liu
School of Computing
National University of Singapore
{wangk,xuchu,liub}@comp.nus.edu.sg

Abstract

In traditional data clustering, similarity of a cluster of objects is measured by pairwise similarity of objects in that cluster. We argue that such measures are not appropriate for transactions that are sets of items. We propose the notion of *large items*, i.e., items contained in some minimum fraction of transactions in a cluster, to measure the similarity of a cluster of transactions. The intuition of our clustering criterion is that there should be many large items within a cluster and little overlapping of such items across clusters. We discuss the rationale behind our approach and its implication on providing a better solution to the clustering problem. We present a clustering algorithm based on the new clustering criterion and evaluate its effectiveness.

1 Introduction

In this paper, the term "transaction" refers to a set of items in general. An example of a transaction is the set of terms in an article, a basket of items purchased during a shopping, a customer profile of interests, a set of symptoms of a patient, a set of features of an image, etc. *Transaction clustering* refers to partitioning a collection of transactions into clusters such that similar transactions are in the same cluster and dissimilar transactions are in different clusters. Due to the recent developments in IR, Web technologies, and data mining, transaction clustering plays an important role and receives new attentions in many applications and fields that go beyond just accelerating near-neighbor search. For example, [3] demonstrated transaction clustering as an effective browsing method in its own right; [2] identified transaction clustering as an essential technique to many web applications; transaction clustering provides solutions to targeted marketing/advertising, discovering the causes of diseases, content-based image retrieval, recommending links to web users, organizing folders and bookmarks, constructing information hierarchies like Yahoo! and Infoseek, etc.

Most clustering approaches adopt a pairwise similarity (e.g., cosine measure, the Dice and Jaccard coefficient, etc) for measuring the "distance" of two transactions. See [13] for example. We argue that, for transactions that are made of sparsely distributed items, pairwise similarity is neither necessary nor sufficient for judging whether a cluster of transactions are similar. Consider transactions $t_1, \ldots, t_k$ such that every pair (t_i, t_j) shares many items in common. In any pairwise similarity measure, $\{t_1, \ldots, t_k\}$ are considered as a cluster of similar transactions. However, it may well be the case that the items shared by each pair have no overlapping with the items shared by any other pair. For a collection of documents, this means that no term is contained in more than two documents, thus, no central topic for the collection. The next example shows that pairwise similarity is not necessary either for a cluster of transactions to be similar.

Example 1.1 *Consider a cluster of five transactions*

$$t_1 = \{a, e, h, k\},\ t_2 = \{a, c, f\},\ t_3 = \{a, b, c\},$$
$$t_4 = \{b, c, i, j\},\ t_5 = \{b, e, g\}.$$

Each transaction t_i represents a set of favorite combat movies of a person. Out of the 10 pairs of transactions, only two pairs, i.e., (t_2, t_3) and (t_3, t_4), share two movies in common; all other pairs share no more than one movie in common. Therefore, pairwise similarity in this cluster is weak. However, we can see that a, b, c are "typical" combat movies in that each of them is liked by 60% (i.e., 3) cluster members. These "typical" movies can be used to characterize the interest of this cluster of people, as opposed to, say, "typical" movies that are used to characterize the interest of a cluster of people interested in romantic movies.

In general, given a large number of combat movies and a large cluster of people who like combat movies, it is unlikely that every person shares many combat movies with every other person in that cluster. What is likely is that there are some (or many) "typical" movies that are liked by at least some fraction of people in the cluster (i.e., 60% in Example 1.1). To our knowledge, this notion of similarity that directly addresses a cluster of transactions without relying on pairwise similarity has not been studied before. Clustering transactions using such similarity is the focus of this paper.

We propose a similarity measure for a *cluster* of transactions based on the notion of large items. An item is *large* in a cluster of transactions if it is contained in a user-specified fraction of transactions in that cluster. This user-specified fraction is called the *minimum support*. We use large items to measure the similarity of the cluster. Our criterion of a good clustering is that there are many large items within a cluster and there is little overlapping of such items across clusters. Here are several features of this approach.

- *Incorporating user's expectation.* Often the user has some expectation on the similarity for a cluster. For example, a cluster of Sports product buyers is expected to have less similarity than a cluster of Golf product buyers because there are far more Sports products than Golf products. Indeed, a Sports product purchased by 10% buyers or more in the first cluster may be considered as "typical", whereas a Golf product purchased by 20% buyers may not in the second cluster. In our framework, the user can specify his/her notion of large items through the minimum support.

- *Discriminating features and noises.* An often heard criticism against clustering approaches is that they do not discriminate attributes. Take the congressional voting as an example (see Section 6). Republicans and democrats often vote similarly on many issues and such issues are thus noises for separating the voting behaviours of the two parties. In our approach, the user can specify a large minimum support to exclude such issues from consideration.

- *Polythetic vs monothetic clusterings.* By specifying a minimum support less than 100%, the clustering is *polythetic* where a transaction may have some items in common with other transactions in that cluster but where there are no specific items required for clustering membership. This is more useful than the *monothetic* clustering produced by the minimum support of 100% where all of the transactions in a given cluster must contain certain items. An example of monothetic clustering approach is [15].

- *Dynamic clustering without using ad hoc parameters.* Several dynamic clusterings that we know of use threshold values to determine the destination of a new transaction, , e.g., [16] using thresholds on dissimilarity and [5, 14] using thresholds on data density/distribution. The whole family of partitional clustering methods require the number of clusters (such as k-means and k-medoids [10]) as an input parameter. Instead of using threshold values, our clustering criterion penalizes a wrong decision in the search of a good clustering. This feature makes our approach more robust and adaptive in a dynamic environment.

- *Scalable clustering.* Our algorithm makes only a few (typically no more than 3) scans of transactions, thus, highly efficient for disk-resident data sets where the I/O cost becomes the bottleneck of efficiency.

2 Related Work

Data clustering has been extensively studied in Statistics, Machine Learning, and Information Retrieval. See [4, 9, 11, 13] for a list. Most methods rely on the notion of pairwise similarity, i.e., a distance measure, between a pair of objects (such as L_p metric, the cosine measure, and the Dice and Jaccard coefficient). These methods suffer from the anomalies for transactions discussed in Introduction. The notion of *document frequency* (i.e., the fraction of documents that contain a term) widely used in text document classification and clustering [11] is related to our notion of large items. However, there are important differences. Unlike document frequency, large items are defined with respect to a cluster of transactions, not the whole collection of transactions. Therefore, during the construction of clusters, large items must be maintained as clusters grow, which is one of the performance issues we address in this paper. Existing work, however, do not address the efficient update of document frequency as documents are added or deleted, therefore, can deal with only a fixed set of documents.

Our method is similar to k-means algorithms in that it scans transactions and assigns the next transaction to the "best" cluster. However, two important differences exist. First, we do not require the number k of clusters. This is very important as the user usually does not know this number in advance or the number may change as the transaction population changes. Second, we choose a cluster for the next transaction not based on the "distance" between the cluster and the transaction, i.e., a local goodness, but based on the global goodness of clustering. In fact, this is a consequence of the first difference because the local goodness implies creating a new cluster for every transaction.

[2] defines clusters as maximal connected components of some pairwise similarity of transactions, thus, suffers from the breakdown of the transitivity of pairwise similarity (e.g., $A \cap B \neq \emptyset$ and $B \cap C \neq \emptyset$ does not imply $A \cap C \neq \emptyset$). [8] adopts hypergraph partitioning for transaction clustering, but the result is a clustering of items (not transactions) that occur together in the transactions. [7] noted some limitations of traditional similarity measures for transactions and proposed the common "neighbors" of two transactions as a measure of pairwise similarity. Our method does not use any notion of pairwise similarity. [16] proposes an I/O-efficient clustering algorithm based on incremental update of some key information maintained for each cluster. [14, 6, 5] use the distribution and density-connectivity of data points to guide the construction of clusters. All these methods make certain assumptions on the clustering structures, i.e., maximum dissimilarity in a cluster [16], uniform distribution of data points in a cluster [14], and maximum size of neighborhood and minimum density [5].

The term "large item" comes from the work on mining association rules [1]. While [1] groups the items that occur in similar transactions, we cluster transactions that contain similar items. The difference is that clustering emphasizes the dissimilarity of clusters.

3 A New Clustering Criterion

In this section, we define our clustering criterion based on the notion of large items. Consider a collection of transactions $\{t_1, \ldots, t_n\}$, where each transaction t_i is a set of items $\{i_1, \ldots, i_p\}$. A *clustering* $\mathcal{C}$ is a partition $\{C_1, \ldots, C_k\}$ of $\{t_1, \ldots, t_n\}$. Each C_i is called a *cluster*.

3.1 The large item based approach

We propose to use large items as the similarity measure of a cluster of transactions. The *support* of an item in cluster C_i is the number of transactions in C_i that contain the item. Let $|S|$ denote the number of elements in set S. For a user-specified *minimum support* θ ($0 < \theta \leq 1$), an item is *large* in cluster C_i if its support in C_i is at least $\theta * |C_i|$; otherwise, the item is *small* in C_i. Intuitively, large items are popular items in a cluster (as per the user-specified minimum support), thus, contribute to similarity in a cluster, whereas small items contribute to dissimilarity in a cluster. Let $Large_i$ denote the set of large items in C_i, and $Small_i$ denote the set of small items in C_i. Consider a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$. The cost of $\mathcal{C}$ to be minimized has two components: the intra-cluster cost and the inter-cluster cost.

The intra-cluster cost. This component is charged for intra-cluster dissimilarity, measured by the total number of small items:

$$Intra(\mathcal{C}) = |\cup_{i=1}^{k} Small_i|. \qquad (1)$$

This component will restrain creating loosely bound clusters that have too many small items. Note that we didn't use $\Sigma_{i=1}^{k}|Small_i|$. Our experiments show that $\Sigma_{i=1}^{k}|Small_i|$ tends to put all transactions into a single or few clusters even though they are not similar. To see this, consider two clusters that are not similar but share some small items. Merging these two clusters will reduce $\Sigma_{i=1}^{k}|Small_i|$ because each small item previously counted twice is now counted only once and because large items remain large after the merging. But this merging is intuitively incorrect because sharing of small items should not be considered as similarity.

The inter-cluster cost. This component is charged for inter-cluster similarity. Since large items contribute to similarity in a cluster, each cluster should have as little overlapping of large items as possible. This overlapping is defined by

$$Inter(\mathcal{C}) = \Sigma_{i=1}^{k}|Large_i| - |\cup_{i=1}^{k} Large_i|. \qquad (2)$$

In words, $Inter(\mathcal{C})$ measures the duplication of large items in different clusters. This component will restrain creating similar clusters.

To put the two together, one can specify weights for their relative importance. The *criterion function* of the clustering $\mathcal{C}$ then is defined as

$$Cost(\mathcal{C}) = w * Intra(\mathcal{C}) + Inter(\mathcal{C}). \qquad (3)$$

A weight $w > 1$ gives more emphasis to the intra-cluster similarity, and a weight $w < 1$ gives more emphasis to the inter-cluster dissimilarity. By default, $w = 1$. We now state a formal definition of the transaction clustering problem.

Definition 3.1 (Transaction clustering) *Given a collection of transactions and a minimum support, find a clustering $\mathcal{C}$ such that $Cost(\mathcal{C})$ is minimum.*

Finding an exact solution is infeasible due to the large number of ways to partition transactions. In practical applications, it suffices to find an approximate solution. This will be the topic in Sections 4 and 5. At this point, several properties of our approach are worth mentioning. First, the definition does not require a number of clusters as an input parameter and the cost is minimized over all numbers of clusters. An useful variation is to impose some maximum number of clusters. Second, we do not use "hard" threshold values to stop grouping dissimilar transactions together; instead, we penalize a wrong decision in the search of a good clustering. These features are crucial for dynamic clustering where the number of clusters and similarity in a cluster may change as new transactions are added. Third, the notion of large items is not another notion of centroid of a cluster. In fact, we do not measure any "closeness" of a transaction to a cluster, like in k-means algorithms. Instead, we use large items to evaluate the quality of the whole clustering.

Example 3.1 *Consider 6 transactions:*

$$t_1 = \{a, b, c\},\ \ t_2 = \{a, b, c, d\}, t_3 = \{a, b, c, e\}$$
$$t_4 = \{a, b, f\},\ \ t_5 = \{d, g, h\}, t_6 = \{d, g, i\}.$$

*Assume that the user-specified minimum support is 60%. A large item must be contained in at least 4 transactions (i.e., 6*60%). Consider the clustering* $\mathcal{C}_1 = \{C_1 = \{t_1, t_2, t_3, t_4, t_5, t_6\}\}$. *We have* $Large_1 = \{a, b\}$, $Small_1 = \{c, d, e, f, g, h, i\}$, $Intra(\mathcal{C}_1) = 7$, *and* $Inter(\mathcal{C}_1) = 0$. *So* $Cost(\mathcal{C}_1) = 7$.

Consider the clustering $\mathcal{C}_2 = \{C_1 = \{t_1, t_2, t_3, t_4\}, C_2 = \{t_5, t_6\}\}$. *For* C_1, *a large item should be contained in at least 3 transactions in* C_1. *We have* $Large_1 = \{a, b, c\}$ *and* $Small_1 = \{d, e, f\}$. *Similarly,* $Large_2 = \{d, g\}$ *and* $Small_2 = \{h, i\}$. *Therefore,* $Intra(\mathcal{C}_2) = 5$, $Inter(\mathcal{C}_2) = 0$, *and* $Cost(\mathcal{C}_2) = 5$. *Thus,* $\mathcal{C}_2$ *improves* $\mathcal{C}_1$ *by reducing intra-cluster dissimilarity without increasing inter-cluster similarity.*

Consider the clustering $\mathcal{C}_3 = \{C_1 = \{t_1, t_2\}, C_2 = \{t_3, t_4\}, C_3 = \{t_5, t_6\}\}$. *We have* $Large_1 = \{a, b, c\}$, $Small_1 = \{d\}$, $Large_2 = \{a, b\}$, $Small_2 = \{c, e, f\}$, $Large_3 = \{d, g\}$, $Small_3 = \{h, i\}$, $Intra(\mathcal{C}_3) = 6$, *and* $Inter(\mathcal{C}_3) = 2$. *So* $Cost(\mathcal{C}_3) = 8$. *Compared to* $\mathcal{C}_2$, *splitting* $\{t_1, t_2, t_3, t_4\}$ *into* C_1 *and* C_2 *creates more inter-cluster similarity.*

3.2 The minimum support

In choosing the minimum support the user should consider how frequently items are expected to occur in the transactions of a cluster in order for them to characterize the cluster. Besides the user's expectation, hierarchical clustering can also help address the choice of the minimum support. Initially, a small minimum support is used to produce a small number of clusters where only very dissimilar transactions go to different clusters. For each cluster, a larger minimum support is used to "fine-cluster" transactions. This is repeated recursively until the cost of a cluster cannot be reduced by further fine-clustering.

For a chosen minimum support, $Cost(\mathcal{C})$ reflects the user's satisfaction level with respect to his/her expectation as specified by the minimum support. Consequently, it does not make sense to compare $Cost(\mathcal{C})$ and clustering results produced by different minimum supports. In fact, with the minimum support of $1/n$, where n is the number of transactions, grouping all transactions into a single cluster will give $Cost(\mathcal{C})$=0. Obviously, this should not be interpreted as saying that the single cluster is always the best solution. What it does say is that if the user's expectation of cluster similarity is low, grouping all transactions into a single cluster gives the most satisfactory solution (because the user does not care anyway).

```
/* Allocation phase */
(1)  while not end of the file do
(2)     read the next transaction $< t, - >$;
(3)     allocate $t$ to an existing or a new cluster $C_i$ to
           maximize $Cost(\mathcal{C})$;
(4)     write $< t, C_i >$;

/* Refinement phase */
(5)  repeat
(6)     not_moved=true;
(7)     while not end of the file do
(8)        read the next transaction $< t, C_i >$;
(9)        move $t$ to an existing non-singleton cluster
              $C_j$ to minimize $Cost(\mathcal{C})$;
(10)       if $C_i \neq C_j$ then
(11)          write $< t, C_j >$;
(12)          not_moved=false;
(13)          eliminate any empty cluster;
(14) until not_moved;
```

Figure 1: The overview of the clustering algorithm

4 Overview of Our Algorithm

Figure 1 shows the overview of our algorithm. The collection of transactions is stored in a file on disk. We read each transaction t in sequence, either assign t to an existing cluster (initially none) or create t as a new cluster, whichever minimizes $Cost(\mathcal{C})$ for the current clustering $\mathcal{C}$. The cluster identifier of each transaction is written back to the file. This is called *Allocation* phase. In *Refinement* phase, we read each transaction t (in the same order as in Allocation phase), move t to an existing non-singleton cluster (possibly stay where it is) to minimize $Cost(\mathcal{C})$. After each move, the cluster identifier is updated and any empty cluster is eliminated immediately. If no transaction is moved in one pass of all transactions, Refinement phase terminates; otherwise, a new pass begins. Essentially, at each step we locally optimize the criterion $Cost(\mathcal{C})$. The key step is finding the destination cluster for allocating or moving a transaction. This will be the topic in Section 5.

The paradigm of allocation phase followed by refinement phase has been adopted in partitional clustering algorithms such as the k-means and k-medoids. However, there are important differences in our algorithm. First, we do not require a pre-determined number k

of clusters; instead, we create and eliminate clusters dynamically on the basis of optimizing our criterion $Cost(\mathcal{C})$. Second, the destination cluster of a transaction is determined not by the nearest distance to a cluster or any mean/centroid of a cluster, but by optimizing the criterion $Cost(\mathcal{C})$. In fact, the nearest distance approach does not work in our case because a new cluster can always be created to be nearest to the next transaction.

5 Update the Criterion Function

In our algorithm, a key step is finding the destination cluster for a transaction at lines (3) and (9) in Figure 1. This requires to compute the new $Cost(\mathcal{C})$ for each possible destination cluster. To avoid scanning all transactions in the destination cluster, we maintain $|Large_i|$, $|\cup_{i=1}^{k} Small_i|$, and $|\cup_{i=1}^{k} Large_i|$ after each allocation or move of a transaction. We expect only a small change on $Large_i$ and $Small_i$ as a single transaction is added to or moved out of C_i. Therefore, this maintenance can be very efficient. We consider only the maintenance for adding a transaction to a cluster. The maintenance for moving a transaction out of a cluster is similar. First, let us explain the data structures maintained for each cluster C_i.

Let $MinSup_i = \lceil \theta * |C_i| \rceil$. The *support* of an item in C_i refers to the number of transactions in C_i that contain the item. Thus, an item is large in C_i if and only if its support in C_i is greater than or equal to $MinSup_i$. For each cluster C_i, we maintain two data structures in memory, i.e., hash table $Hash_i$ and B-tree $Btree_i$. Hash tables and B-trees are standard indexing techniques for large databases. For more information, we recommend [12] or any database text book.

$Hash_i$: The hash table for C_i with items as the index key. For each item e in C_i, there is an entry of the form $< e, tree\_addr >$ in $Hash_i$, where $tree\_addr$ is the address of the corresponding leaf entry for e in $Btree_i$ (see below). $Hash_i$ provides the access path to insert, delete, or update the support of a given item.

$Btree_i$: The B-tree with the support of items in C_i as the index key. For each item e in C_i, there is a leaf entry of the form $< sup, hash\_addr >$ in $Btree_i$, where sup is the support of e in C_i and $hash\_addr$ is the address of the corresponding entry for e in $Hash_i$. $Btree_i$ provides the access path to find all items having a given support.

The minimum support $MinSup_i$ separates the leaf entries of $Btree_i$ into those for $Large_i$ (on the right) and those for $Small_i$ (on the left). Of particular interests are those items near the boundary: the small items having support $MinSup_i - 1$ and the large items having support $MinSup_i$. As a transaction is allocated to or moved away from C_i, the support of some of these items will be increased or decreased by 1. Consequently, these items may move across the boundary. Keeping track of such changes efficiently is the main task of the maintenance. First, we define two operations.

We define $Inc(C_i, e)$ to be the operation that increases the support of a given item e in C_i by 1. Several steps are involved:

1. Look up $Hash_i$ for the entry $< e, tree\_addr >$. Let $< sup, hash\_addr >$ be the leaf entry in $Btree_i$ addressed by $tree\_addr$.
2. Increase sup by 1 in $< sup, hash\_addr >$.
3. Move $< sup, hash\_addr >$ to the right to pass all leaf entries $< sup', hash\_addr' >$ with $sup' < sup$.
4. For each entry $< sup', hash\_addr' >$ moved in (c), update its tree address contained in the corresponding entry in $Hash_i$.
5. Update ancestor index entries of $< sup, hash\_addr >$ to reflect the change of the support, if necessary.

5.1 Join transaction t into cluster C_i

As transaction t joins cluster C_i, $MinSup_i$ increases by at most 1 and the support of every item in t increases by 1. Let $OldMinSup_i$ and $MinSup_i$ denote the minimum support for C_i before and after t joins C_i.

5.1.1 Update $|Large_i|$

The algorithm for updating $|Large_i|$ is given in Figure 2. For each item e in t, we look up $Hash_i$. If e is found, we increment its sup in $Btree_i$. If e is not found, we insert e with $sup = 1$ into $Hash_i$ and $Btree_i$. These are given in lines (4)-(9).

Small items become large: A small item e becomes large if (a) $MinSup_i = OldMinSup_i$, (b) e is in t, and (c) $sup = MinSup_i$. This case is checked by lines (10)-(13).

Large items become small: A large item e becomes small if (a) $MinSup_i = OldMinSup_i + 1$, (b) e is not in t, and (c) $sup = OldMinSup_i$. This case is checked in lines (14)-(17).

5.1.2 Update $|\cup_{i=1}^{k} Small_i|$ and $|\cup_{i=1}^{k} Large_i|$

We use two hash tables $LargeHash$ and $SmallHash$ to maintain the number of clusters in which an item is large and small. As a small item e becomes large in a cluster, its number in $SmallHash$ is decreased by 1 and its number in $LargeHash$ is increased by 1. Similarly, as a large item e becomes small in a cluster, its numbers in $LargeHash$ and $SmallHash$ are updated accordingly. Whenever a number reaches 0 in a hash table, the corresponding entry is deleted from that table.

```
(1)  $|C_i| + +$;
(2)  $OldMinSup_i = MinSup_i$;
(3)  $MinSup_i = \lceil \theta * |C_i| \rceil$;
     /* update the support of items in $t$ */
(4)  **foreach** item $e$ in $t$ **do**
(5)      look up $Hash_i$ for $e$;
(6)      **if** $e$ is found **then**
(7)          $Inc(C_i, e)$;
(8)      **else**
(9)          insert $e$ into $Hash_i$ and $Btree_i$ with $sup = 1$;
     /* small items become large */
(10) **if** $MinSup_i == OldMinSup_i$ **then**
(11)     search $Btree_i$ for the items $e$ with $sup = MinSup_i$;
(12)     **foreach** returned item $e$ **do**
(13)         **if** $e$ is in $t$ **then** $|Large_i| + +$;
     /* large items become small */
(14) **if** $MinSup_i == OldMinSup_i + 1$ **then**
(15)     search $Btree_i$ for the items $e$
             with $sup = OldMinSup_i$;
(16)     **foreach** item $e$ returned **do**
(17)         **if** $e$ is not in $t$ **then** $|Large_i| - -$;
```

Figure 2: Update $|Large_i|$ for joining t into C_i

Whenever a new item e is added to a cluster, a new entry with the initial number 1 is inserted to $LargeHash$ or $SmallHash$, depending on whether e is large or small in that cluster. As a transaction t joins a cluster, the change of $| \cup_{i=1}^{k} Small_i |$ ($| \cup_{i=1}^{k} Large_i |$, resp.) is the number of new entries inserted minus the number of entries deleted in $SmallHash$ ($LargeHash$, resp.). These computations can be easily incorporated into the algorithm in Figure 2.

In actual implementation, the update of data structures is performed only after all potential destination clusters are tested.

5.2 Move transaction t away from cluster C_i

A similar reasoning applies to the case of moving a transaction away from a cluster. We omit the detail.

6 Experiments

Most clustering algorithms do not scale up for large databases because they require to examine pairs of objects due to a pairwise similarity measure. k-means algorithms, on the other hand, assume a fixed number k of clusters, thus, cannot be applied to applications where the number of clusters can evolve. In this experiment, we compare our method with two algorithms, the traditional hierarchical clustering [9], denoted HC, and the link-based hierarchical clustering [7], denoted LBHC. [7] addressed the limitation of distance metric between points (which we call pairwise similarity) using the concept of *links* to measure the similarity between a pair of data points, where links are the common "neighbors", in any distance measure, of the two points. It was shown in [7] that the link-based measure produces better clusterings than distance-based measures. To facilitate comparison, we use the same data sets as used in [7], i.e., the Congressional votes and Mushroom data sets from the UCI Machine Learning Repository (http://www.ics.uci.edu / mlearn/MLRepository.html). (The US Mutual Funds in [7] is not available to us because the web page does not exist any more.) We map a table to a collection of transactions by creating one transaction to contain attribute/value pairs in each record in the table.

Congressional votes. This is the 1984 United States Congressional Votes Records Database. Every record contains 16 boolean attributes corresponding to one congressman's votes on 16 key issues. (A few records contain missing values, which we treat as NO.) There are 435 records, 168 for Republicans and 267 for Democrats. All clustering algorithms ignore the class labels "Republicans" and "Democrats". The class labels are only used to verify the clustering result.

Table 1(a) contrasts the clustering results on the Congressional votes data. In our algorithm, we set the minimum support to 60% to reflect the fact that democrats and republican do vote similarly on many issues. The clusters produced by BLHC covers only 372 records out of the 435 records because the rest are treated as outliers by [7]. The clusters produced by our algorithm and HC cover all 435 records. Our algorithm better separates "Democrat" and "Republican" than HC and BLHC in terms of class purity of each cluster. Table 1(b) shows some characteristics of the two larger clusters produced by our algorithm, given by large items. For example, (physician-fee-freeze,157 (93%)) for Cluster 1 means that 157 or 93% records in this cluster vote for physician-fee-freeze. As Table 1(b) indicates, these two clusters, one representing Republicans and one representing Democrats, have very different voting characteristics.

Mushroom. Each record contains information about the physical attributes of a single mushroom, e.g., color, odor, shape, habitat, etc. A class label of "Poisonous" or "Edible" is assigned to each record. There are 8,124 records for 4,208 edible mushrooms and 3,916 poisonous mushrooms.

Table 2 shows the clusters and class distribution produced by the three algorithms. In our algorithm, Clusters 1-14 are obtained hierarchically. First, we set the minimum support at 20% and obtain Clusters 1-4 and a huge cluster called X. We fine-cluster X at the minimum support of 50%, yielding Clusters 5-8 and a huge clus-

	HC		LBHC		Our algorithm	
Cluster No	No of Rep	No of Demo	No of Rep	No of Demo	No of Rep	No of Demo
1	157	52	144	22	161	3
2	11	215	5	201	7	257
3					0	7

(a) Class distribution in clusters

Cluster 1 (for Republicans)	Cluster 2 (for Democrats)
(physician-fee-freeze, 157 (93%))	(adoption-of-the-budget-resolution, 226 (85%))
(crime, 154 (92%))	(aid-to-nicaraguan-contras, 215 (81%))
(el-salvador-aid, 154 (92%))	(anti-satellite-test-ban, 197 (74%))
(religious-groups-in-schools, 145 (86%))	(mx-missle, 185 (69%))
(superfund-right-to-sue, 132 (78%))	(export-administration-act-south-africa,173 (65%))
(education-spending, 131 (78%))	

(b) Voting characteristics produced by our algorithm

Table 1: Clustering results on the congressional vote data set

ter called Y. We further fine-cluster Y at the minimum support of 35% and obtain Clusters 9-14. The decision on whether to fine-cluster a cluster is made on the basis of reducing $Cost(\mathcal{C})$ and the minimum support is chosen based on our expectation. Except for Clusters 6,7 and 12, which together contains only 3.9% of the data, our algorithm is able to distinguish the two classes. We can further fine-cluster Clusters 6,7 and 12 to get more purity, but we are satisfied with the current clustering.

As Table 2 shows, our algorithm and LBHC perform much better than HC. LBHC achieves the purity of classes by producing more clusters. However, the purity alone does not capture the "structure" of the data in a concise representation. To explain this point, let us consider the 2 largest clusters produced by each algorithm (because 2 is the ideal number of clusters). For LBHC, the 2 largest clusters, i.e., Clusters 7 and 16, cover 43% of the data with pure separation. For our algorithm, the 2 largest clusters, i.e., Clusters 5 and 9, cover 84% of the data with nearly pure separation. By doubling the coverage without losing separation power, our algorithm better captures the structure of the data. Due to the space limitation, the characteristics of clusters are left out.

In summary, our algorithm does group similar transactions together and group dissimilar transactions apart. In several other experiments, we studied the scalability and order sensitivity of our algorithm. The result shows that our algorithm makes only a few, typically 2 or 3, scans of the database and the execution time scales up linearly with the size of the database. The result also shows that the processing order of transactions does not have a major impact on the clustering. Due to the space limitation, we omit the detail.

7 Conclusion

Most clustering methods rely on a measure of pairwise similarity, i.e., the distance measure. For transactions made of sparsely distributed items, we argued that pairwise similarity is neither necessary nor sufficient for a cluster of transactions to be similar. We proposed a new clustering criterion based on the notion of *large items* without using any measure of pairwise similarity. This new clustering criterion helps address several important issues in transaction clustering, namely, robustness of similarity measures, incorporating user expectation of similarity, discriminating features and noises, dynamic clustering, and handling large data sets. We presented an I/O-efficient clustering algorithm based on the new clustering criterion. Experiments show that our approach is effective.

References

[1] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. SIGMOD 1993, 207-216

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse and G. Zweig. Syntactic clustering of the web. WWW Conference 1997

[3] D. R. Cutting, D. R. Karger, J. O. Pederson and J. W. Turkey. Scatter/Gather: A cluster-based approach to browsing large document collections. SIGIR 1992, 318-29

[4] P. Cheeseman, J. Stutz. Baysian classification (autoclass): Theory and results. In U.M. Fayyad, G.

HC					
Cluster No	No of Edible	No of Poisonous	Cluster No	No of Edible	No of Poisonous
1	666	478	11	120	144
2	283	318	12	128	140
3	201	188	13	144	163
4	164	227	14	198	163
5	194	125	15	131	211
6	207	150	16	201	156
7	233	238	17	151	140
8	181	139	18	190	122
9	135	78	19	175	150
10	172	217	20	168	206
LBHC					
Cluster No	No of Edible	No of Poisonous	Cluster No	No of Edible	No of Poisonous
1	96	0	12	48	0
2	0	256	13	0	288
3	704	0	14	192	0
4	96	0	15	32	72
5	768	0	16	0	1728
6	0	192	17	288	0
7	1728	0	18	0	8
8	0	32	19	192	0
9	0	1296	20	16	0
10	0	8	21	0	36
11	48	0			
Our algorithm					
Cluster No	No of Edible	No of Poisonous	Cluster No	No of Edible	No of Poisonous
1	94	0	8	0	287
2	13	0	9	61	3388
3	6	0	10	372	77
4	682	26	11	9	0
5	2631	30	12	19	10
6	121	37	13	21	0
7	69	61	14	110	0

Table 2: Clustering results on the mushroom data set

Piatetsky-Shapiro, O. Smith, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, 153-180. AAAI/MIT Press, 1996

[5] M. Ester, H. P. Kriegel, J. Sander, M. Wimmer, X. Xu. Incremental clustering for mining in a data warehousing environment. VLDB 1998, New York, USA

[6] M. Ester, H-P Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD 1996

[7] S. Guba, R. Rastogi, K. Shim. A clustering algorithm for categorical attributes. ICDE 1999

[8] E.H. Han, G. Karypis, V. Kumar and B. Mobasher. Clustering based on association rule hypergraphs. SIGMOD workshop on research issues on Data Mining and Knowledge Discovery, 1997

[9] A.K. Jain, R.C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Englewood Cliffs, N.J., 1988

[10] L. Kaufman and P.J. Rousseeuw. Finding groups in data: an introduction to cluster analysis, John Wiley & Son, 1990

[11] Van Rijsbergen, C.J. *Information retrieval.* London: Butterworths, 1979

[12] R. Ramakrishnan, Database Management Systems, McGRAW-HILL, 1998

[13] P. Willet. Recent trends in hierarchical document clustering: A critical review. Information Processing & Management, 24(5):577-597, 1988.

[14] X. Xu, M. Ester, H.P.Kriegel, J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. ICDE 1998

[15] O. Zamir, O. Etzioni, O. Madani and R. M. Karp. Fast and intuitive clustering of web documents. KDD 1997, 287-290

[16] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. SIGMOD 1996, 103-114.