Data Mining and Knowledge Discovery, 9, 29–57, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Hypergraph Models and Algorithms for Data-Pattern-Based Clustering*

MUHAMMET MUSTAFA OZDAL ozdal@uiuc.edu
Department of Computer Science, University of Illinois at Urbana-Champaign

CEVDET AYKANAT aykanat@cs.bilkent.edu.tr
Computer Engineering Department, Bilkent University

Editors: Fayyad, Mannila, Ramakrishnan

Received April 2, 2001; Revised February 7, 2003

Abstract. In traditional approaches for clustering market basket type data, relations among transactions are modeled according to the items occurring in these transactions. However, an individual item might induce different relations in different contexts. Since such contexts might be captured by interesting patterns in the overall data, we represent each transaction as a set of patterns through modifying the conventional pattern semantics. By clustering the patterns in the dataset, we infer a clustering of the transactions represented this way. For this, we propose a novel hypergraph model to represent the relations among the patterns. Instead of a local measure that depends only on common items among patterns, we propose a global measure that is based on the cooccurences of these patterns in the overall data. The success of existing hypergraph partitioning based algorithms in other domains depends on sparsity of the hypergraph and explicit objective metrics. For this, we propose a two-phase clustering approach for the above hypergraph, which is expected to be dense. In the first phase, the vertices of the hypergraph are merged in a multilevel algorithm to obtain large number of high quality clusters. Here, we propose new quality metrics for merging decisions in hypergraph clustering specifically for this domain. In order to enable the use of existing metrics in the second phase, we introduce a vertex-to-cluster affinity concept to devise a method for constructing a sparse hypergraph based on the obtained clustering. The experiments we have performed show the effectiveness of the proposed framework.

Keywords: data mining, clustering, hypergraph models, data patterns

1. Introduction

Clustering can be informally defined as grouping data items together such that the items belonging to the same group are similar to each other. This problem has several important applications in data mining. For example large corporations may be interested in grouping their customers according to their buying patterns for various purposes such as targeted advertisements, customized promotions, catalog design. Generally, a market database is considered as a set of transactions such that each transaction contains all the items purchased

*This work was partially supported by The Scientific and Technical Research Council of Turkey (TUBITAK) under Grant 199E013.

by one customer. So, the customer segmentation problem becomes equivalent to clustering these transactions based on the items they contain.

In most of the existing clustering algorithms, relations between transactions are defined based on the individual items they contain. However, it is possible that an item might induce different relations in different contexts. For example, consider three transactions $T_1 = \{milk, butter, cheese\}$, $T_2 = \{milk, egg, bread\}$ and $T_3 = \{milk, babyfood, diapers\}$. Although *milk* is common among all three transactions, its meaning is different in the first two transactions compared to the third one. We can argue that T_1 and T_2 correspond to customers who buy food for breakfast, while T_3 corresponds to a customer with infant. So, modeling relations among transactions based only on the occurrences of items may not always be appropriate.

In a dataset, some correlations exist between different items due to their cooccurences in transactions. For instance a correlation might exist between the items *milk*, *toy*, *babyfood* and *diapers* due to customers with infants who buy these items frequently together. So a customer might be considered as incurring relations over a set of items. Assume that by using a pattern discovery algorithm, a set of *interesting* relations have been discovered for the whole database. Then we can model the similarity between different transactions based on the interesting relations they incur. For instance, if $p_1 = \{babyfood, diapers, toy\}$ is found to be an interesting relation, we can argue that the transactions that incur this relation are similar to each other. Furthermore, it is possible that some of the interesting relations are similar to each other. For this example, a relation such as $p_2 = \{milk, babyfood, toy\}$ is expected to be similar to p_1. In this case, the transactions that incur a relation of the form p_1 should also be considered to be similar to the transactions that incur p_2. Note that we will refer the relations among a set of items as *patterns* throughout the paper to avoid confusion. By grouping similar patterns in a database together, it is possible to infer a clustering of the transactions that incur these relations.

The framework we use for pattern clustering is as given in figure 1. A dataset $\mathcal{D}$ is represented as a tuple $(\mathcal{I}, \mathcal{T})$, corresponding to the set of items and the set of transactions respectively. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices (Berge, 1976). Each net $n \in \mathcal{N}$ is a set of vertices in $\mathcal{V}$, and defines a relation among them. Observe that it is possible to consider a one-to-one correspondence between $\mathcal{D} = (\mathcal{I}, \mathcal{T})$ and $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. Here, each item in $\mathcal{I}$ can be viewed as a vertex in $\mathcal{V}$ and each transaction in $\mathcal{T}$ as a net in $\mathcal{N}$ so that each transaction defines a relation among the set of items it contains.

For the ease of presentation, the implementation details and choices are omitted in figure 1. For example, it is an implementation issue to choose which data representation to use in each step (i.e. $\mathcal{D}_i$ or $\mathcal{H}_i$). Also, it might be more convenient to combine some of those steps together, and avoid extra storage required for the intermediate data representations. In this paper, we will not discuss such issues, but will only concentrate on the main concepts.

In this framework, we assume that an algorithm to find interesting cooccurence patterns in a dataset is available. In figure 1, $\mathcal{D}_0$ is the original dataset, and $\mathcal{P}$ is the interesting pattern set in $\mathcal{D}_0$. We give definitions in Section 3.1 to represent each transaction as a set of patterns (i.e. as in $\mathcal{D}_1$), instead of a set of items (i.e. as in $\mathcal{D}_0$). The similarities among different patterns are determined based on the cooccurences of these patterns in transactions. So, in

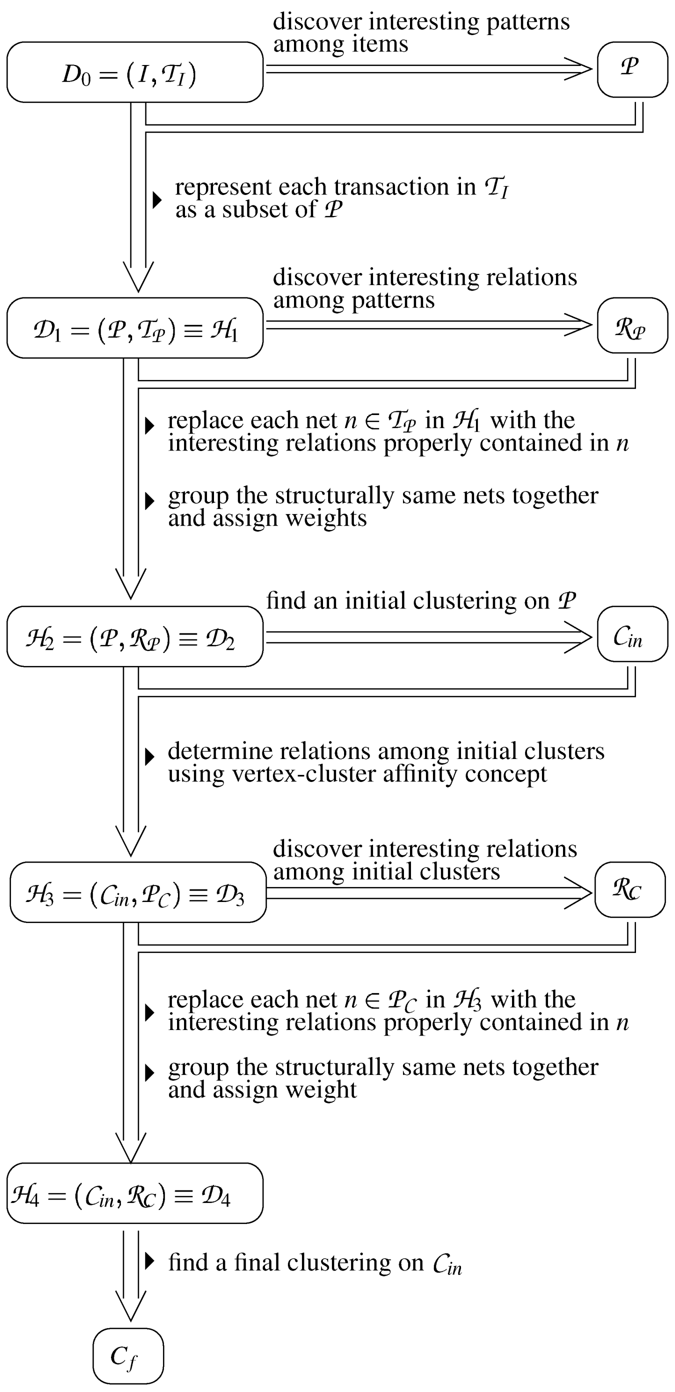

discover interesting patterns among items
$D_0 = (I, \mathcal{T}_I)$
$\mathcal{P}$
represent each transaction in $\mathcal{T}_I$ as a subset of $\mathcal{P}$
discover interesting relations among patterns
$\mathcal{D}_1 = (\mathcal{P}, \mathcal{T}_\mathcal{P}) \equiv \mathcal{H}_1$
$\mathcal{R}_\mathcal{P}$
replace each net $n \in \mathcal{T}_\mathcal{P}$ in $\mathcal{H}_1$ with the interesting relations properly contained in n
group the structurally same nets together and assign weights
find an initial clustering on $\mathcal{P}$
$\mathcal{H}_2 = (\mathcal{P}, \mathcal{R}_\mathcal{P}) \equiv \mathcal{D}_2$
$\mathcal{C}_{in}$
determine relations among initial clusters using vertex-cluster affinity concept
discover interesting relations among initial clusters
$\mathcal{H}_3 = (\mathcal{C}_{in}, \mathcal{P}_C) \equiv \mathcal{D}_3$
$\mathcal{R}_C$
replace each net $n \in \mathcal{P}_C$ in $\mathcal{H}_3$ with the interesting relations properly contained in n
group the structurally same nets together and assign weight
$\mathcal{H}_4 = (\mathcal{C}_{in}, \mathcal{R}_C) \equiv \mathcal{D}_4$
find a final clustering on $\mathcal{C}_{in}$
$\mathcal{C}_f$

Figure 1. The framework for clustering patterns.

$\mathcal{H}_1$, transaction set $\mathcal{T}_\mathcal{P}$ defines a relation among the pattern set $\mathcal{P}$. Since $\mathcal{H}_1$ is in the size of the original dataset, a reduction technique is proposed in Section 3.2 to obtain $\mathcal{H}_2 = (\mathcal{P}, \mathcal{R}_\mathcal{P})$.

The two-phase bottom-up clustering algorithm given in Section 4 is applied on $\mathcal{H}_2$ to obtain a set of pattern clusters C_f. The proposed clustering metrics and multilevel clustering approach used in the first phase are discussed in Sections 4.1 and 4.2, respectively. In the first phase, a large number of clusters with small sizes (i.e. set C_{in}) is obtained. Here, although each vertex belongs to only one cluster, it can have *affinities* to other clusters. If a vertex has affinities to a set of clusters, it is possible to argue that a relation exists among these clusters. To capture these relations, hypergraph $\mathcal{H}_3$ is created as discussed in Section 4.3, and then a method similar to the one given in Section 3.2 is used to obtain reduced hypergraph $\mathcal{H}_4$. Finally in the second phase, the existing hypergraph-partitioning metrics are used on $\mathcal{H}_4$ to obtain a final clustering C_f, since $\mathcal{H}_4$ is expected to be sparse and well-separated.

To test the validity of this framework, the clustering result C_f is used to cluster the transactions in the given dataset. For this, the transactions are simply assigned to clusters according to the patterns that they contain. Section 5 demonstrates such results for two real life datasets (Section 5.1), as well as for synthetic datasets (Section 5.2). Experimental results on synthetic datasets are used to perform empirical performance comparison with state-of-the-art clustering algorithm ROCK in terms of sensitivity and scalability analysis.

2. Related work

In this section an overview of the relevant research conducted in this area is given. For a more thorough analysis of clustering algorithms, the reader may refer to some survey papers on this subject (Jain et al., 1999; Fasulo, 1999).

The k-means algorithm (MacQueen, 1967) is one of the widely-used *partitional* clustering algorithms. Here, data is mapped onto a d-dimensional metric space and a distance function between data points is defined. The aim of the algorithm is to partition the data into k clusters such that the distance of each data point to the mean of its cluster is minimized. In some cases it might not be possible to define the mean of a cluster. For this, a similar algorithm called k-medoids (Kaufman and Rousseeuw, 1990) has been proposed, where a data point is chosen as a representative of each cluster. These algorithms have been given mainly for data with numerical attributes. The k-modes algorithm (Huang, 1997) extends these concepts to the categorical domains by defining new dissimilarity measures. There are two inherent weaknesses of these approaches. First, they assume that the clusters are in the form of a sphere; second, noise in the data may pose some problems. One of the early scalable algorithms in this class is CLARANS (Clustering Large Applications based upon Randomized Search) (Ng and Han, 1994). Also, Bradley et al. (1998) have given a scalable clustering framework applicable to such algorithms.

Another class of clustering algorithms is *hierarchical* algorithms. These algorithms find a dendogram representing the nested sequence of clusters. *Divisive* hierarchical algorithms start with one cluster and recursively partition it to find the singleton clusters. Conversely, *agglomerative* hierarchical algorithms start with singleton clusters and successively merge them. The metrics used in such algorithms are generally variations of the single-link (Sneath and Sokal, 1973), average-link (Voorhees, 1986), complete-link (King, 1967) and minimum variance (Ward, 1963) metrics. Some of the hierarchical clustering algorithms for large datasets are CURE (Guha et al., 1998) and ROCK (Guha et al., 1999).

It is also possible to scan data to compute certain data summaries in the preprocessing step. Then, these summaries can be used instead of the original data in the clustering algorithm. BIRCH (Zhang et al., 1996) is one of the earliest algorithms that propose such a preclustering approach for the purpose of scalability. An extension of BIRCH has recently been given for mixed numerical and categorical attributes by Chiu et al. (2001). Also, similar ideas have been applied for metric spaces in BUBBLE (Ganti et al., 1999).

Generally, the distance based algorithms do not perform well when the number of dimensions is high. The main reason is the distribution of data over different dimensions. That is, there might exist some irrelevant attributes, and the data points may be far apart due to these. One method to overcome this problem is to use feature selection or extraction algorithms such as Principal Component Analysis (Jolliffe, 1986) to reduce the dimension of the dataset. FastMap (Faloutsos and Lin, 1995) is a dimensionality reduction algorithm that is scalable for large datasets. However, one problem with such an approach is that the relevant features may be different for different clusters. So, it might not be possible to obtain features that are relevant to all clusters. As a remedy for this, Agrawal et al. (1998) give a scalable algorithm CLIQUE to identify dense regions in subspaces of maximum dimensionality. There exist also variants of this algorithm such as PROCLUS (Projected Clustering) (Aggarwal et al., 1999), MAFIA (Merging of Adaptive Finite Intervals) (Nagesh et al., 1999), ENCLUS (Entropy-based Clustering) (Cheng et al., 1999), and ORCLUS (Oriented Projected Cluster Generation) (Aggarwal and Yu, 2000).

Although these algorithms can be used effectively for numerical attributes, the *region* definition may not be appropriate for categorical data, mainly because there exists no natural ordering between the values of such attributes. As an extension, Ganti et al. (1999) have defined *interval region* for categorical data as the cross product of subsets of attribute values.

Another scalable algorithm for clustering categorical attributes is ROCK (Guha et al., 1999). Here, unlike CACTUS, each cluster is defined as a set of data tuples. The similarity between two tuples is defined to be the number of *common neighbors* for them. Since the complexity of the algorithm is quadratic in the number of input data, the algorithm is first applied on a random sample from the dataset, and then the remaining data is assigned to the clusters discovered. We give an empirical comparison of ROCK with our algorithm in Section 5.2.

Han et al. (1997) have given a clustering algorithm based on the large itemsets discovered in the database. Here, each itemset is considered as a relation over some set of items, and a hypergraph model is used to represent these relations. A hypergraph partitioning algorithm is then used to obtain a set of item clusters. Finally as a postprocessing step, the vertices and clusters that do not satisfy some constraints are eliminated. A disadvantage of such an approach is that some unnatural constraints like *balance* are required for such partitioning algorithms. It is possible to relax such constraints, but then some assumptions about the input data (e.g. sparsity, well separatedness, etc.) might be required. Another disadvantage here is that the clusters are defined as distinct sets of items. However, it is possible that some items might define different clusters in different contexts, as explained in the previous section.

3. Relation modeling

We assume that the given dataset $\mathcal{D} = (I, \mathcal{T})$ is in the form of a market basket data, and it contains a set of items I and a set of transactions $\mathcal{T}$, such that each $t \in \mathcal{T}$ is a subset of I. Our discussions in this paper will be based on cooccurence patterns; however it is also possible to extend these ideas to sequential relations (Feldman, 1997). To determine the interestingness of a pattern, we define a weight function in Appendix A based on confidence (Agrawal et al., 1993) values. However, it is also possible to incorporate other measures such as interest (Silverstein et al., 1997) and improvement (Bayardo et al., 2000). In the subsequent subsections, we give definitions for representing each transaction as a set of these patterns and then we propose a hypergraph model based on these concepts.

3.1. Pattern representation

Given a set of interesting patterns $\mathcal{P}$ derived from the transaction set $\mathcal{T}$, we propose the following definition to represent each transaction $t \in \mathcal{T}$ as a set of patterns in $\mathcal{P}$:

Definition 3.1. A transaction t is defined to *contain* a pattern $p \in \mathcal{P}$ *properly* if and only if t contains p while not containing any superset of p, i.e., $p \subset t$ and there does not exist any $p_s \in \mathcal{P}$ for which $p \subset p_s$ and $p_s \subset t$.

This definition is mainly because of our assumption that we can infer no relation between patterns by only inspecting the items that they contain. Initially, we treat all interesting patterns as independent of each other, even if they have subset-superset relationships. Consider for example two interesting patterns $p_1 = \{milk, sugar\}$ and $p_2 = \{milk, sugar, flour\}$. If for a transaction t, $p_1 \subset p_2 \subset t$, we say that the relationship t induces among these items is of the form $\{milk, sugar, flour\}$, but not of the form $\{milk, sugar\}$. That is because both p_1 and p_2 have been discovered as interesting patterns, and it is possible that they have different meanings. For instance p_1 might be induced by the customers who like to drink milk with sugar; however, the existence of *flour* in t might change the context to the customers who want to prepare pastry.

This suggests a modification of the usual semantics for a pattern. The conventional meaning of a pattern is the existence of the items in that pattern. So, a pattern p will always imply its subset patterns, because the existence of the items in p will imply the existence of the items in the subsets of p. However in our case, we do not want such implications to exist between patterns, because we assume that we can not infer a relation between patterns based only on the items they contain. For this reason, we give a different semantics for a pattern as suggested by Definition 3.1. Here, if a relation among a set of items can be captured by a pattern p, then the patterns that are subsets of p do not exist for that set of items. To put it another way, for a pattern p to exist in a transaction t, it is required not only the existence of the items in p but also the absence of items that will cause any superset of p to exist in t. For the above example, pattern p_1 means that *milk* and *sugar* exist, but *flour* does not. So, the customers who buy *milk*, *sugar*, and *flour* are initally distinguished from the ones who buy only *milk* and *sugar*.

On the other hand, there might be an immediate relation between two patterns such as $p_1 = \{\textit{coffee}, \textit{sugar}\}$ and $p_2 = \{\textit{coffee}, \textit{sugar}, \textit{cream}\}$, although our initial relation modeling distinguishes them. However, we can argue that these two patterns are expected to be clustered together in the later phases of the algorithm if they really belong to the same cluster. The main idea here is to model the relationships according to global relations rather than just the contents. As an example consider the items *coffee* and *water*, which probably characterize more than one cluster in a dataset. Assume that there are patterns as $p_1 = \{\textit{coffee}, \textit{water}\}$, $p_2 = \{\textit{coffee}, \textit{water}, \textit{bagel}, \textit{cheese}\}$, and $p_3 = \{\textit{coffee}, \textit{water}, \textit{chips}, \textit{crackers}\}$. In the beginning, our relation model distinguishes each of these patterns, and considers them as separate entities. After the clustering algorithm, they might end up in the same cluster, or in different clusters, depending on the global relations in the dataset. For instance, it is possible that p_2 and p_3 might end up in the clusters that characterize *breakfast items* and *snacks*, correspondingly, while p_1 is left out as an outlier. The point here is that we can not conclude a direct relationship among these patterns in the beginning, just because they contain *coffee* and *water*.

In association rule mining problem (Agrawal et al., 1993), *support* of a pattern p (denoted as $\textit{sup}(p)$) in a dataset $\mathcal{D}$ is defined to be the ratio of the number of transactions that contain all the items in p to the total number of transactions in $\mathcal{D}$. To extend this idea to the new semantics of the patterns, we propose the following definition:

Definition 3.2. Assume that we are given a dataset $\mathcal{D}$ and a set of patterns $\mathcal{P}$ for this dataset. The *proper count* of a pattern $p \in \mathcal{P}$, denoted as $\textit{procount}(p)$, is defined to be the number of transactions in $\mathcal{D}$ that *contain p properly*. The *proper support* of p is the ratio of proper count of p to the total number of transactions in $\mathcal{D}$, and it is denoted as $\textit{prosup}(p)$.

It is possible to make modifications on the itemset counting algorithms such as Apriori (Agrawal et al., 1993) or DIC (Brin et al., 1997) for the purpose of finding the proper support values of patterns. In Appendix B, we give such an algorithm, which is based on DIC.

3.2. *Hypergraph modeling and relation summarization*

As discussed earlier, having items in common may not necessarily mean that there is a relation between patterns. Here, we propose a hypergraph model that uses the information of the relations that exist globally in the database while determining the local relations.

Since a transaction is a set of items, it is natural to view it as incurring a relation over the items it contains. Definition 3.1 suggests that it is also possible to view a transaction as a set of patterns. Hence, each transaction can be considered as incurring a relation also over some set of patterns. These relations can be modeled as a hypergraph by representing each pattern as a vertex and each transaction as a net, corresponding to $\mathcal{H}_1 = (\mathcal{P}, \mathcal{T}_{\mathcal{P}})$ in figure 1. Note that the number of nets in $\mathcal{H}_1$ is equal to the number of transactions in $\mathcal{D}_0$, which is expected to be too large. For this reason, we first determine the interesting relations over patterns in dataset $\mathcal{D}_1 = (\mathcal{P}, \mathcal{T}_{\mathcal{P}}) \equiv \mathcal{H}_1$, and then approximate the relation corresponding to each transaction in $\mathcal{H}_1$ by a set of these interesting relations. So, in the new hypergraph (i.e. $\mathcal{H}_2$), each transaction corresponds to a set of *interesting* nets instead

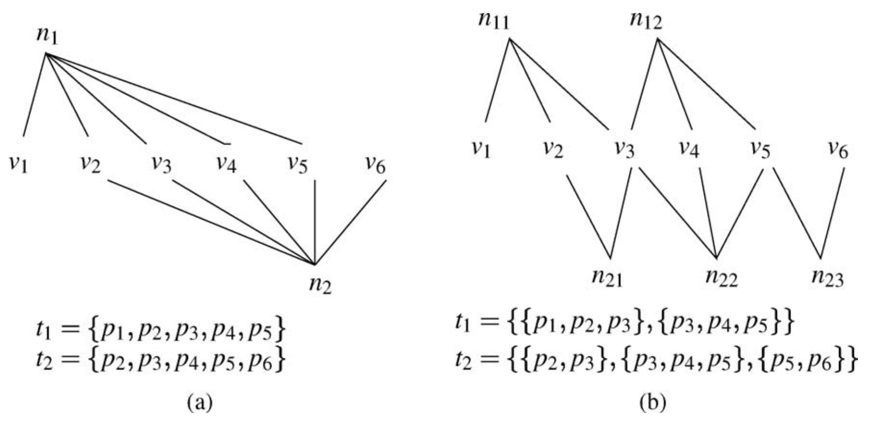

n_1
v_1 v_2 v_3 v_4 v_5 v_6
n_2
$t_1 = \{p_1, p_2, p_3, p_4, p_5\}$
$t_2 = \{p_2, p_3, p_4, p_5, p_6\}$
(a)
n_{11} n_{12}
v_1 v_2 v_3 v_4 v_5 v_6
n_{21} n_{22} n_{23}
$t_1 = \{\{p_1, p_2, p_3\}, \{p_3, p_4, p_5\}\}$
$t_2 = \{\{p_2, p_3\}, \{p_3, p_4, p_5\}, \{p_5, p_6\}\}$
(b)

Figure 2. Relations among patterns induced by transactions: (a) actual relations, and (b) reduced set of relations.

of only one net. Since an interesting relation is expected to occur in several transactions, there will be several nets in $\mathcal{H}_2$ that are structurally the same. By grouping such nets and assigning an appropriate weight to each group, we can reduce the number of nets in the hypergraph to the number of interesting relations discovered.

Figure 2 shows an example about such a hypergraph modeling, where each net is connected to its vertices through a set of line segments. In part (a), transactions t_1 and t_2 are represented as a set of patterns p_1–p_6. In the corresponding hypergraph model, there exists a vertex v_i for each pattern p_i, and a net n_j for each transaction t_j. Assume that in the overall dataset, the discovered interesting relations are $\{p_1, p_2, p_3\}$, $\{p_3, p_4, p_5\}$ and $\{p_5, p_6\}$ together with all their subsets. Depending on the concepts that Definition 3.1 is based on, we find the interesting relations that are maximal with respect to each transaction. In part (b), transactions t_1 and t_2 are shown as sets of such relations. Based on this, nets n_1 and n_2 are replaced with n_{11}, n_{12} and n_{21}, n_{22}, n_{23} respectively. Here, since n_{12} and n_{22} are structurally equivalent, they can be represented by only one net with a weight equal to 2 (assuming weights of individual nets are 1).

The rationale behind choosing the maximal relations with respect to a transaction can be explained by an instance of this example. Consider the interesting relations $\{p_1, p_2, p_3\}$ and $\{p_2, p_3\}$. Although both are subsets of t_1, only the maximal one $\{p_1, p_2, p_3\}$ has been taken for this transaction. Clearly the nets corresponding to these relations (i.e. n_{11} and n_{21}) have different semantics and t_1 conforms to only one of them (i.e. n_{11}).

Note that the hypergraph illustrated in figure 2(b) is in fact the reduced form of the hypergraph in figure 2(a). Here, the globally uninteresting relations have been eliminated while the interesting ones have been kept. For instance, since no interesting relation among p_4 and p_6 exists in the database, the relation imposed by transaction t_2 over these patterns is ignored. So, we can say that the local relations induced by n_1 and n_2 are remodeled in figure 2(b) according to the global relation information.

We can formally describe the above procedure for hypergraph modeling as follows. Assume that we are given a dataset $\mathcal{D}_0 = (I, \mathcal{T}_I)$ and a set of interesting patterns $\mathcal{P}$. We construct (possibly abstractly) a new dataset $\mathcal{D}_1 = (\mathcal{P}, \mathcal{T}_{\mathcal{P}})$ such that every transaction t_P in $\mathcal{T}_{\mathcal{P}}$ corresponds to one transaction t_I in $\mathcal{T}_I$ and t_P is the set of patterns $p \in \mathcal{P}$ that are properly contained in t_I. Then we find the interesting relation patterns in $\mathcal{D}_1$ by using some pattern discovery algorithm, and denote this set as $\mathcal{R}_{\mathcal{P}}$ (meaning the relation set over patterns). After that, we use an algorithm such as the one described in Appendix B to find the proper support of each relation in $\mathcal{R}_{\mathcal{P}}$. Finally, using the sets $\mathcal{P}$ and $\mathcal{R}_{\mathcal{P}}$, we construct hypergraph $\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2)$ such that $\mathcal{V}_2 = \mathcal{P}$ and $\mathcal{N}_2 = \mathcal{R}_{\mathcal{P}}$. That is, for every $p \in \mathcal{P}$ there exists a vertex in $\mathcal{V}_2$ and for every $r \in \mathcal{R}_{\mathcal{P}}$ there exists a net in $\mathcal{N}_2$. The weight of each net is assigned the proper support of the corresponding r.

4. Pattern clustering

We propose a two phase approach to cluster hypergraph $\mathcal{H}_2$, which represents the relations among patterns. As the first phase, a set of clusters $\mathcal{C}_{\text{in}}$ is obtained using a multilevel bottom-up hypergraph clustering algorithm. Then, the sparse and well separated hypergraph $\mathcal{H}_4$ is constructed from $\mathcal{H}_2$ based on these initial clusters. As the second phase, the existing metrics and algorithms are applied on $\mathcal{H}_4$ to find the final clustering result.

4.1. Clustering model and metrics

In most of the agglomerative hierarchical clustering algorithms (e.g. CURE (Guha et al., 1998), ROCK (Guha et al., 1999), CHAMELEON (Karypis et al., 1999) the decision of merging two clusters depends on the similarity between them. Such an approach is in fact appropriate for a graph model, because each edge in the graph represents a relation between a pair of vertices. However, a hypergraph model is more general in that the relations are given among a set of vertices. So using a metric that exploits this extra information is more appropriate in this case.

For the following discussions, assume that we are given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ and a set of clusters $\mathcal{C}$ such that each $c_i \in \mathcal{C}$ is a subset of $\mathcal{V}$ and for all pairs of clusters c_i and c_j $(i \neq j)$, $c_i \cap c_j = \emptyset$. In the given hypergraph, each net $n \in \mathcal{N}$ has a weight $w(n)$, while each vertex is assumed to be nonweighted. Since a net n and a cluster c are both subsets of vertices, their intersection can be defined as the number of vertices that are both in n and c, and denoted as $|n \cap c|$. Finally, the number of vertices in n is denoted as $|n|$ and the number of vertices in c is denoted as $|c|$.

For a net n and a cluster c, if $|n \cap c| \neq 0$, we can say that there exists a mutual relationship between n and c. We call such a relation *net cluster co-relation* and define it as follows:

Definition 4.1. Let $P(A \mid B)$ denote the conditional probability of A given that B has occurred. The *net cluster co-relation* $cr(n, c)$ between a net n and a cluster c is:

$$cr(n, c) \triangleq P(v \in c \mid v \in n) \times P(v \in n \mid v \in c), \tag{1}$$

where v denotes an arbitrary vertex.

In fact the co-relation function is the product of two one-way relations. The first multiplicand indicates *how much net n implies cluster c* and the second one indicates *how much cluster c implies net n*. This function can be formulated as:

$$cr(n, c) = \frac{|n \cap c|}{|n|} \times \frac{|n \cap c|}{|c|} \tag{2}$$

Note that throughout the paper, the first multiplicand will be referred to as *the fraction of n belonging to c*, and the second will be as *the fraction of c spanned by n*.

Assume that we are to merge clusters c_i and c_j and obtain cluster $c_{ij} = c_i \cup c_j$. For a net n for which $|n \cap c_i| \neq 0$, if $cr(n, c_{ij}) > cr(n, c_i)$, this means that: (1) n has vertices in also c_j, (2) there are not too many vertices not spanned by n in c_j. So, we can say that the change in the cr value captures both the similarity and the dissimilarity in terms of net n between clusters c_i and c_j. Before defining a gain function for merging decisions based on this concept, we should extend this idea for a cluster c to include all nets $n \in \mathcal{N}$ for which $cr(n, c) \neq 0$.

Definition 4.2. The *quality* of a cluster c, denoted as $Q(c)$, is defined to be the weighted sum of $cr(n, c)$ values for all $n \in \mathcal{N}$. Each weight in this sum is assigned according to the weight of the corresponding n and the number of vertices that n spans in c. This can be seen in the following formula:

$$Q(c) \triangleq \sum_{n \in \mathcal{N}} |n \cap c| \times w(n) \times cr(n, c) \tag{3}$$

Note that this quality function has been defined to facilitate the evaluation of a merging decision. The initial quality of a cluster c can be compared with its quality after a merging operation. However, it might not be so meaningful to compare the qualities of different clusters, because there might be variations in the characteristics (net weights, number of nets, etc.) of these clusters. Nevertheless, this does not pose a problem because, such a comparison is not required in the algorithm we propose. As will be explained in the next subsection, the vertices are visited in a random order, and the merging decisions are made in a local fashion.

It is also possible to extend Definition 4.2 to define the quality function for a subset of a cluster to facilitate comparison of the qualities of a cluster before and after merging.

Definition 4.3. Assume that we are given a cluster c and a set of vertices s such that $s \subseteq c$. The *quality* of *s in c* (denoted as $Q(s, c)$) is defined as:

$$Q(s, c) \triangleq \sum_{n \in \mathcal{N}} |n \cap s| \times w(n) \times cr(n, c) \tag{4}$$

Observe that $Q(s_i, c) + Q(s_j, c) = Q(s_i \cup s_j, c)$ for any two mutually disjoint subsets s_i and s_j of c. Based on these definitions, we can finally define the merge gain for two clusters as follows:

Definition 4.4. If clusters c_i and c_j are to be merged to obtain cluster c_{ij}, the *merge gain* $g(c_i, c_j)$ for c_i and c_j is defined as:

$$\begin{aligned} g(c_i, c_j) &\triangleq (Q(c_i, c_{ij}) - Q(c_i)) + (Q(c_j, c_{ij}) - Q(c_j)) && \text{(5a)} \\ &= Q(c_{ij}) - (Q(c_i) + Q(c_j)) && \text{(5b)} \end{aligned}$$

To examine the intuition behind this gain function, consider the first term in Eq. 5(a). It is in fact the weighted sum of the changes in net cluster co-relation values $cr(n, c_i)$ for all nets $n \in \mathcal{N}$ and $n \cap c_i \neq \emptyset$. As discussed before, the change in a $cr(n, c)$ value during a merging operation indicates both the similarity and dissimilarity between the merging clusters in terms of net n. So, if we take a weighted sum of these changes for all relevant nets, we can capture the overall similarity and dissimilarity between two clusters. For the proper weight function, we should consider the relative importance of each net n for a given cluster c. Here, if n spans more vertices in c and/or its weight is higher compared to other nets, we expect that the change in $cr(n, c)$ should be more important compared to other nets. This is in fact what Eq. 5(a) suggests, as also can be clearly observed from the reformulation of this equation:

$$\begin{aligned} g(c_i, c_j) = &\sum_{n \in \mathcal{N}} |n \cap c_i| \times w(n) \times (cr(n, c_{ij}) - cr(n, c_i)) \\ &+ \sum_{n \in \mathcal{N}} |n \cap c_j| \times w(n) \times (cr(n, c_{ij}) - cr(n, c_j)) \end{aligned} \tag{6}$$

4.2. *Multilevel clustering algorithm*

Based on the metrics we have defined in the previous subsection, we propose a bottom-up clustering algorithm, which is applied on hypergraph $\mathcal{H}_2$ to produce a set of initial clusters $\mathcal{C}_{\text{in}}$. In this algorithm, each vertex is treated as a singleton cluster in the beginning and such clusters are merged successively during the course of the algorithm. Since it is possible that some incorrect merging decisions are made, we also propose a refinement scheme that is incorporated between the merging operations.

The input of the algorithm is a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, and the output is a set of clusters $\mathcal{C}$. The pseudocode for the top level of the algorithm is as given in figure 3. Here, in the beginning of each level, the cluster set is initialized such that every cluster in $\mathcal{C}$ corresponds to a vertex in $\mathcal{V}$. Then the single level clustering procedure is executed, followed by the refinement procedure. At the end of each level, a new hypergraph is created such that each cluster in the old hypergraph corresponds to a single vertex in the new hypergraph. Note that during this conversion, the necessary information (e.g. the number of vertices in a cluster spanned by a net, etc.) is stored so that the calculations of the defined metrics give exactly the same values for the old and new hypergraphs. In our implementation, the termination condition occurs when no pair of clusters has been clustered in the previous level.

The single level clustering procedure is as shown in figure 4. Here, the aim is to increase the quality of each cluster by successively merging appropriate clusters. For this, the vertices are visited in a random order and the gain function given in Eq. (6) is used to determine the vertices to be merged.

```
MLCLUSTER($\mathcal{V}$,$\mathcal{N}$)
    level-no = 1
    while termination condition not occurred do
        $\mathcal{C}$ = Initialize-Clusters($\mathcal{V}$)
        $\mathcal{C}$ = SLCLUSTER($\mathcal{V}$,$\mathcal{N}$,$\mathcal{C}$)
        $\mathcal{C}$ = Refine($\mathcal{V}$,$\mathcal{N}$,$\mathcal{C}$)
        ($\mathcal{V}$,$\mathcal{N}$) = Create-Next-Level-Hypergraph($\mathcal{V}$,$\mathcal{N}$,$\mathcal{C}$)
        level-no = level-no + 1
    return $\mathcal{C}$
```

Figure 3. Pseudocode for multivel clustering.

```
SLCLUSTER($\mathcal{V}$, $\mathcal{N}$, $\mathcal{C}$)
    VisitOrder = Create-Random-Order(1,|$\mathcal{C}$|)
    for $i$ = 1 to |$\mathcal{C}$| do
        $m$ = VisitOrder[$i$]
        find $c_k$ such that g($c_m$,$c_k$) is maximum
        if $g(c_m, c_k)$ is positive then
            merge $c_m$ and $c_k$
    return $\mathcal{C}$
```

Figure 4. Pseudocode for single level clustering.

After each level of clustering, a refinement step is applied on the clusters. The reason for this step is that the merging decisions made early in the clustering are not necessarily the right ones. The approach we use here is based on extracting individual vertices from their clusters and assigning them to the clusters that seem best at that time. In fact this procedure is similar to the one given in figure 4. However, here instead of clusters, their constituent vertices are visited in a random order. Each visited vertex is extracted from its cluster, and it is treated as a singleton cluster for a remerging decision. Here, the same operations used on a visited cluster in figure 4 are applied on this singleton cluster.

In fact, this refinement scheme is based on the assumption that during clustering operations, it is more difficult for a poor cluster to grow in size than a good cluster.[1] This assumption is in fact due to the merging criterion we use: two clusters are merged only if the similarities between them are high enough and the dissimilarities are low enough. Since a poor cluster contains vertices from different natural clusters, it is expected to be dissimilar to most of the other clusters. So it is more difficult for a poor cluster to be merged.

As an example, consider the hypergraph with eight vertices v_1–v_8 and seven equal-weighted nets n_1–n_7 shown in figure 5. Here the natural clusters that we want to discover are shown with dashed curves. In the beginning, one cluster for each vertex is created and they are visited in a random order. Assume that the first visited cluster is the one corresponding to v_4. In this case, the merge gain will be maximum if the cluster corresponding to v_5 is selected. So the vertices corresponding to different clusters are merged together and form

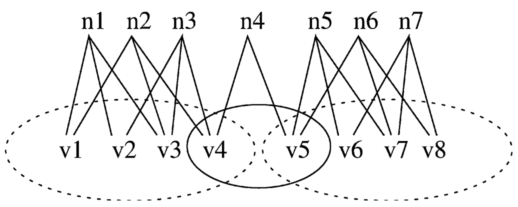


Figure 5. An incorrect merging decision is made if v_4 is visited first. Solid curve shows the cluster formed after this merging. Dashed curves show the natural clusters.

a poor cluster. Whatever the remaining visit order is, this cluster will not be merged with another one. The reason is that it is dissimilar to v_1, v_2, and v_3 because it contains v_5; and it is dissimilar to v_6, v_7 and v_8, because it contains v_4. So, the group v_1, v_2, v_3 and the group v_6, v_7, v_8 are merged next. In the refinement step, when v_4 and v_5 are visited, they leave their current clusters and move to the ones that they really belong to, that is v_4 moves to the cluster containing v_1, v_2, and v_3; and v_5 moves to the cluster containing v_6, v_7 and v_8. Note that in the early phases of the clustering, the most appropriate vertex to be merged with v_4 seemed to be v_5. The main reason here was that none of the initial clusters had been reflecting the characteristics of the natural cluster that v_4 belonged to before v_1, v_2 and v_3 were merged together.

For complexity analysis, consider one level of the clustering algorithm, which is given in figure 4. Here, each cluster $c_m \in \mathcal{C}$ is visited in a random order, and the gain function $g(c_m, c_j)$ is computed for all relevant clusters c_j. Observe that the merging operation here is performed only if this gain function is positive. So it is not required to consider the clusters that share no nets with cluster c_m, because they will definitely give nonpositive gains, according to Definition 4.4. To compute the relevant gain functions for the visited cluster c_m, each net $n \in \mathcal{N}$ for which $|n \cap c_m| \neq 0$ is processed. Here, processing a net n corresponds to computing its contribution to the gain function $g(c_m, c_j)$ for each cluster c_j for which $|n \cap c_j| \neq 0$. Let s_n, *size of net n*, denote the number of vertices connected to n, i.e., $s_n = |n|$. In the first level of the multilevel algorithm, each cluster in $\mathcal{C}$ corresponds to a single vertex in $\mathcal{V}$. So, the number of operations performed when net n is processed is $O(s_n)$ in the worst case. Since each cluster is visited only once in one level, the number of times net n is processed is $O(s_n)$. Hence, the total complexity of computing gain functions for all vertices in one level becomes $O(\sum_{n \in \mathcal{N}} {s_n}^2)$. Note that the merge operations in figure 4 are performed using disjoint set operations, the complexity of which are dominated by the complexity of gain function computations. For practical purposes, we can consider the number of levels in figure 3 as constant. As a result, the worst case complexity of this multilevel algorithm can be considered to be $O(\sum_{n \in \mathcal{N}} {s_n}^2)$. Observe that the relation summarization technique proposed in Section 3.2 effectively reduces the execution time of this algorithm through reducing the sizes and the number of nets of the input hypergraph.

4.3. Inter-cluster relation summarization

The metrics used in the multilevel algorithm to obtain the set of initial clusters $\mathcal{C}_{\text{in}}$ dictate that the cluster sizes are not too large compared to the sizes of individual nets. This is

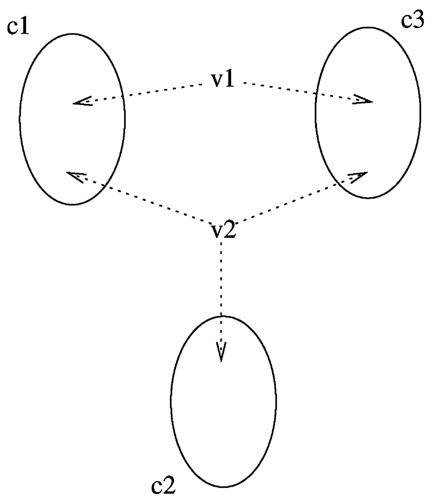


Figure 6. Affinities of vertices v_1 and v_2 defining relations among clusters.

because, as the cluster sizes increase far beyond the sizes of the nets, there will be more and more nets that induce dissimilarity between clusters. For instance, an internal net of a cluster will always indicate dissimilarity for the further merging decisions. So the cluster set $\mathcal{C}_{\text{in}}$ is expected to contain clusters that contain only strongly related vertices. Here, we give a method to model the relations among these clusters by a more sparse hypergraph. Here, each cluster $c \in \mathcal{C}_{\text{in}}$ is represented as a vertex in the new hypergraph, and the relations are determined based on the vertices in $\mathcal{H}_2$.

Although each vertex in $\mathcal{H}_2$ has been assigned to a particular cluster, a vertex might still have *affinities* to other clusters. We say a vertex v has affinity to a cluster c if the quality of v in c (i.e. $Q(v, c)$ in Definition 4.3) is *high enough*. Consider figure 6 as an example. Here, the clusters that vertices v_1 and v_2 belong to are not specified, but the affinities of them to the given clusters are shown with the dashed arrows. Observe that if a vertex has affinity to different clusters, we can infer that there should exist some relation among these clusters.

A user controllable threshold value is used to determine the affinities of vertices to clusters. For a vertex v, assume that c_m is the cluster for which $Q(v, c_m)$ is maximum. Then, vertex v is defined to have affinity to a cluster c_k if and only if $Q(v, c_k)/Q(v, c_m)$ is greater than the threshold value. It is possible to model the nets of the new hypergraph based on these affinities. The straightforward approach is to create a new net n for each vertex v in $\mathcal{H}_2$, such that n spans the vertices corresponding to the clusters that v is affine to. For the example of figure 6, the net created for v_2 would span the vertices corresponding to the clusters c_1, c_2 and c_3. Let us denote this hypergraph as $\mathcal{H}_3 = (\mathcal{C}_{\text{in}}, \mathcal{P}_{\mathcal{C}})$. Note that the number of nets in $\mathcal{H}_3$ is equal to the number of vertices in $\mathcal{H}_2$. Since our aim is to obtain a sparse hypergraph, it is required to decrease this number.

In fact, this problem is similar to the one in Section 3.2. In both, we have a set of data items with several relations among them, and our aim is to obtain a global summary of these relations. So, we use a similar method also here. Assume that a set of interesting relations $\mathcal{R}_\mathcal{C}$ is discovered for $\mathcal{D}_3 \equiv \mathcal{H}_3$, using an existing pattern discovery algorithm. We construct the new hypergraph $\mathcal{H}_4 = (\mathcal{V}_4, \mathcal{N}_4)$, by assigning a vertex for each cluster in $\mathcal{C}_{\text{in}}$ and a net for each relation in $\mathcal{R}_\mathcal{C}$. Then, for each net $n \in \mathcal{N}_4$, we assign a weight that is equal to $\mathit{procount}(r_n)/\sum_{v \in n} |c_v|$, where r_n is the relation corresponding to n and c_v is the cluster corresponding to v. Observe that the weight of each net is normalized by the sum of the sizes of relevant clusters. The intuition here can be explained by the help of an example. Assume that there exists a relation r over three clusters c_1, c_2 and c_3. If this relation is extremely strong, then all the vertices in these clusters would induce this relation. So the fraction of these vertices that actually incur this relation gives a measure of the strength of this relation.

4.4. Clustering sparse hypergraph

Since hypergraph $\mathcal{H}_4$ is expected to be sparse and well separated, the existing partitioning based metrics and algorithms can be used effectively. In our current implementation, we have used a straightforward approach that is based on the *minimum cutsize* metric (Lengauer, 1990), in which the aim is to minimize the number of *external* nets. Note that we say net n is *external* if all the vertices $v \in n$ do not belong to only one cluster. The algorithm we use here is greedy in nature. At every step, we merge the clusters that will cause the largest decrease in the cost function. Such merging operations goes on until no such operation is possible, or the user wants no more merging.

5. Experimental results

First, we have applied our clustering framework on two real life datasets to demonstrate the effectiveness of our models. After that, we have used a synthetic data generator to perform empirical comparisons with state-of-the-art clustering algorithm ROCK in terms of sensitivity and scalability characteristics.

In all our experiments, we have used a simple scheme based on discovering large itemsets to determine the sets $\mathcal{P}$, $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_\mathcal{C}$. For the relation sets $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_\mathcal{C}$, we have taken all the large itemsets discovered in the corresponding dataset $\mathcal{D}_1$ and $\mathcal{D}_3$, respectively. For the pattern set $\mathcal{P}$, we have eliminated the large itemsets with too small *weights* (as defined in Appendix A), assuming that they are not interesting. In our experiments, we have used the public license Apriori implementation by Christian Borgelt[2] to discover the large itemsets.

The minimum support parameter for the Apriori program is set according to the context and the input dataset. The method we have proposed in Appendix B to find the proper support values and our multilevel clustering algorithm require no parameters at all. To determine the *affinity* of a vertex to a cluster for creating the hypergraph of the second phase, we have allowed a 20% difference margin in the quality functions; that is we have set the threshold value mentioned in Section 4.3 to 0.8. The merging operations of the second phase did not require our intervention, and continued until no such operation is possible.

To evaluate the final clustering result, we have made one additional pass over the database to assign each transaction in the dataset to one of the clusters in C_f. Let $w(p)$ denote the weight value for pattern p as defined in Appendix A. To assign transaction t to a cluster, we have considered each pattern p that is *properly contained* in t and have selected the cluster c for which $\sum_{p \in c} w(p)$ is maximum.

We have performed our experiments on a 1.33 GHz AMD Athlon PC with 512 MB RAM, and Linux Operating System. The programs used for our framework have been written partly in C and C++ languages. For the comparisons we make in Section 5.2, we have used an implementation of ROCK in C language, supplied by Vipin Kumar and Micheal Steinbach.

5.1. Experiments on real life datasets

We have applied our clustering framework on two real life datasets. The first one is the Mushroom dataset from UCI Machine Learning Repository.[3] Here, each record in the dataset contains the attributes of a single mushroom. Although the mushrooms are from 23 different species, the class information provided is only in terms of being *edible* or *poisonous*. The number of data records is 8124, and the number of attributes is 22. We have used the approach given by Guha et al. (1999) to convert this dataset to a market basket type data. Then we have eliminated the items that occurred in more than 3000 transactions, because such items cause an exponential blow-up in the resource consumption of the Apriori algorithm (Bayardo et al., 2000). As a result, the new dataset contained 8124 transactions and 95 items.

The second dataset we use is Reuters-21578, version 1.0, text categorization test collection.[4] In this dataset, each document is a Reuters newswire story, and has been assigned some category information by a human indexer. We have eliminated the documents that belonged to no category or more than one category to obtain a set of 8654 single category documents. To convert this dataset to a market basket type data, we have made use of the term weighting scheme given by Salton et al. (1988). To apply it in our context, we have first eliminated the words that occurred too infrequently (in less than 50 documents) or too frequently (in more than 1000 documents). Then for each document, we have selected 15 words that give the highest value for the product of *term frequency* and *inverse document frequency* (Salton and Buckley, 1988). As a result, we have obtained a market basket data where each document corresponds to a transaction, and each selected word corresponds to an item. The number of transactions in this dataset is 8654, and the number of items is 16839.

For both datasets, the overall framework has been completed in less than 2 minutes. We have observed that discovering interesting patterns and relations were the dominating operations in terms of execution times.

5.1.1. Mushroom dataset. We have applied the Apriori algorithm on the original dataset with a minimum support threshold of 1%. Then we have eliminated the patterns that have *weights* (as defined in Appendix A) less than 0.3. So, we have obtained an interesting pattern set containing 999 elements. The minimum support parameters used to discover $\mathcal{R}_{\mathcal{P}}$ and $\mathcal{R}_{\mathcal{C}}$ have been set to 0.5%. In Table 1, the discovered clusters that have sizes greater than 30 are demonstrated. Here, 32 out of 36 clusters contain either completely edible, or completely poisonous mushrooms. Observe that the poisonous mushrooms have been

Table 1. Clustering results for Mushroom dataset.

Edible	Poisonous	Edible	Poisonous	Edible	Poisonous
32	0	32	0	42	0
54	0	48	0	44	0
56	0	30	0	96	0
48	0	48	0	0	276
96	0	30	0	38	0
180	0	184	0	0	1708
88	0	104	0	52	0
48	0	48	0	0	1280
48	192	304	0	908	15
140	0	56	256	48	0
48	0	48	0	72	0
220	0	72	0	48	35

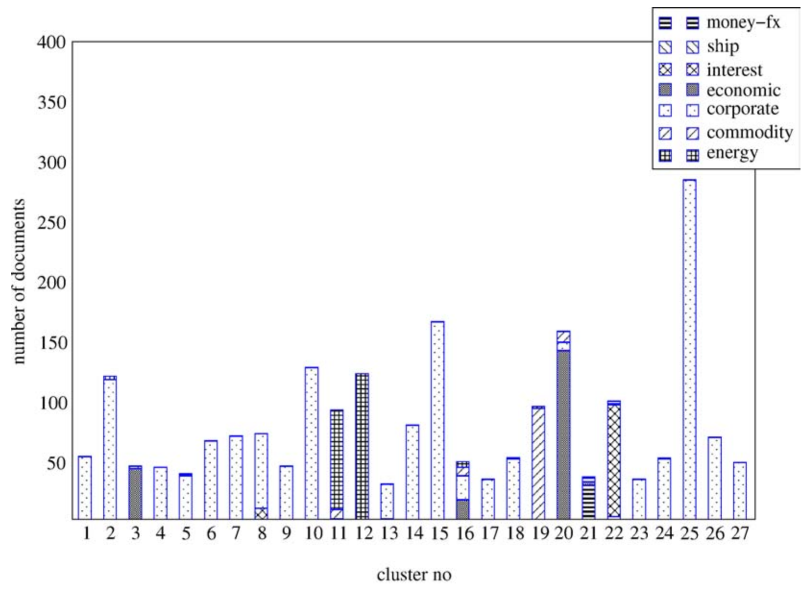
grouped in a small number of clusters with large sizes. The main reason is that the characteristics that cause a mushroom to be poisonous are more decisive compared to the edible characteristics.

5.1.2. Reuters dataset. The minimum support thresholds we have used for discovering the sets $\mathcal{P}$, $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_\mathcal{C}$ are 0.5%, 0.05%, and 2% respectively. While selecting the patterns, we have used a minimum *weight* threshold of 0.2. The results are demonstrated in figure 7. Here, each bar corresponds to one cluster, and illustrates the category distribution of the documents in that cluster. Note that only the clusters that contain more than 30 documents have been included in these figures. We also demonstrate some of the corresponding pattern clusters in figure 8.

In Section 1, we have argued that some items might have different meanings in different contexts. An example for this can be the word *adjust* in clusters 3 and 4 as seen in figure 8. When it occurs with *season*, a *corporate* category is inferred (i.e. cluster 3), and when it occurs with *split*, the category becomes *economy* (i.e. cluster 4). Also, consider cluster 29, which is characterized by only one pattern. Here, this pattern occurs due to the articles about the shortage of Sterling in England. Observe that none of these words are meaningful by themselves; but when they occur together, they define a cluster with category *money-fx* (i.e. money foreign exchange) as seen in the document clustering results in figure 7.

Observe that in the majority of the clusters in figure 7, only a single category dominates each document cluster. However, there also exist some poor clusters, in which documents from different categories exist. By observing the pattern clusters given in figure 8, it is possible to argue that some of these poor clusters in fact define new categories. For instance, clusters 16 and 45 contain documents from different categories (see figure 7), but all these documents are related to countries Canada and Japan respectively (see figure 8).

Another interesting point to mention is that there exist different clusters for the same categories as can be seen in figure 7. This is mainly due to different subcategories in a



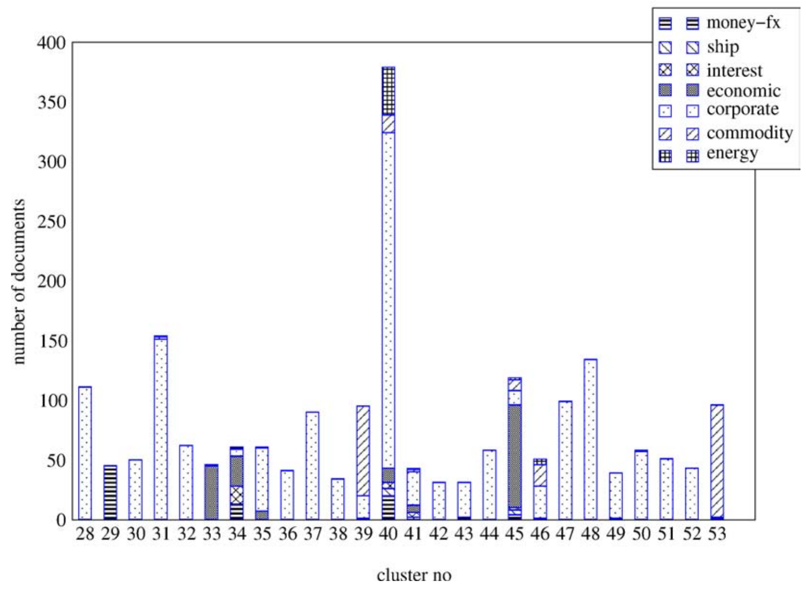


Figure 7. Document clustering results for Reuters dataset.

category. Consider for example clusters 19 and 53. Although both belong to the category *commodity*, one is about *coffee*, and the other is about *sugar* as seen in figure 8. Clusters 15, 25 and 31 can be another example for this. Again, all of them are from the same category *corporate*, but each one describes a different concept.

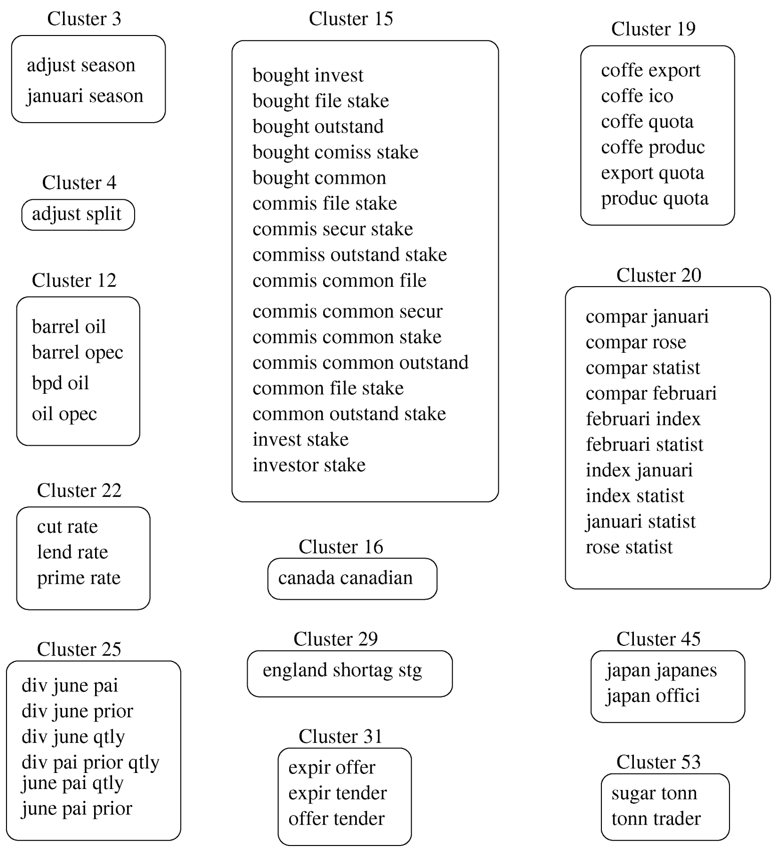


Figure 8. Some of the pattern clusters for Reuters dataset.

5.1.3. Discussions. At the end of the framework we have given, each transaction is assigned to one of the pattern clusters discovered. However, another application here could be to use these pattern clusters as new features for the transactions. For instance, Baker et al. (1998) have proposed a supervised feature clustering method for text categorization. Here, the *similar* words are grouped together and these groups are used as the new features of the documents. Obviously, such a clustering can be used only for the words that have context insensitive meanings (e.g. wheel, tire, throttle, honda, harley, etc.). To overcome this problem, we can extend this idea to the pattern clusters. Consider for example cluster 20 illustrated in figure 8. The words *compare*, *january*, *february*, *statistics*, *rose*, *index* may be meaningless when they are viewed independent of each other. However, if they occur in the patterns shown in the figure, they impose the category *economic*. Since all these patterns in fact describe the same concept, it should not matter which pattern is contained by a transaction. So, we can view each pattern cluster c as a feature f_c, and say a transaction t has the feature f_c if and only if t contains at least one of the patterns in c, according to Definition 3.1. Such an approach can be effectively used as a dimensionality reduction algorithm.

5.2. Experiments on synthetic data

We have also performed experiments on synthetic datasets to compare the effectiveness of our algorithm with ROCK. For this, we have implemented a synthetic data generator similar to the one that has been used by Guha et al. (1999) to perform experiments on ROCK. Here, we have started with a base case where the clusters are fairly well separated, and the transaction sizes are uniform. Then step by step, we have increased the inter-cluster similarities; increased the variations in transaction sizes; and decreased the intra-cluster similarities. For each case, we have compared the results of our algorithm with ROCK.

For the base case we have a dataset with 100000 transactions, and 5000 items. While a small portion of these items belong to a group of clusters, the majority of the items are outliers. For the base case there are 10 clusters, each containing 16 items. The 60 percent of the items in a cluster exclusively belong to that cluster. The remaining 40 percent is selected from a shared pool of items. For the purpose of creating significant inter-cluster similarities, we have kept the size of this shared pool much smaller than the actual number of items. Specifically, there are 48 items in this pool for the base case. In other words, 40 percent of the items in each cluster are selected randomly among these 48 items.

After the natural clusters in the dataset are specified, each transaction is randomly assigned to one of these clusters. A transaction that belongs to a cluster contains a small number of items from that cluster, along with other randomly selected items. Namely for the base case, each transaction contains 25 items, 4 of which are randomly selected from the items of the corresponding cluster, and the remaining 21 are randomly selected among all items. We can argue that this is close to the real life situation where a customer buys a couple of items from a cluster, along with a variety of other irrelevant items.[5] Also, such an approach results in weak intra-cluster similarities, so that we can compare the performance of the algorithms effectively.

Dataset 1 in Table 2 is the base case described above. To create *Dataset 2* mentioned in the same table, we have decreased the number of items in the shared pool from 48 to 12. That is, 40 percent of the items in each cluster are selected randomly among 12 items now, instead of 48. So, the similarities between different clusters (in terms of the common items contained) increases considerably in *Dataset 2*. For *Dataset 3* in Table 3, we have used the same parameters used for *Dataset 2*, but we allowed variations in the transaction sizes. Specifically, instead of setting the size of each transaction to 25, we have set it to a random

Table 2. Clustering results for synthetic dataset.

	Dataset 1			Dataset 2		
	% Clustered	% Error	Time (sec)	% Clustered	% Error	Time (sec)
Pattern Based Cl.	99.60	0.11	340	98.63	1.09	301
ROCK-sample	100.00	0.38	113	98.16	17.42	129
ROCK-fast label	100.00	2.07	522	100.00	20.26	533
ROCK-slow label	100.00	0.51	915	100.00	16.50	907

Table 3. Clustering results for synthetic dataset (cont'd).

	Dataset 3			Dataset 4		
	% Clustered	% Error	Time (sec)	% Clustered	% Error	Time (sec)
Pattern Based Cl.	98.58	1.05	319	90.98	4.61	274
ROCK-sample	91.26	32.96	202	94.34	63.32	234
ROCK-fast label	99.41	25.23	500	100.00	66.94	586
ROCK-slow label	99.98	20.67	881	98.35	67.77	885

number between 10 and 40. Finally for *Dataset 4*, we have decreased the ratio of exclusive items in each cluster from 60 percent to 40 percent, in addition to the changes described above. That is, 40 percent of the items in each cluster exclusively belong to that cluster, and the remaining items are selected from the shared pool of 12 items. It is obvious that such an approach decreases the intra-cluster similarities, while increasing the similarities between different clusters.

We have executed our algorithm and ROCK on these datasets, and reported the results in Tables 2 and 3. Here, we have given the results for three different executions of ROCK. First, we have performed random sampling on the given dataset to obtain a dataset of 5000 transactions. Then, we have executed ROCK on this small dataset, and reported the results on the second rows (i.e. *ROCK-sample*) of the tables. Note that only the transactions in the sample dataset are clustered here. The method given by Guha et al. (1999) is to label the transactions in the original dataset based on these clustering results. For this purpose, a set of representative transactions L_i is chosen for each cluster i. Then, each transaction T in the original dataset is assigned to cluster j such that T has the maximum normalized number of neighbors in L_j. In our experiments, we have used this approach to obtain the results given on the third and fourth rows (i.e. *ROCK-fast label* and *ROCK-slow label*) of the tables. Here, we report two different results for the labeling, because there is a runtime-quality tradeoff depending on the sizes of the chosen representative sets L_i. For the faster execution (i.e. third row of the table), we have chosen the representative set L_i as 20 percent of the transactions in cluster i. For better quality but slower execution (i.e. fourth row), we have set this ratio to 40 percent.

We needed two more parameters to execute ROCK on a sample dataset: k and θ. k is simply the number of clusters to be obtained as a result, whereas θ is the threshold for the similarity measure. More specifically, two transactions T_i and T_j are defined to be neighbors if and only if $\frac{|T_i \cap T_j|}{|T_i \cup T_j|} \geq \theta$. For the purpose of handling outliers effectively, it is suggested by Guha et al. (1999) that k should be set to a number larger than the actual number of natural clusters. In accordance with this, we have executed ROCK on each dataset by setting k to 10, 25, 50, and 75 in turns. At the same time, for each k value we have tried a different θ value in 0.01 increments to find the best (k, θ) combination for each dataset. It turned out that the best results for each dataset are obtained by using one of the following (k, θ) combinations: (25, 0.05), (25, 0.06), (50, 0.05), and (50, 0.06). Note that we have reported only the best results obtained for each dataset in Tables 2 and 3.

On the other hand, we needed to supply parameters for the pattern discovery algorithms we have used in our framework. Here, we have set the minimum support thresholds as 0.004, 0.0004, and 0.004 to discover the sets $\mathcal{P}$, $\mathcal{R}_{\mathcal{P}}$ and $\mathcal{R}_{\mathcal{C}}$, respectively.[6] The minimum confidence thresholds we have used for the corresponding datasets are 0.1, 0.1, and 0.4, respectively. We have used the same parameters to obtain all the results reported in the first rows of Tables 2 and 3.

Observe that the given tables illustrate the clustering results in terms of three different criteria. The first one (*% clustered* column) is the percentage of the transactions properly clustered. We define the properly clustered transactions for both algorithms as the ones belonging to clusters of sizes larger than 2 percent of the given dataset. The second criteria (*% error* column) is the percentage of the incorrectly clustered transactions. For instance, if a discovered cluster contains 2000 transactions from natural cluster A, 500 transactions from natural cluster B, and 200 transactions from natural cluster C, we define the number of incorrectly clustered transactions for this cluster as 700. Finally, the last criteria is the total execution time required to obtain the reported clustering results.

5.2.1. Discussions. The results demonstrate that performance of ROCK starts to degrade when the intra-cluster similarities are weakened, and the inter-cluster similarities are made denser. However, the framework we propose still gives relatively good results because of the existence of patterns defining the natural clusters. In other words, despite the fact that different clusters share a lot of items, it is the cooccurence of these items in the transactions that defines the natural clusters.

Actually, there are some similarities between ROCK and our algorithm in capturing global relationships in the dataset. For example, consider two transactions T_1 and T_2. According to the model used in ROCK, each transaction T_i that shares *enough* number of items with T_1, and *enough* number of items with T_2 will incur a relationship between T_1 and T_2, since T_i is defined to be a *common neighbor* for T_1 and T_2. However in our model, T_1 and T_2 are considered in terms of the *interesting* patterns that they contain. For simplicity, assume that T_1 contains only pattern p_1, and T_2 contains only p_2. Then each transaction T_i that contains both p_1 and p_2 (and hence that is a common neighbor for T_1 and T_2) will incur a relation among p_1 and p_2—but not among T_1 and T_2, as it is the case in ROCK.

Modeling relations among patterns instead of transactions has some advantages. First of all, our clustering algorithm does not directly depend on the actual transactions. For instance, we have shown in the experiments that our framework is not affected by the variations in the transaction sizes. However, since ROCK uses a static model for similarity modeling, we have observed some degradations by such variations. We can argue that a single θ value is not sufficient to model both the *dense* relationships (e.g. two transactions for which 4 out of 10 items are common), and the *weak* relationships (e.g. two transactions for which 4 out of 40 items are common) in the dataset.

Another advantage is that we don't need to make some unrealistic assumptions about the transactions, as long as we can use an effective pattern discovery algorithm for the given dataset. For instance in real life, a customer is unlikely to buy items from only one cluster. As an example, consider a customer who buys items for breakfast, and items for an infant. We can say that this customer belongs to two different clusters. If most of the

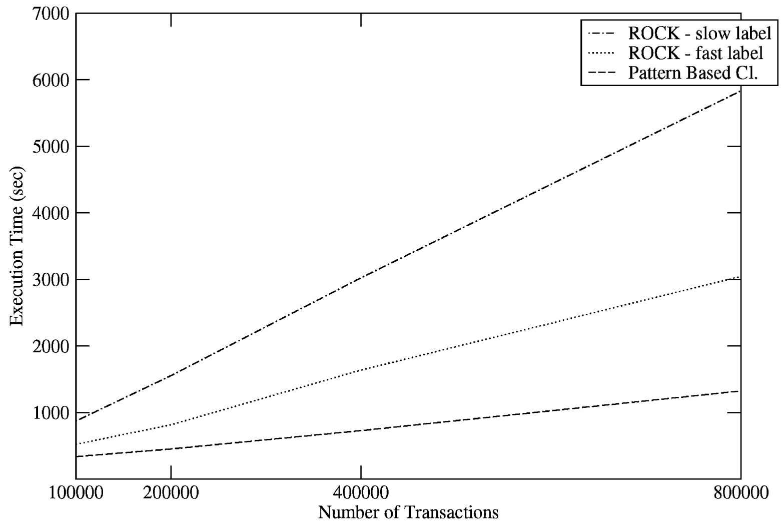


Figure 9. Scalability results for ROCK and our pattern based clustering algorithm.

transactions in the dataset are in this form, it becomes difficult to model relationships based on pairwise similarities between transactions. However, if the pattern discovery algorithm used can handle such cases effectively, our framework can still be used for clustering such a dataset.

5.2.2. Scalability analysis. As mentioned earlier, our framework uses the original dataset only to discover the interesting patterns, and to model the relations among these patterns. The actual clustering algorithm is performed on the hypergraph that represents these relations. In the end, a single scan is performed on the original dataset for the purpose of assigning each transaction to a cluster. So, we can say that scalability of our framework mainly depends on the scalability characteristics of the pattern discovery algorithms used. Since we have used Apriori in our implementations for this purpose, we can expect the overall framework to scale well with the size of the dataset.

Figure 9 gives a comparison of our framework with ROCK algorithm in terms of scalability. The dataset we have used in this experiment is the base dataset we have described in the previous subsections. The results show that both algorithms scale linearly with the size of the dataset. Note that ROCK performs the actual clustering on the sample dataset, of which size was kept constant in our experiments. The only operations performed on the original dataset by ROCK are sampling and labeling.

6. Conclusions

We have proposed a novel framework for the problem of clustering in large databases. The main idea here has been to view each transaction as a set of interesting data patterns. We did not focus on how to select such patterns, because it is a separate research subject by itself.

However, we have shown that even the simplest scheme that is based on support and confidence definitions gives meaningful results. The main reason for this is that, the clustering algorithm we have proposed handles some of the possible imperfections in selecting these patterns. For example, if a long pattern has not been selected, the patterns that are subsets of this pattern are clustered together to give the same effect. Or, if an irrelevant pattern is selected, it is left as a singleton, because it is *dissimilar* to the other patterns.

In the experiments, we have first used two real life datasets of which characteristics are quite different. Mushroom dataset is in fact a categorical data with a relatively small number of attributes, and the relations in this dataset are quite dense. Conversely, the relations in Reuters data are sparse, and over ten thousand items (i.e. words) are contained in the dataset. Despite this difference, the framework we have given has discovered meaningful clusters in both datasets. Also, our experiments on synthetic data demonstrated that the framework we propose performs better than state-of-the-art clustering algorithm ROCK especially when the clusters are not well separated.

In this work, several novel concepts were blended in a framework for the problem of clustering in large databases. However, these concepts can also be effectively used separately for other applications in different domains. For instance, the idea of representing each transaction as a set of interesting patterns can be used for modeling similarities between transactions in some other data mining algorithms that require similarity modeling. The relation summarization technique we have given to obtain a reduced hypergraph can find applications in the out-of-core hypergraph partitioning problem. The models and metrics we have proposed for merging decisions in initial clustering can also be exploited in the coarsening phases of other hypergraph partitioning algorithms. Similarly the refinement step embedded between levels and the sparsening model based on an initial clustering can be used with slight modifications in different multilevel bottom-up algorithms.

Appendix A: Pattern weighting

An association rule is defined as an implication of the form $A \Rightarrow B$, where A and B are disjoint itemsets. A is called the *antecedent* and B is called the *consequent* of the rule. The *support* of an association rule is the support of the itemset $(A \cup B)$. The *confidence* of the rule, denoted as $conf(A \Rightarrow B)$ is given as the ratio of $sup(A \cup B)$ to $sup(A)$. To assign a weight for a pattern, Han et al. (1997) used an approach based on confidence values. According to this, the weight of a pattern is the average confidence value of the association rules that include all the items in the corresponding pattern and that have a singleton as a consequent. For example, the weight of a pattern $\{a, b, c\}$ is equal to the average of confidence values of the following rules: $\{a, b\} \Rightarrow \{c\}$, $\{a, c\} \Rightarrow \{b\}$ and $\{b, c\} \Rightarrow \{a\}$. This weighting schema is in fact based on the existence of a pattern and its subsets. However, for the new semantics we have given in Definition 3.1, the absence of the supersets of a pattern should also be considered. For example, assume that all subsets and supersets of a pattern $\{a, b, c\}$ are as shown in figure 10. Assume further that each rational number on the arrow of the form $X \rightarrow Y$ is equal to the confidence value of the rule $X \Rightarrow Y$. Note that according to the approach given by Han et al. (1997), the weight of $\{a, b, c\}$ should be equal to the average value of the arrows entering it (i.e. 0.5). In our case, we should also

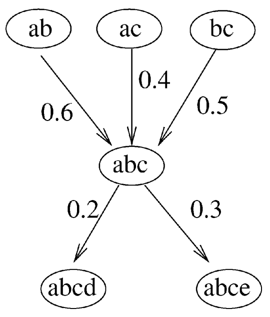


Figure 10. Assigning a weight to pattern {a, b, c}.

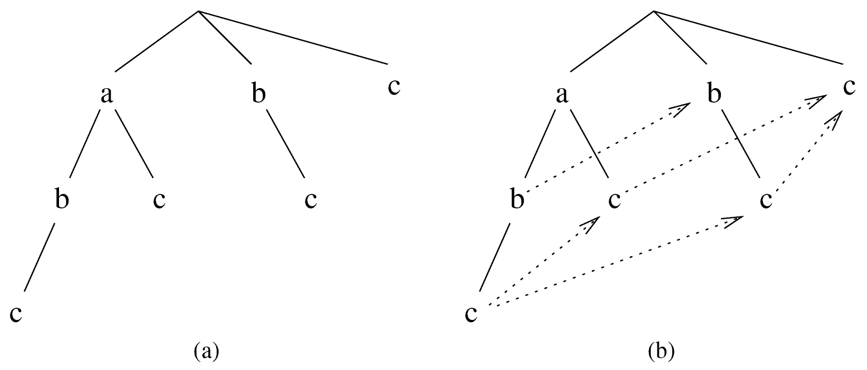


Figure 11. Trie data structures (a) used in DIC algorithm and (b) modified version for finding proper supports.

take into account the leaving arrows. An arrow leaving p and entering p_s with a confidence value f means that f fraction of the transactions t for which $p \subset t$ do not *properly contain* p because $p \subset p_s \subset t$. If there are too many such transactions for a pattern p, we can expect that p is not so important compared to its supersets. To incorporate such information into the weighting schema, we multiply the ratio $prosup(p)/sup(p)$ (see Definition 3.2 in Section 3.1 for *prosup*) with the weight function given above. For instance, in figure 10, the weight of $p = \{a, b, c\}$ becomes $0.5 \times prosup(p)/sup(p)$.

Appendix B: Finding proper support values

Given a dataset $\mathcal{D}$ and a set of patterns $\mathcal{P}$, we are to find the proper support value of each $p \in \mathcal{P}$. At first thought, it might seem that this can be accomplished by applying Sylvester's principle of inclusion and exclusion on the support values of the given patterns. Unfortunately, such a simple scheme is not guaranteed to work due to the possibility of absence of some patterns in $\mathcal{P}$. For instance, if $\mathcal{P} = \{\{a\}, \{a, b\}, \{a, c\}\}$, it is not possible to compute the proper support of $\{a\}$ using only the support values of these patterns, because the support of $\{a, b, c\}$ is not given. So it is required to make a pass over the data and count the proper occurrences of given patterns.

```
INCREMENT-COUNTERS(u, t, index)
    /* u is a node of the trie structure, t is a transaction */
    /* subptr and supptr refer to the links to subsets and supersets */
    for each i, $1 \leq i \leq u.depth - 1$ do
        if $(u.subptr[i]) \rightarrow lastsub \neq t.id$ then
            $(u.subptr[i]) \rightarrow count --$
            $(u.subptr[i]) \rightarrow lastsub = t.id$
    $u.count ++$
    for each i, $index \leq i \leq t.size$ do
        if u.supptr[i] exists then
            INCREMENT-COUNTERS(u.subptr[i], t, $i + 1$)
    return
```

Figure 12. Algorithm to increment the counts of patterns properly contained in a transaction.

For this aim, we represent the pattern set $\mathcal{P}$ by a trie structure similar to the one used in the DIC algorithm (Brin et al., 1997). Here, a node with depth d is created corresponding to each pattern with d items. The items in the patterns are assumed to be sorted according to some specific order. The root node in the trie represents the empty pattern, and each remaining node u represents a pattern $p \in \mathcal{P}$. The label of node u is the last item in the corresponding pattern p, and the labels of the nodes from root to u give the remaining items in p. Such a data structure corresponding to the pattern set $\mathcal{P} = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ is given in figure 11(a). The modification we have made to this structure can be seen in figure 11(b). Here, each node u in the tree has pointers to the nodes corresponding to the subsets of u that are exactly one level above of u.

After constructing this data structure, a pass over the entire dataset is made and for each transaction t_i, function INCREMENT-COUNTERS (figure 12) is called with the parameters: $u = root$, $t = t_i$ and $index = 1$. Note that this function is based on the corresponding *Increment* function in the DIC algorithm. The only difference is the for loop at the beginning. During the traversal of the nodes for a transaction t, if the node corresponding to a pattern (e.g. {a, b, c}) is reached, this means that the nodes corresponding to its subsets either have been reached (e.g. {a, b}) or will be reached later (e.g. {b, c}). So, the count values of such subsets should be decremented to compensate the increment performed when these nodes are reached. The *lastsub* field assures that the count of a node is decremented at most once for a transaction.

Compared to DIC, it seems that time and space complexity of this algorithm is higher. However, observe that this is only a postprocessing step, and the data structure used consists of only the already discovered patterns instead of all the candidate itemsets.

Acknowledgments

We thank Vipin Kumar and Micheal Steinbach for supplying the source codes of ROCK.

Notes

1. A cluster is said to be good if most of the vertices in that cluster are from only one natural cluster and poor if no such generalization can be made.
2. see http://fuzzy.cs.uni-magdeburg.de/ borgelt.
3. see http://www.ics.uci.edu/mlearn/MLRepository.html.
4. see http://www.research.att.com/ lewis/reuters21578.html.
5. Of course, the assumption that a transaction belongs to only one cluster is still unrealistic (i.e. a customer might buy sets of items from more than one cluster), however we will discuss this issue in the next subsection.
6. We have also performed some parameter analysis on *Dataset 4* (i.e. the most challenging dataset). We have observed that independently varying the first support value between 0.001–0.004, the second between 0.0001–0.0006, and the third between 0.003–0.006 had almost no effect on the results obtained. However, decreasing the support values further led to exponential blow-up in number of patterns, and the programs terminated due to insufficient memory. On the other hand, increasing them beyond these values resulted in not discovering most of the patterns, and the number of transactions properly clustered was decreased.

References

Aggarwal, C.C., Procopiuc, C., Wolf, J.L., Yu, P.S., and Park, J.S. 1999. Fast algorithms for projected clustering. In Proceedings of ACM SIGMOD Conference on Management of Data, pp. 61–72.

Agrawal, R., Imielinski, T., and Swami, A. 1993. Mining associations between sets of items in massive databases. In Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data, pp. 207–216.

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. In Proc. of the 20th Int'l Conference on Very Large Databases, pp. 487–498.

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. 1998. Automatic subspace clustering of high dimensional data for data mining applications. In Proceedings of ACM SIGMOD'98 International Conference on Management of Data, pp. 94–105.

Baker, D. and McCallum, A. 1998. Distributional clustering of words for text classification. In Proc. of SIGIR'98, Melbourne, pp. 96–103.

Bayardo, R.J. 1998. Efficiently mining long patterns from databases. In Proc. of the 1998 ACM-SIGMOD Int'l Conf. on Management of Data, pp. 85–93.

Bayardo, R.J., Agrawal, R., and Gunopulos, D. 2000. Constraint-based rule mining in large, dense databases. Data Mining and Knowledge Discovery, 4:217–240.

Berge, C. 1976. Graphs and Hypergraphs. American Elsevier.

Bradley, P.S., Fayyad, U., and Reina, C. 1998. Scaling clustering algorithms to large databases. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining. AAAI Press, pp. 9–15.

Brin, S., Motwani, R., Ullman, J., and Tsur, S. 1997. Dynamic itemset counting and implication rules for market basket data. In Proc. of the 1997 ACM-SIGMOD Int'l Conf. on the Management of Data, pp. 255–264.

Catalyurek, U. and Aykanat, C. 1999. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems, 10(7):673–693.

Cheng, C., Fu, A., and Zhang, Y. 1999. Entropy-based Subspace Clustering for Mining Numerical Data. KDD-99, San Diego, CA, pp. 84–93.

Aggarwal, C.C. and Yu, P.S. 2000 Finding generalized projected clusters in high dimensional spaces. SIGMOD Conference.

Chiu, T., Fang, D., Chen, J., and Wang, Y. 2001. A robust and scalable clustering algorithm for mixed type attributes in large database environments. KDD-2001, San Francisco, CA, pp. 263–268.

Faloutsos, C. and Lin, L-I. 1995. Fastmap: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. SIGMOD Conference, pp. 163–174.

Fasulo, D. 1999. An analysis of recent work on clustering algorithms. Available at http://www.cs.washington.edu/homes/dfasulo/clustering.ps.

Feldman, R., Aumann, Y., Amir, A., and Mannila, H. 1997. Efficient algorithms for discovering frequent sets in incremental databases. In 2nd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.

Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A., and French, J. 1999. Clustering large datasets in arbitrary metric spaces. In Proceedings of the 15th International Conference on Data Engineering, pp. 502–511.

Ganti, V., Gehrke, J., and Ramakrishnan, R. 1999. CACTUS—Clustering categorical data using summaries. In Proc. of the 5th Int'l Conf. on Knowledge Discovery and Data Mining (KDD-99), pp. 73–83.

Guha, S., Rastogi, R., and Shim, K. 1999. ROCK: A robust clustering algorithm for categorical attributes. In Proc. of the 15th Int'l Conf. on Data Eng.

Guha, S., Rastogi, R., and Shim, K. 1998. CURE: An efficient clustering algorithm for large databases. In Proc. of ACM SIGMOD Int'l Conference on Management of Data, pp. 73–84.

Han, E.H., Karypis, G., and Mobasher, B. 1997. Clustering in a high dimensional space using hypergraph models. In Technical Report, University of Minnesota, Department of Computer Science, pp. 97–019.

Huang, Z. 1997. A fast clustering algorithm to cluster very large categorical data sets in data mining. In Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery.

Jain, A.K., Murty, M.N., and Flynn, P.J. 1999. Data clustering: A review. ACM Computing Surveys, 31(3): 264–323.

Jolliffe, I.T. 1986. Principal Component Analysis. New York: Springer Verlag.

Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In Proceedings ACM/IEEE Design Automation Conference.

Karypis, G., Han, E., and Kumar, V. 1999. Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer, 32:68–75.

Kaufman, L. and Rousseeuw, P.J. 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, Inc.

King, B. 1967. Step-wise clustering procedures. J. Am. Stat. Assoc, 69:86–101.

Lengauer, T. 1990. Combinatorial Algorithms for Intergrated Circuit Layout. Chichester, U.K. Wiley.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symposium, 1:281–297.

Nagesh, H., Goil, S., and Choudhary, A. 1999. MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report 9906–010, Northwestern University, June.

Ng, R.T. and Han, J. 1994. Efficient and effective clustering methods for spatial data mining. In Proceedings of the Twentieth International Conference on Very Large Databases, pp. 144–155.

Park, J.S., Chen, M.S., and Yu, P.S. 1996. An effective hash based algorithm for mining association rules. In Proc. of the 1995 SIGMOD Conf. on the Management of Data, pp. 175–186.

Salton, G. and Buckley, C. 1988. Term weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513–523.

Silverstein, C., Brin, S., and Motwani, R. 1998. Beyond market baskets: Generalizing association rules to dependence rules. Knowledge Discovery and Data Mining, 2(1):39–68. Kluwer Academic Publishers.

Sneath, P.H.A. and Sokal, R.R. 1973. Numerical Taxonomy. London, UK: Freeman.

Voorhees, E.M. 1986. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. Information Processing & Management, 22(6):465–476.

Ward, J.H. 1963. Hierarchical grouping to optimize an objective function. J. Am. Stat. Assoc, 58:236–244.

Zait, M. and Messatfa, H. 1997. A comparative study of clustering methods. Future Generation Computer Systems, 13:149–159.

Zaki, M.J., Parthasarathy, S., Ogihara, M., and Li, W. 1997. New algorithms for fast discovery of association rules. In Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining, pp. 283–286.

Zhang, T., Ramakrishnan, R., and Livny, M. 1996. BIRCH: An efficient data clustering method for very large databases. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pp. 103–114.

Muhammet Mustafa Ozdal received his BS degree in Electrical Engineering, and MS degree in Computer Engineering, both from Bilkent University. He is currently a PhD student in Computer Science department of University of Illinois at Urbana Champaign. His research interests include algorithm design for VLSI physical design, data mining, and parallel computing.

Cevdet Aykanat received the PhD degree from The Ohio State University, Columbus, in electrical and computer engineering. He was a Fullbright scholar during his PhD studies. He worked at Intel, Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Computer Engineering Department of Bilkent University, where he is currently a professor. His research interests include parallel computing (especially for irregular and unstructured applications on distributed-memory architectures), parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph partitioning, load balancing, high-performance information retrieval and GIS systems, distributed databases, and grid computing. Some of his applied research projects have been funded by TUBITAK, Intel SSD, and European Commission DG III.