CLUSTERING SEQUENCES OF CATEGORICAL VALUES

Tadeusz MORZY, Marek WOJCIECHOWSKI, Maciej ZAKRZEWICZ*

Abstract. Conceptual clustering is a discovery process that groups a set of data in the way that the intra-cluster similarity is maximized and the inter-cluster similarity is minimized. Traditional clustering algorithms employ some measure of distance between data points in n-dimensional space. However, not all data types can be represented in a metric space, therefore no natural distance function is available for them. We address the problem of clustering sequences of categorical values. We present a measure of similarity for the sequences and an agglomerative hierarchical algorithm that uses frequent sequential patterns found in the database to efficiently generate the resulting clusters. The algorithm iteratively merges smaller, similar clusters into bigger ones until the requested number of clusters is reached.

1 Introduction

Clustering is one of the most popular data mining methods [5] [6] [8] [9] [10] [11] [13] [14] [22] [23] [27]. It consists in discovering interesting data distributions and patterns in very large databases. The problem of clustering is to partition a set of *k* data objects into *n* clusters such that the data objects within a cluster are closer (more similar) to each other than data objects in different clusters. Clustering is often used for market segmentation, in which the customers are divided into groups based on the similarity of their characteristics. Market segmentation is commonly used in targeted marketing and advertising, where specific products are directed towards specific customer segments. Other applications of clustering cover catalog design, store layout, stock market segmentation, buying patterns prediction, etc.

* Poznan University of Technology, Institute of Computing Science, ul. Piotrowo 3a, 60-965 Poznan, Poland

Clustering techniques developed in literature usually partition points in d-dimensional metric space into n clusters $c_1, c_2, ..., c_n$ optimizing some distance-based criterion function. The most commonly used criterion is the square-error criterion defined as follows:

$$E = \sum_{i=1}^{n} \sum_{p \in c_i} \|p - m_i\|^2$$

where m_i is the mean of cluster c_i, p is a data point, and E is the square error. There are two major groups of clustering algorithms: partitional and agglomerative. Partitional algorithms start with a single cluster containing all data objects and iteratively divide it into smaller clusters, while agglomerative algorithms start with one-object clusters and iteratively merge them.

In this paper we address the problem of clustering event sequences, where each event is represented by a set of categorical values (values that cannot be naturally ordered by a metric, e.g. products purchased in a supermarket). We notice that this problem cannot be solved using traditional distance-based clustering methods because: 1. the sequences are variable-length, 2. the sequences cannot be represented in a d-dimensional metric space, and 3. no natural distance function is available. Algorithms for clustering categorical data that have been proposed recently are not suitable either, as they cannot take advantage of sequential dependencies in data. Moreover, most of them do not provide descriptions for discovered clusters, which we believe is very important. Although similarity measures and clustering algorithms were proposed for other kinds of sequential data, such as time series or strings, they are not applicable for event sequences, which have a significantly different nature.

Consider a database of event sequences, where each event is described by a set of categorical values. An example of such database is a customer purchase history given in Figure 1a with its transformed form given in Figure 1b, where an ordered set of purchased products is stored for each customer. Assume that the problem is to cluster the sequences into two clusters, containing the customers having similar purchase histories. The basic question here is: how to measure the similarity of two customer's sequences? It seems that e.g. the sequences *101* and *105* are similar since they both contain the same subsequence '*book* $\rightarrow$ *c_disk'*. But what can we say about the similarity of the sequences e.g. *102* and *103*? We argue that these two sequences also can be considered similar, since there is a sequence (*104*) which contains both the subsequence that is common to *102* ('*bicycle* $\rightarrow$ *b_ball'*) and the subsequence that is common to *103* ('*tv_set* $\rightarrow$ *vcr* $\rightarrow$ *cassette'*). In general, we assume that two sequences are similar if either they contain the identical subsequences, or there exists a *connecting path* (Figure 1c) through a set of other sequences. In our example, the best solution would be to put the customers *101* and *105* into one cluster and the customers *102*, *103*, *104* into the other. Notice that the similarity between two sequences always depends on the presence of other sequences contributing to connecting paths. Because of that, we do not formalize the similarity measure between two sequences. Instead, we directly introduce a criterion function that represents the quality of clustering. A cluster in our approach is a set of data sequences such that within the set several subsequences occur much more frequently than outside the set. Quality of clustering depends on how frequently subsequences which are typical for a given cluster occur in other clusters. We do not consider all possible subsequences that can be extracted from the database,

because their number is likely to be very large. We concentrate on subsequences that frequently occur in the database, called sequential patterns [3] [25]. There exist a number of fast algorithms for discovering sequential patterns and we can use any of them as the preprocessing step.

We propose a heuristic algorithm for discovering an arbitrary number of possibly overlapping clusters that hold the customers, whose behavior is similar to each other. We refer to our clustering method as to *partial* clustering, because we allow the customers who are not similar to any other not to be covered by any cluster, and we allow a customer to belong to more than one cluster. We believe that for clustering customers based on their behavior, hard membership algorithms leading to disjoint clusters are not suitable, because a given client may follow buying patterns typical for several classes of customers. It is even possible that a single customer sequence may present products bought by different family members sharing one frequent buyer's card. The goal of our algorithm is not only to divide the customers' sequences into clusters, but also to develop a clustering model that can be used in the future to classify unknown sequences. In our model a single sequence may belong to several clusters.

Our clustering algorithm is agglomerative in nature. It starts with a number of small clusters and merges them together to reach the given number of resulting clusters. In the initial set of clusters, each cluster corresponds to one frequent pattern and contains all sequences supporting it. Clusters are iteratively merged according to the set similarity measure called Jaccard coefficient [15] applied to their contents.

time	cust_id	item
03/18	101	lamp
03/18	104	bicycle
03/19	102	t_rocket
03/21	101	l_bulb
03/21	102	bicycle
03/27	101	pillow
04/01	103	tv_set
04/03	104	tv_set
04/06	105	book
04/06	101	book
04/11	101	c_disk
04/15	105	c_disk
04/16	102	b_ball
04/17	102	s_bindings
04/17	103	vcr
04/17	104	b_ball
04/20	105	Dryer
04/22	103	c_phone
04/24	104	vcr
04/25	103	cassette
04/26	104	cassette
04/28	105	lamp
04/29	105	pillow
04/30	105	d_washer

Fig. 1a. Example database

cust_id	sequence
101	lamp → l_bulb → pillow → book → c_disk
102	t_rocket → bicycle → b_ball → s_bindings
103	tv_set → vcr → c_phone → cassette
104	bicycle → tv_set → b_ball → vcr → cassette
105	book → c_disk → dryer → lamp → pillow → d_washer

Fig. 1b. Transformed form of the example database

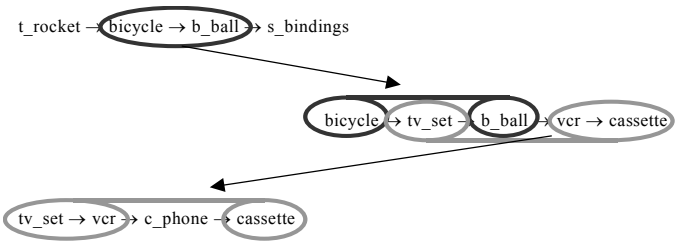

Fig. 1c. The idea of *connecting path*.

The paper is organized as follows. In Section 2, we present related work in the area. The basic definitions and the formulation of the problem are given in Section 3. Section 4 contains the problem decomposition and the description of the algorithm for

pattern-oriented clustering. The idea behind our algorithm is illustrated by a detailed example. In Section 5 we present our experimental results on synthetic data. We conclude with a summary and directions for future work in Section 6.

2 Related work

Many clustering algorithms have been proposed in the area of machine learning [7] [12] [15] and statistics [18]. Those traditional algorithms group the data based on some measure of similarity or distance between data points. They are suitable for clustering data sets that can be easily transformed into sets of points in n-dimensional space. Most of these algorithms are effective when the dimensionality of the space (the number of variables describing objects) is relatively small. Traditional algorithms also have problems with clusters having non-spherical shapes. They were not optimized for large databases and are not appropriate for categorical data.

In recent years, a number of clustering algorithms for large databases has been proposed. In [22], a clustering method based on randomized search, called CLARANS has been introduced. CLARANS was dedicated to solve problems of data mining in spatial databases. The problem of clustering in large spatial databases was also addressed in [5] and [6]. In [27], the authors presented a clustering method named BIRCH, which required a little more than one scan of the data. In [10], the hierarchical clustering algorithm named CURE was presented. The algorithm was designed for identifying clusters having non-spherical shapes.

Recently, several clustering algorithms for categorical data have been proposed. In [13] and [14], a method for hypergraph-based clustering of transaction data in a high dimensional space has been presented. The method used frequent itemsets to cluster items. Discovered clusters of items were then used to cluster customer transactions. In [23], a method for clustering data without distance functions was considered, and the proposed algorithm tried to group together records that had frequently co-occurring items. The implementation of the algorithm was similar to the one from [13] and [14].

In [9] a novel approach to clustering collections of sets and its application to the analysis and mining of categorical data was described. The proposed algorithm facilitated a type of similarity measure arising from the co-occurrence of values in the data set. This similarity measure was transitive in such a way that if A and B are similar and so are B and C there is said to be a weak type of similarity between A and C. This approach was applied to analysis of sequential data by looking for co-occurrences of events close in time, regardless of their ordering. In [11] an agglomerative hierarchical clustering algorithm called ROCK was introduced. The algorithm heuristically optimizes a criterion function based on links between tuples. The number of links between tuples depends on the number of common neighbors. In [8] an algorithm named CACTUS was presented together with the definition of a cluster for categorical data which is the generalization of the definition for numerical

data. The algorithm is fast and scalable. In contrast with the previous approaches to clustering categorical data, CACTUS gives formal descriptions of discovered clusters.

Many similarity measures that can be applied in clustering algorithms were presented for time series (see e.g. [1]). Similarity between two time series depends on the distances between corresponding elements, which are numbers. Elements of event sequences are sets of categorical values for which no distance function is available.

Another type of sequences, whose similarity has been extensively studied in literature, are strings (see e.g. [26]). A widely used notion of string similarity is the edit distance, which is expressed as the minimum number of insertions, deletions, and substitutions to transform one string into the other. We believe that the edit distance is not a good similarity measure for event sequences, because event sequences are likely to contain elements that correspond to very rare actions, which should be treated as noise. We also claim that sequence elements that are different but seem to be related, because they frequently co-occur together, should contribute to the similarity measure between sequences. The edit distance measure cannot take that into account.

Significant research has been done in the area of clustering of multivariate data with dynamic behavior using hidden Markov models [24]. However, Markov chains are not applicable in our problem, because they focus on direct precedence of events only.

Sequential dependencies are also taken into account in the problem of protein clustering [17], but the similarity measures considered there are based on common contiguous subsequences. We take into account only the ordering between events, not considering their distance in time.

The most similar approach to ours is probably the approach to document clustering proposed in [4]. The most significant difference between their similarity measure and ours is that we look for the occurrence of variable-length subsequences and concentrate only on frequent ones.

The problem of clustering sequences of complex objects was addressed in [16]. The clustering method presented there used class hierarchies discovered for objects forming sequences in the process of clustering sequences seen as complex objects. The approach assumed applying some traditional clustering algorithm to discover classes of sub-objects, which makes it suitable for sequences of objects described by numerical values, e.g. trajectories of moving objects.

Most of the research on sequences of events concentrated on the discovery of frequently occurring patterns. The problem of mining frequent patterns in a set of data sequences was first introduced in [3] and three different algorithms were given. The class of patterns considered there, called sequential patterns, had a form of sequences of sets of items. The statistical significance of a pattern (called support) was measured as a percentage of data sequences containing the pattern. In [25], the problem was generalized by adding taxonomy (is-a hierarchy) on items and various time constraints.

Another approach to the problem of mining frequent patterns in event sequences was presented in [21], where discovered patterns (called episodes) could have different type of ordering: full (serial episodes), none (parallel episodes) or partial and had to appear within a user-defined time window. The episodes were mined over a single event sequence and their statistical significance was measured as a percentage of windows containing the episode (frequency) or as a number of occurrences. In [20],

the model was extended to handle episodes described by a set of unary and binary predicates on event attributes.

In [19] an interesting approach to sequence classification was presented. In the approach, sequential patterns were used as features describing objects and standard classification algorithms were applied. To reduce the number of features used in the classification process, only distinctive (correlated with one class) patterns were taken into account.

3 Problem formulation

In this section, we formulate the problem of clustering of event sequences.

Definition 3.1. Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. A *sequence* $S = <X_1 X_2 ... X_n>$ is an ordered list of sets of items such that each set of items $X_i \subseteq L$. Let the database D be a set of sequences.

Definition 3.2. We say that the sequence $S_1 = <Y_1 Y_2 ... Y_m>$ *supports* the sequence $S_2 = <X_1 X_2 ... X_n>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We also say that the sequence S_2 is a *subsequence* of the sequence S_1 (denoted by $S_2 \subset S_1$).

Definition 3.3. A *frequent pattern* is a sequence that is supported by more than a user-defined minimum number of sequences in D. Let P be a set of all frequent patterns in D.

Definition 3.4. A *cluster* c is an ordered pair $<Q,S>$, where $Q \subseteq P$ and $S \subseteq D$, and S is a set of all database sequences supporting at least one pattern from Q. We call Q a *cluster description*, and S a *cluster content*. We use a dot notation to refer to a cluster description as to $c.Q$ and to a cluster content as to $c.S$.

In order to measure, how much a sequence matches a cluster, we introduce the *error function*. The function takes the minimum value of *0* when a sequence matches the cluster perfectly.

Definition 3.5. Given the set of clusters C, the *error function* $err(c,s)$ for a cluster c and a sequence s is defined as follows:

$$err(c,s) = \left|\{p \in c.Q : s \ \text{ does not support } \ p\}\right| + \left|\{p \in x.Q : x \in C \wedge x \neq c \wedge s \ \text{ supports } \ x\}\right|.$$

In other words, the error of placing a sequence s in a cluster c is equal to the number of patterns, which define c but are *not* supported by s, plus the number of patterns, which define the other clusters but *are* supported by s. The perfect match of a

sequence s and a cluster c occurs when s supports all patterns from c and none of the patterns from any other cluster.

Problem statement. Given a database $D = \{s_1, s_2, ..., s_k\}$ of data sequences, and a set $P = \{p_1, p_2, ..., p_m\}$ of frequent patterns in D, the problem is to divide P into a set of n clusters $c_1, c_2, \ldots, c_n$, such that $\underset{i,j,i\neq j}{\forall} c_i.Q \cap c_j.Q = \varnothing$, and the following error criterion:

$$E = \sum_{i=1}^{n} \sum_{s \in c_i.S} \left(err(c_i, s)\right)^2 .$$

is minimized.

4 Pattern-oriented hierarchical clustering

In this section, we describe a new clustering algorithm POPC for clustering large volumes of sequential data. The algorithm implements the general idea of agglomerative hierarchical clustering. However, instead of starting with a set of clusters containing one data sequence each, our algorithm uses previously discovered frequent patterns and starts with clusters containing data sequences supporting the same frequent pattern. We assume that a set of frequent patterns has already been discovered.

Before we present the clustering algorithm itself, we provide a formal definition of a *union* of two clusters, which is the result of merging two clusters together. We also give a formula that serves as *inter-cluster similarity* used to determine the order in which clusters are merged.

Definition 4.1 A *union* c_{ab} of the two clusters c_a and c_b is defined as follows:

$$c_{ab} = union(c_a, c_b) = < c_a.Q \cup c_b.Q \, , \, c_a.S \cup c_b.S >$$

Definition 4.2 *Inter-cluster similarity* between two clusters c_a and c_b is a Jaccard coefficient applied to cluster contents:

$$f(c_a, c_b) = \frac{\left| c_a.S \cap c_b.S \right|}{\left| c_a.S \cup c_b.S \right|} .$$

The above similarity function reflects co-occurrence of patterns describing two clusters (the size of the intersection of the cluster contents represents the number of sequences containing at least one pattern from each of the two cluster descriptions). It returns values from the range of *<0;1>*, where the value of *1* means that the clusters are identical while the value of *0* means that the clusters exhibit no similarity at all.

The inter-cluster similarity measure was chosen so that it reduces the number of sequences supporting patterns associated with other clusters.

4.1 Algorithm POPC

The algorithm for partial clustering based on frequently occurring patterns is decomposed into two following phases:

- *Transformation Phase*, which prepares the database for effective similarity evaluations,
- *Merge Phase*, which iteratively reduces the number of clusters by merging the most similar ones.

4.1.1 Transformation phase

In this phase, the database is transformed into a pattern-oriented form, which is more suitable for evaluating unions and intersections of cluster contents (used in the Merge Phase). For each frequent pattern we keep an ordered list of data sequences supporting the pattern. Each data sequence is represented by its identifier, e.g. in our example of sequences corresponding to lists of products bought by clients of a supermarket, a sequence could be identified by a unique identifier of a client. Sequences that do not support any frequent pattern are ignored.

Each pattern, together with the list of sequences supporting it, constitutes a cluster whose description is a set that contains the pattern as its only element. The cluster's content is made up of a set of data sequences from the list.

The proposed database representation simplifies evaluation of inter-cluster similarities. There is no need to refer to the original database in subsequent phases of the algorithm. Moreover, the size of the transformed database reduces as clusters are being merged together. When the process is finished, the database contains the result of clustering (descriptions and contents of the discovered clusters).

4.1.2 Merge phase

Figure 2 presents the Merge Phase of the clustering algorithm. First, the m patterns are mapped into m clusters, forming an initial set of clusters C_1, where each cluster is described by exactly one pattern. In the next step, the similarity function values are evaluated for all possible combinations of clusters. The similarity values are stored in a form of a matrix M_1. Next, the algorithm iteratively merges together pairs of clusters according to their similarity values and cluster contents' sizes. In each iteration k, the

two most similar clusters $c_a, c_b \in C_k$ are determined, and replaced by a new cluster $c_{ab} = union(c_a, c_b)$. If there are several pairs of clusters having maximal similarity values, then the two clusters having the smallest contents are merged. The actual merging is done by the function called *cluster*, described in detail in Section 4.1.4. When the new cluster is created, the matrix containing similarity values has to be re-evaluated. This operation is performed by means of the function called *simeval*, described in Section 4.1.3.

The Merge Phase stops when the number of clusters reaches n (the required number of clusters) or when there is no such pair of clusters $c_a, c_b \in C_k$ whose similarity is greater than *0*. The latter condition implies that the algorithm may discover a larger number of clusters than requested by a user. In this case, the number of discovered clusters (as well as the fraction of the original database covered by them) depends on the number and strength of frequent patterns used for clustering. If the quality of clustering is unsatisfactory, the clustering should be repeated with a higher number of frequent patterns (a set of patterns satisfying a lower frequency threshold).

$C_1 = \{c_i: c_i.Q=\{p_i\}, c_i.S=\{s_j: s_j \in D \wedge s_j \textit{ supports } p_i\}\}$;
$M_1 = simeval(C_1, \varnothing)$;
k=1;
while $|C_k| > n$ and exist $c_a, c_b \in C_k$ such that $f(c_a, c_b) > 0$ ***do begin***
  $C_{k+1} = cluster(C_k, M_k)$;
  $M_{k+1} = simeval(C_{k+1}, M_k)$;
  k++;
end;
Answer $= C_k$;

Fig. 2. Merge phase

4.1.3 Similarity matrix evaluation: *simeval*

Similarity matrix M_l stores the values of the inter-cluster similarity function for all possible pairs of clusters in an l-th algorithm iteration. The cell $M_l(x,y)$ represents the similarity value for the clusters c_x and c_y from the cluster set C_l (see example in Figure 3). The function *simeval* computes the values of the similarity matrix M_{l+1}, using both the similarity matrix M_l and the current cluster contents. Notice that in all iterations except the first one, the similarity matrix need not be completely re-computed. Only the similarity values concerning the newly created cluster have to be evaluated. Due to diagonal symmetry of the similarity matrix, for k clusters, only $(k^2-k)/2$ similarity function values need to be computed before the first iteration, and only $(k-1)$ in the subsequent ones.

In each iteration, the size of the matrix decreases since two rows and two columns corresponding to the clusters merged to form a new one are removed and only one column and one row are added for a newly created cluster.

-	$f(c_2, c_1)$	$f(c_3, c_1)$
$f(c_1,c_2)$	-	$f(c_3, c_2)$
$f(c_1,c_3)$	$f(c_2, c_3)$	-

$f(c_1,c_2) = f(c_2, c_1)$
$f(c_1,c_3) = f(c_3, c_1)$
$f(c_2, c_3) = f(c_3, c_2)$

Fig. 3. Structure of the similarity matrix for three clusters

4.1.4 Cluster merging: *cluster*

In each iteration, the number of processed clusters decreases by one. The similarity-based merging is done by the function called *cluster*. The function *cluster* scans the similarity matrix and finds pairs of clusters, such that their similarity is maximal. If there are many pairs of clusters that reach the maximal similarity values, then the function *cluster* selects the one with the smallest size of the union of their contents. Notice that no access to the original database is required to perform this phase of the algorithm. The function *cluster* takes a set of clusters C_k as one of its parameters and returns a set of clusters C_{k+1} such that $C_{k+1} = (C_k \setminus \{c_a, c_b\}) \cup \{c_{ab}\}$, where $c_a, c_b \in C_k$ are clusters chosen for merging and $c_{ab} = union(c_a, c_b)$.

4.2 Sequence classification

Having discovered the clusters, we can use the clustering model, represented by cluster descriptions, to classify new event sequences. For each new event sequence, we look for frequent patterns supported by it. If a sequence supports a pattern from a cluster's description, it is assigned to that cluster. If a sequence supports patterns from descriptions of more than one cluster it is mapped to all of them. Taking into account the number of patterns in each cluster's description that are supported by a given sequence, we could compute membership probability of the sequence in each cluster (based on the error function *err*). For example, in this way we can perform market segmentation on a customer purchase history database, and later we can easily classify new customers into the discovered market segments (clusters).

4.3 Example

Consider a database of customer transactions shown in Figure 4. For each transaction, we keep the transaction's time, items bought in the transaction and a unique customer identifier. Let us assume that a user wants to cluster customers who

follow similar frequent buying patterns into three clusters. Figure 5 presents an alternative representation of the database, where an ordered set of purchased items is given for each customer. Figure 6 shows frequent sequential patterns discovered in the database from Figure 4 (with a support threshold of 25%). The clustering algorithm starts with the Transformation Phase, which results in the initial set of clusters shown in Figure 7.

Customer Id	Transaction Time	Items Bought
1	October 10 1998	10 60
1	December 10 1998	20 30
1	December 15 1998	40
1	February 19 1999	50
2	November 10 1998	40
2	November 21 1998	50
2	December 12 1998	10
2	January 18 1999	20 30 70
3	October 15 1998	40
3	November 29 1998	50
3	December 14 1998	10
3	January 22 1999	80
3	February 11 1999	20 30
4	December 20 1998	10
4	February 4 1999	20
5	February 12 1999	80
6	November 1 1998	10
6	November 22 1998	30 90
7	February 1 1999	20 30
8	October 10 1998	60
8	November 22 1998	100
9	January 12 1999	100
10	January 21 1999	90 100

Fig. 4. Database sorted by Customer ID and Transaction Time

ID	Customer sequence
1	< (10 60) (20 30) (40) (50) >
2	< (40) (50) (10) (20 30 70) >
3	< (40) (50) (10) (80) (20 30) >
4	< (10) (20) >
5	< (80) >
6	< (10) (30 90) >
7	< (20) (30) >
8	< (60) (100) >
9	< (100) >
10	< (90 100) >

Fig. 5. Customer-sequence representation of the database

Patterns with support > 25%	
p_1	< (10) (20 30) >
p_2	< (10) (20) >
p_3	< (10) (30) >
p_4	< (20 30) >
p_5	< (10) >
p_6	< (20) >
p_7	< (30) >
p_8	< (40) (50) >
p_9	< (40) >
p_{10}	< (50) >
p_{11}	< (100) >

Fig. 6. Pattern set used for clustering

Cluster	Description	Sequences
c_a	p_1	1, 2, 3
c_b	p_2	1, 2, 3, 4
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_e	p_5	1, 2, 3, 4, 6
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_h	p_8	1, 2, 3
c_i	p_9	1, 2, 3
c_j	p_{10}	1, 2, 3
c_k	p_{11}	8, 9, 10

Fig. 7. Pattern-oriented representation of the database

Before the first iteration of the Merge Phase the similarity matrix should be build. The similarity matrix for the initial set of clusters from Figure 7 is shown in Figure 8.

	c_a	c_b	c_c	c_d	c_e	c_f	c_g	c_h	c_i	c_j	c_k
c_a	x	0.75	0.75	0.75	0.6	0.6	0.6	1	1	1	0
c_b	0.75	x	0.6	0.6	0.8	0.8	0.5	0.75	0.75	0.75	0
c_c	0.75	0.6	x	0.6	0.8	0.5	0.8	0.75	0.75	0.75	0
c_d	0.75	0.6	0.6	x	0.5	0.8	0.8	0.75	0.75	0.75	0
c_e	0.6	0.8	0.8	0.5	x	0.66	0.66	0.6	0.6	0.6	0
c_f	0.6	0.8	0.5	0.8	0.66	x	0.66	0.6	0.6	0.6	0
c_g	0.6	0.5	0.8	0.8	0.66	0.66	x	0.6	0.6	0.6	0
c_h	1	0.75	0.75	0.75	0.6	0.6	0.6	x	1	1	0
c_i	1	0.75	0.75	0.75	0.6	0.6	0.6	1	x	1	0
c_j	1	0.75	0.75	0.75	0.6	0.6	0.6	1	1	x	0
c_k	0	0	0	0	0	0	0	0	0	0	x

Fig. 8. Initial similarity matrix

In the first iteration there are six pairs of clusters having maximal similarity (similarity = *1*): (c_a, c_h), (c_a, c_i), (c_a, c_j), (c_h, c_i), (c_h, c_j) and (c_i, c_j). Since all the six pairs have the same sum of cluster contents' sizes, any of them can be chosen for merging (the actual choice may depend on a particular implementation of the algorithm). In this example we assume that a pair of clusters first found during a scan of the similarity matrix (performed row after row, from left to right) is chosen in such situations. This leads to selecting (c_a, c_h) as the first pair of clusters to be merged. In the next two iterations clusters having similarity = *1* are merged to form c_{ahij}. The database and the similarity matrix after the third iteration are shown in Figure 9. In a practical implementation of the algorithm, it would be desirable to look for sets of clusters having similarity = *1* for each pair of clusters from a given set, because such groups of clusters could be merged in a single iteration. In our example, clusters c_a, c_h, c_i, and c_j could be merged at once.

In the fourth iteration, we merge the clusters c_b and c_e (see Figure 10). In the fifth iteration, the clusters c_{be} and c_c are merged to form the intermediate result presented in Figure 11. Then, the merging of the cluster clusters c_d and c_f in the sixth iteration leads to the state illustrated by Figure 12.

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_b	p_2	1, 2, 3, 4
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_e	p_5	1, 2, 3, 4, 6
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_b	c_c	c_d	c_e	c_f	c_g	c_k
c_{ahij}	x	0.75	0.75	0.75	0.6	0.6	0.6	0
c_b	0.75	x	0.6	0.6	0.8	0.8	0.5	0
c_c	0.75	0.6	x	0.6	0.8	0.5	0.8	0
c_d	0.75	0.6	0.6	x	0.5	0.8	0.8	0
c_e	0.6	0.8	0.8	0.5	x	0.66	0.66	0
c_f	0.6	0.8	0.5	0.8	0.66	x	0.66	0
c_g	0.6	0.5	0.8	0.8	0.66	0.66	x	0
c_k	0	0	0	0	0	0	0	x

Fig. 9. Database and similarity matrix after 3 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{be}	p_2, p_5	1, 2, 3, 4, 6
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{be}	c_c	c_d	c_f	c_g	c_k
c_{ahij}	x	0.6	0.75	0.75	0.6	0.6	0
c_{be}	0.6	x	0.8	0.5	0.66	0.66	0
c_c	0.75	0.8	x	0.6	0.5	0.8	0
c_d	0.75	0.5	0.6	x	0.8	0.8	0
c_f	0.6	0.66	0.5	0.8	x	0.66	0
c_g	0.6	0.66	0.8	0.8	0.66	x	0
c_k	0	0	0	0	0	0	x

Fig. 10. Database and similarity matrix after 4 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bce}	p_2, p_3, p_5	1, 2, 3, 4, 6
c_d	p_4	1, 2, 3, 7
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bce}	c_d	c_f	c_g	c_k
c_{ahij}	x	0.6	0.75	0.6	0.6	0
c_{bce}	0.6	x	0.5	0.66	0.66	0
c_d	0.75	0.5	x	0.8	0.8	0
c_f	0.6	0.66	0.8	x	0.66	0
c_g	0.6	0.66	0.8	0.66	x	0
c_k	0	0	0	0	0	x

Fig. 11. Database and similarity matrix after 5 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bce}	p_2, p_3, p_5	1, 2, 3, 4, 6
c_{df}	p_4, p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bce}	c_{df}	c_g	c_k
c_{ahij}	x	0.6	0.6	0.6	0
c_{bce}	0.6	x	0.66	0.66	0
c_{df}	0.6	0.66	x	0.66	0
c_g	0.6	0.66	0.66	x	0
c_k	0	0	0	0	x

Fig. 12. Database and similarity matrix after 6 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bcdef}	p_2, p_3, p_4, p_5, p_6	1, 2, 3, 4, 6, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bcdef}	c_g	c_k
c_{ahij}	x	0.5	0.6	0
c_{bcdef}	0.5	x	0.83	0
c_g	0.6	0.83	x	0
c_k	0	0	0	x

Fig. 13. Database and similarity matrix after 7 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bcdefg}	$p_2, p_3, p_4, p_6, p_5, p_7$	1, 2, 3, 4, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bcdefg}	c_k
c_{ahij}	x	0.5	0
c_{bcdefg}	0.5	x	0
c_k	0	0	x

Fig. 14. Database and similarity matrix after 8 iterations

The results of the seventh and eighth iterations are presented in Figures 13 and 14. After the eighth iteration, the requested number of clusters is reached and the Merge Phase ends. Figure 15 presents final clustering results.

Cluster	Description	Customer Sequences
1	p_1, p_8, p_9, p_{10}	1, 2, 3
2	p_2, p_3, p_4, p_6, p_5, p_7	1, 2, 3, 4, 6, 7
3	p_{11}	8, 9, 10

Fig. 15. Discovered clusters

The algorithm found three clusters, two of which overlap (the content of one of them even includes the content of the other, but their descriptions do not imply that). Data sequence of the customer *5* is not contained in any cluster because it did not support any frequent pattern, which is a consequence of the fact that the customer did not follow any typical buying pattern.

5 Experimental results

To assess the performance and results of our clustering algorithm, we performed several experiments on a PC with *Pentium II* 300 MHz processor and 128 MB of main memory, cooperating with *Oracle 8.1.5* relational database management system on another PC with *Pentium II* 300 MHz processor and 128 MB of main memory. Experimental data sets were created by synthetic data generator *GEN* from Quest project [2]. *GEN* generates textual data files containing sets of numerical items.

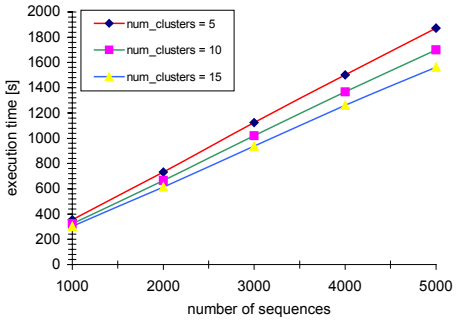

Fig. 16. Execution time for different database sizes

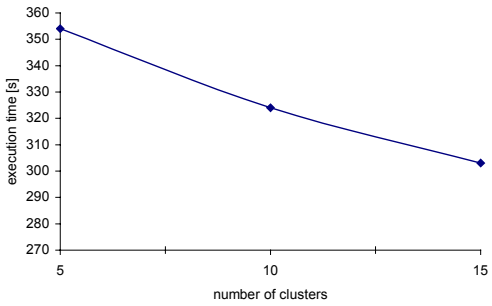

Fig. 17. Execution time for different number of resulting clusters

Figure 16 shows the performance of the clustering algorithm for different database sizes expressed as the number of sequences in the database. In the experiment, for all the database sizes the data distribution was the same, which resulted in the same set of patterns used for clustering. The support threshold for sequential patterns was set to

4%. The algorithm scales linearly with the number of source sequences, which makes it suitable for large databases. The key factor is that the number of frequent patterns (equal to the number of initial clusters in our approach) does not depend on the database size but on the data distribution only. The execution time depends linearly on the number of input sequences, because the number of sequences supporting a given frequent pattern (for the same support threshold) grows linearly as the number of sequences in the database increases.

Figure 17 illustrates the influence of the number of requested clusters on the execution time of our algorithm. In the experiment the database contained 1000 sequences and the support threshold for sequential patterns was set to 4%. We observe that the execution time depends almost linearly on the number of iterations of the algorithm (each iteration merges one pair of clusters). We could expect the execution time of subsequent iterations to decrease, because the number of clusters reduces. However, we should notice that the size of clusters in fact increases, and it compensates the above feature.

We have also evaluated the quality of results produced by our heuristic algorithm. Using our implementation of a combinatory algorithm to find the ideal clustering, we compared the ideal solution with the heuristic one and we computed the quantity of incorrectly clustered sequences. Due to NP-complexity of the combinatory algorithm we were able to perform the test for a small number of initial clusters only (10-15). However, the test showed that our heuristic algorithm correctly clusters c.a. 91% of database sequences, what seems to be a very good result.

6 Concluding remarks

We considered the problem of clustering sequential data in large databases. Due to the limitations of the existing clustering methods, we introduced the new algorithm, which uses frequent patterns to generate both clustering model and cluster contents. The algorithm iteratively merges smaller, similar clusters until the requested number of clusters is reached. In the absence of a well-defined metric space, we propose the inter-cluster similarity measure based on co-occurrence to be used in cluster merging.

An important feature of the algorithm is that it does not only divide the source sequences into clusters but also delivers a classification model that can be used to classify future data. Since the model is formed by a set of frequent patterns to be contained, the classification of a new sequence simply consists in checking if it contains any of the clusters' descriptions. If the new sequence contains patterns from different clusters, then it belongs to many clusters with different membership probabilities.

References

1. Agrawal R., Faloutsos C., Swami A.: Efficient Similarity Search in Sequence Databases. In: Lomet D.B., ed. *Foundations of Data Organization and Algorithms, 4th International Conf., Proceedings*. Springer, Chicago, Illinois (1993), 69-84.
2. Agrawal, R.; Mehta, M.; Shafer, J.; Srikant, R.; Arning, A.; Bollinger, T.: The Quest Data Mining System. In: Simoudis E., Han J., Fayyad U.M., eds. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Portland, Oregon (1996), 244-249.
3. Agrawal R., Srikant R.: Mining Sequential Patterns. In: Yu P.S., Chen A.L.P., eds. *Proceedings of the Eleventh International Conference on Data Engineering*.. IEEE Computer Society, Taipei, Taiwan (1995), 3-14.
4. Broder A., Glassman S., Manasse M., Zweig G.: Syntactic clustering of the Web. *Computer Networks and ISDN Systems* **29***, Proceedings of the Sixth International World Wide Web Conference* (1997), 1157-1166.
5. Ester M., Kriegel H-P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Simoudis E., Han J., Fayyad U.M., eds. *Proceedings of the 2nd International Conf. on Knowledge Discovery and Data Mining*. AAAI Press, Portland, Oregon (1996), 226-231.
6. Ester M., Kriegel H-P., Xu X.: A Database Interface for Clustering in Large Spatial Databases. In: Fayyad U.M., Uthurusamy R., eds. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Montreal, Canada (1995), 94-99.
7. Fisher D.H.: Knowledge acquisition via incremental conceptual clustering. *Machine Learning* **2** (1987) 139-172.
8. Ganti V., Gehrke J., Ramakrishnan R.: CACTUS-Clustering Categorical Data Using Summaries. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Diego, California (1999), 73-83.
9. Gibson D., Kleinberg J.M., Raghavan P.: Clustering Categorical Data: An Approach Based on Dynamical Systems. In: Gupta A., Shmueli O., Widom J., eds. *Proceedings of 24th International Conference on Very Large Data Bases*. Morgan Kaufmann, New York City, New York (1998), 311-322.
10. Guha S., Rastogi R., Shim K.: CURE: An Efficient Clustering Algorithm for Large Databases. In: Haas L.M., Tiwary A., eds. *Proceedings ACM SIGMOD International Conference on Management of Data*. ACM Press, Seattle, Washington (1998), 73-84.
11. Guha S., Rastogi R., Shim K.: ROCK: A Robust Clustering Algorithm for Categorical Attributes. In: *Proceedings of the 15th International Conference on Data Engineering*. IEEE Computer Society Press, Sydney, Australia (1999), 512-521.
12. Hartigan J.A.: *Clustering Algorithms*. John Wiley & Sons, New York, 1975.
13. Han E., Karypis G., Kumar V., Mobasher B.: Clustering based on association rules hypergraphs. In: *Proccedings of SIGMOD'97 Workshop on Research Issues in Data Mining and Knowledge Discovery*. (1997).

14. Han E., Karypis G., Kumar V., Mobasher B.: Hypergraph Based Clustering in High-Dimensional Data Sets: A summary of Results. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, **21** (1998), no. 1 15-22.
15. Jain A.K., Dubes R.C.: *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.
16. Ketterlin A.: Clustering Sequences of Complex Objects. In: Heckerman D., Mannila H., Pregibon D., eds. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Newport Beach, California (1997), 215-218.
17. Krogh A., Brown M., I. Mian S., Sjölander K., Haussler D.: Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal of Molecular Biology* **235** (1994), no. 5 1501-1531.
18. Kaufman L., Rousseeuw P.: *Finding Groups in Data*. John Wiley & Sons, New York, 1989.
19. Lesh N., Zaki M.J., Ogihara M.: Mining Features for Sequence Classification. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Diego, California (1999), 342-346.
20. Mannila H., Toivonen H.: Discovering generalized episodes using minimal occurrences. In: Simoudis E., Han J., Fayyad U.M., eds. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Portland, Oregon (1996), 146-151.
21. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. In: Fayyad U.M., Uthurusamy R., eds. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Montreal, Canada (1995), 210-215.
22. Ng R.T., Han J.: Efficient and effective clustering methods for spatial data mining. In: Bocca J.B., Jarke M., Zaniolo C., eds. *Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann, Santiago de Chile, Chile (1994), 144-155.
23. Ramkumar G. D., Swami A.: Clustering Data without Distance Functions. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **21** (1998) no. 1 9-14.
24. Smyth P.: Clustering sequences with hidden Markov models. In: Mozer M.C., Jordan M.I., Petsche T., eds. *Advances in Neural Information Processing* **9**. MIT Press (1997), 648.
25. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers P.M.G., Bouzeghoub M., Gardarin G., eds. *Advances in Database Technology, 5th International Conference on Extending Database Technology, Proceedings*. Springer, Avignon, France (1996), 3-17.
26. Sankoff D., Kruskal J.B., eds.: *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley, Reading, Massachusetts, 1983.
27. Zhang T., Ramakrishnan R., Livny M.: Birch: An efficient data clustering method for very large databases. In: Jagadish H.V., Mumick I.S., eds. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. ACM Press, Montreal, Canada (1996), 103-114.