Scalable Hierarchical Clustering Method for Sequences of Categorical Values[*]

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Tadeusz.Morzy@put.poznan.pl
Marek.Wojciechowski@cs.put.poznan.pl
Maciej.Zakrzewicz@cs.put.poznan.pl

Abstract. Data clustering methods have many applications in the area of data mining. Traditional clustering algorithms deal with quantitative or categorical *data points*. However, there exist many important databases that store *categorical data sequences*, where significant knowledge is hidden behind sequential dependencies between the data. In this paper we introduce a problem of clustering categorical data sequences and present an efficient scalable algorithm to solve the problem. Our algorithm implements the general idea of agglomerative hierarchical clustering and uses frequently occurring subsequences as features describing data sequences. The algorithm not only discovers a set of high quality clusters containing similar data sequences but also provides descriptions of the discovered clusters.

1 Introduction

Clustering is one of the most popular unsupervised data analysis methods that aims at identifying groups of similar objects based on the values of their attributes [14][15]. Many clustering techniques have been proposed in the area of machine learning [7][12][14] and statistics [15]. Those techniques can be classified as *partitional* and *hierarchical*. Partitional clustering obtains a partition of data objects into a given number of clusters optimizing some clustering criterion. Hierarchical clustering is a set of partitions forming a cluster hierarchy. An *agglomerative* hierarchical clustering starts with clusters containing single objects and then merges them until all objects are in the same cluster. In each iteration two most similar clusters are merged. *Divisive* hierarchical clustering starts with one cluster and iteratively divides it into smaller pieces.

Emerging data mining applications place additional requirements on clustering techniques, namely: scalability with database sizes, effective treatment of high dimensionality and interpretability of results [1]. Recently, the problem of data

[*] This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

clustering has been redefined in the data mining area. The concept of *cluster mining* [18] is used to represent a method which analyzes very large data sets to efficiently identify a small set of high-quality (statistically strong) clusters of data items. Cluster mining does not aim at partitioning of all the data items – instead, less frequent noise and outliers are ignored. In other words, cluster mining finds only the highest density areas hidden in the data space.

A number of efficient and scalable clustering algorithms for clustering *data points* represented by multidimensional quantitative values [1][6][10][22], as well as by sets of categorical values (product names, URLs, etc.) [8][9][11][13][19][21] have been proposed so far in the data mining area. However, we notice that there exist many important databases that store *categorical data sequences*, where significant knowledge is hidden behind sequential dependencies between the data (credit card usage history, operating system logs, database redo logs, web access paths, etc.). The existing clustering algorithms either cannot be easily transformed to deal with categorical data sequences or cannot take into account the sequential dependencies.

Categorical data sequence clustering can have many applications in the area of behavioral segmentation – e.g. *web users segmentation* [18]. The problem of web users segmentation is to use web access log files to partition a set of users into clusters such that the users within a cluster are more similar to each other than users from different clusters. The discovered clusters can then help in on-the-fly transformation of the web site content. In particular, web pages can be automatically linked by additional hyperlinks. The idea is to try to match an active user's access pattern with one or more of the clusters discovered from the web log files. Pages in the matched clusters that have not been explored by the user may serve as navigational hints for the user to follow.

In this paper we define the problem of clustering categorical data sequences and we propose an efficient algorithm to solve the problem. The algorithm employs the idea of *agglomerative hierarchical clustering*, which consists in merging pairs of similar clusters to form new larger clusters. We have taken the following assumptions: 1. the simplest cluster (elementary cluster) is a set of data sequences containing a common subsequence, 2. a significant cluster is a cluster that contains a large number of data sequences, 3. two clusters can be merged if a large number of their corresponding sequences overlap. Our algorithm starts with a set of significant elementary clusters and merges them iteratively until a user defined stop condition is satisfied.

Let us illustrate the approach with the following example (Fig. 1). We are given a web log file, which records paths used by users for navigation (e.g. the user s1 has visited the URLs: A, B, C, Z, and then D). Assume we are interested in discovering groups of users (sequences), whose behavior is similar to each other, i.e. who visit identical pages in the identical order. First, we create the elementary clusters c1, c2, c3, c4 that contain overlapping sequences. Then we notice that we can merge the clusters c1, c2, and c3 since the sequences they contain overlap between the clusters. Finally, the algorithm ends with two clusters, which represent web users of similar behavior.

s1: A->B->C->Z->D	s4: L->M->N->O
s2: P->Q->R	s5: E->F->G->A->B
s3: C->D->E->F	s6: Q->X->R ->S

c1:	c2:	c3:	c4:
s1: A->B->**C**->Z->**D**	s3: C->D->**E->F**	s1: **A->B**->C->Z->D	s2: P->**Q->R**
s3: **C->D**->E->F	s5: **E->F**->G->A->B	s5: E->F->G->**A->B**	s6: **Q**->X->**R** ->S

c1:	c4:
s1: **A->B**->**C**->Z->**D**	s2: P->**Q->R**
s3: **C->D**->**E->F**	s6: **Q**->X->**R** ->S
s5: **E->F**->G->**A->B**	

Fig. 1. Behavioral segmentation example

The paper is organized as follows. Section 2 discusses related work. In Section 3, the basic definitions and the formulation of the problem are given. Section 4 contains the problem decomposition and the description of the algorithm for pattern-oriented clustering. Experimental results concerning the proposed clustering method are presented in Section 5. We conclude with a summary in Section 6.

2 Related Work

Many clustering algorithms have been proposed in the area of machine learning [7][12][14] and statistics [15]. 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, which makes them inappropriate for categorical data.

Recently, several clustering algorithms for categorical data have been proposed. In [13] 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. [9] described a novel approach to clustering collections of sets, and its application to the analysis and mining of categorical data. The proposed algorithm facilitated a type of similarity measure arising from the co-occurrence of values in the data set. In [8] an algorithm named CACTUS was presented together with the definition of a cluster for categorical data. In contrast with the previous approaches to clustering categorical data, CACTUS gives formal descriptions of discovered clusters.

In [21] the authors replace pairwise similarity measures, which they believe are inappropriate for categorical data, with a clustering criterion based on the notion of large items. An efficient clustering algorithm based on the new clustering criterion is also proposed.

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.

The most similar approach to ours is probably the approach to document clustering proposed in [5]. 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.

Our clustering method can be seen as a scalable version of a traditional agglomerative clustering algorithm. Scaling other traditional clustering methods to large databases was addressed in [4], where the scalable version of the K-means algorithm was proposed.

Most of the research on sequences of categorical values concentrated on the discovery of frequently occurring patterns. The problem was introduced in [2] and then generalized in [20]. 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 [17] 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 Pattern-Oriented Agglomerative Hierarchical Clustering

Traditional agglomerative hierarchical clustering algorithms start by placing each object in its own cluster and then iteratively merge these atomic clusters until all objects are in a single cluster. Time complexity of typical implementations vary with the cube of the number of objects being clustered. Due to the poor scalability, traditional hierarchical clustering cannot be applied to large collections of data, such as databases of customer purchase histories or web server access logs. Another very important issue that has to be addressed when clustering large data sets is automatic generation of conceptual descriptions of discovered clusters. Such descriptions should summarize clusters' contents and have to be comprehensible to humans.

To improve performance of hierarchical clustering for large sets of sequential data, we do not handle sequences individually but operate on groups of sequences sharing a common subsequence. We concentrate only on the most frequently occurring subsequences, called frequent patterns (sequential patterns). We start with initial clusters associated with frequent patterns discovered in the database. Each of the initial clusters (clusters forming the leaf nodes of the hierarchy built by the clustering process) consists of sequences containing the pattern associated with the cluster. Clusters being results of merging of smaller clusters are described by sets of patterns and consist of sequences that contain at least one pattern from the describing set.

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* 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 description or a content of a given cluster (*c.Q* represents the description of a cluster *c* while *c.S* represents its content).

Definition 3.5. A cluster *c* is called an *elementary cluster* if and only if $| c.Q | = 1$.

Definition 3.6. A *union* c_{ab} of the two clusters c_a and c_b is defined as follows:
$c_{ab} = union(c_a, c_b) = < c_a.Q \cup c_b.Q \, , c_a.S \cup c_b.S >$.

Elementary clusters form leaves in the cluster hierarchy. The clustering algorithm starts with the set of all elementary clusters. Due to the above formulations, input sequences that do not support any frequent pattern will not be assigned to any elementary cluster and will not be included in the resulting clustering. Such sequences are treated as outliers. The above definitions also imply that in our approach clusters from different branches of the cluster hierarchy may overlap. We do not consider it to be a serious disadvantage since sometimes it is very difficult to assign a given object to exactly one cluster, especially when objects are described by categorical values. In fact, if two clusters overlap significantly, then it means that patterns describing one of the clusters occur frequently in sequences contained in the other cluster and vice versa. This means that such clusters are good candidates to be merged to form a new larger cluster. The measures of similarity between clusters we propose in the paper are based on this observation. The cluster similarity measures we consider in this paper are based on the co-occurrence of the frequent patterns.

Definition 3.7. The *co-occurrence* of two frequent patterns p_1 and p_2 is a Jaccard coefficient [14] applied to the sets of input sequences supporting the patterns:

$$co(p_1, p_2) = \frac{|\{ s_i \in D : s_i \supset p_1 \wedge s_i \supset p_2 \}|}{|\{ s_i \in D : s_i \supset p_1 \vee s_i \supset p_2 \}|} \, . \qquad \mathbf{(1)}$$

The similarity of two elementary clusters is simply the co-occurrence of patterns from their descriptions. The first of our inter-cluster similarity measures for arbitrary clusters is the extension of the above pattern co-occurrence measure i.e. similarity between two clusters c_a and c_b is a Jaccard coefficient applied to cluster contents:

$$f_J(c_a, c_b) = \frac{|c_a.S \cap c_b.S|}{|c_a.S \cup c_b.S|} \ . \qquad (2)$$

The second inter-cluster similarity measure we consider in the paper is defined as the average co-occurrence of pairs of patterns between two clusters' descriptions (group-average similarity):

$$f_{GA}(c_a, c_b) = avg(co(p_i, p_j)), where: p_i \in c_a.Q \wedge p_j \in c_b.Q) \ . \qquad (3)$$

Traditional hierarchical clustering builds a complete cluster hierarchy in a form of a tree. We add two stop conditions that can be specified by a user: the required number of clusters and the inter-cluster similarity threshold for two clusters to be merged. The first stop condition is suitable when a user wants to obtain the partitioning of the data set into the desired number of parts, the second is provided for cluster mining, which identifies high quality clusters.

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 build a cluster hierarchy starting from elementary clusters as leaves, iteratively merging the most similar clusters until the required number of clusters is reached or there are no pairs of clusters exceeding the specified similarity threshold.

4 Algorithms

In this section, we describe a new clustering algorithm POPC for clustering large volumes of sequential data (POPC stands for Pattern-Oriented Partial Clustering). The algorithm implements the general idea of agglomerative hierarchical clustering. As we mentioned before, 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 and we do not include the pattern discovery phase in our algorithm. The influence of the pattern discovery process on the overall performance of our clustering method is described in the next section.

The POPC algorithm is database-oriented. It assumes that the input data sequences and the contents of clusters to be discovered are stored on a hard disk, possibly managed by a standard DBMS. Only the structures whose size depends on the number of patterns used for clustering and not on the number of input data sequences are stored in the main memory. These structures are similarity and co-occurrence matrices and cluster descriptions.

We introduce two variants of our algorithm based on two different cluster similarity measures: POPC-J using the Jaccard coefficient of the clusters' contents, and POPC-GA using the group average of co-occurrences of patterns describing clusters. First we present the generic POPC algorithm and then we describe elements specific to particular variants of the algorithm.

4.1 Generic POPC Algorithm

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

- *Initialization Phase*, which creates the initial set of elementary clusters and computes the co-occurrence matrix between patterns which serves as the similarity matrix for the initial set of clusters,
- *Merge Phase*, which iteratively merges the most similar clusters.

4.1.1 Initialization Phase

In this phase, the initial set of clusters C_1 is created by mapping each frequent pattern into a cluster. During the sequential scan of the source database, the contents of initial clusters are build. At the same time the co-occurrence matrix for the frequent patterns is computed. This is the only scan of the source database required by our algorithm. (It is not the only place where access to disk is necessary because we assume that cluster contents are stored on disk too.) Figure 2 presents the Initialization Phase of the clustering algorithm.

```
$C_1$ = {$c_i$: $c_i$.Q={$p_i$}, $c_i$.S=∅};
UNION[][] = {0}; INTERSECT[][] = {0};
for each $s_j$∈D do
begin
  for each $p_i$∈P do
     if $s_j$ supports $p_i$ then
       $c_i$.S = $c_i$.S ∪ { $s_j$ };
     end if;
  for each $p_i$, $p_k$ ∈P do
    if $s_j$ supports $p_i$ or $s_j$ supports $p_k$ then
        UNION[i][k]++; UNION[k][i]++;
        if $s_j$ supports $p_i$ and $s_j$ supports $p_k$ then
           INTERSECT[i][k]++; INTERSECT[k][i]++;
       end if;
    end if;
end;
for each $p_i$, $p_k$ ∈P do
  CO[i][k] = INTERSECT[i][k] / UNION[i][k];
$M_1$ = CO;
```

Fig. 2. Initialization phase

To compute the pattern co-occurrence matrix *CO*, for each pair of patterns we maintain two counters to count the number of sequences supporting at least one of the patterns and both of the patterns respectively. Those counters are represented by temporary matrices *UNION* and *INTERSECT*, and are used to evaluate the coefficients in the matrix *CO* after the database scan is completed. The similarity matrix M_1 for the initial set of clusters C_1 is equal to the pattern co-occurrence matrix.

4.1.2 Merge Phase

Figure 3 presents the Merge Phase of the clustering algorithm. This phase of the algorithm iteratively merges together pairs of clusters according to their similarity values. In each iteration *k*, the two most similar clusters $c_a,c_b \in C_k$ are determined, and replaced by a new cluster $c_{ab} = union(c_a, c_b)$. 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.

```
k = 1;
while |$C_k$| > n and exist $c_a,c_b \in C_k$
such that f($c_a,c_b$) > min_sim 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. 3. Merge phase

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 *min_sim* (the similarity threshold).

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 the *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. The function *simeval* computes the values of the similarity matrix M_{l+1}, using both the similarity matrix M_l and the current set of clusters. Notice that in each iteration, 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-1)* similarity function values need to be computed in each iteration.

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.

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 that was found as first. 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 Algorithm Variants POPC-J and POPC-GA

The POPC-J version of the algorithm optimizes the usage of main memory. The pattern co-occurrence matrix is used only in the Initialization Phase as the initial cluster similarity matrix. It is not needed in the Merge Phase of the algorithm because the similarity function based on Jaccard coefficient does not refer to the co-occurrences of patterns describing clusters. Thus the co-occurrence matrix *CO* becomes the initial cluster similarity matrix M_1 and no copying is done, which reduces the amount of main memory used by the algorithm.

In case of the POPC-GA version of the algorithm the initial cluster similarity matrix M_1 is created as a copy of the pattern co-occurrence matrix *CO*, since the latter is used in each iteration of the Merge Phase to compute the similarity between the newly created cluster and the rest of clusters. The advantage of this version of the POPC algorithm is that it does not use clusters' contents to evaluate similarity between clusters in the Merge Phase (only clusters' descriptions and the pattern co-occurrence matrix are used). Due to this observation, the POPC-GA does not build clusters' contents while the Merge Phase progresses. This is a serious optimization as compared to the POPC-J algorithm which has to retrieve clusters' contents from disk to re-evaluate the similarity matrix. Contents of discovered clusters can be built in one step after the Merge Phase completes according to the following SQL query (using the clusters' descriptions maintained throughout the algorithm and sets of input sequences supporting given patterns, built in the Initialization Phase):

```
select distinct d.cluster_id, p.sequence_id
from CLUSTER_DESCRIPTIONS d , PATTERNS p
where d.pattern_id = p.pattern_id.
```

Each row in the CLUSTER_DESCRIPTIONS table contains information about the mapping of one pattern to the description of one cluster, while each row in the PATTERNS table contains information that a given data sequence supports a given pattern.

5 Experimental Results

To assess the performance and results of the clustering algorithm, we performed several experiments on a PC machine with Intel Celeron 266MHz processor and 96 MB of RAM. The data were stored in an Oracle8i database on the same PC machine. Experimental data sets were created by synthetic data generator *GEN* from Quest project [3].

First of all, we compared the sets of clusters generated by the two versions of the POPC algorithm. The difference was measured as a percentage of all pairs of sequences from all clusters discovered by one version of the algorithm that were not put into one cluster by the other version of the algorithm. This measure is asymmetric but we believe that it captures the difference between two results of clustering. We performed several tests on a small data set consisting of 200 input sequences, using about 100 frequent patterns. As a stop condition for both versions of the algorithm we chose the desired number of clusters ranging from 5 to 45. An average difference between clustering results generated by the two methods was less than 10%.

In the next experiment we compared the performance of the two POPC variants and tested their scalability with respect to the size of the database (expressed as a number of input sequences) and the number of frequent patterns used for clustering.

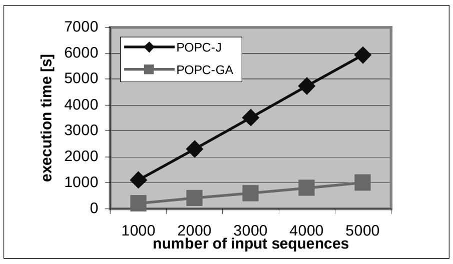

Fig. 4. Execution time for different database sizes

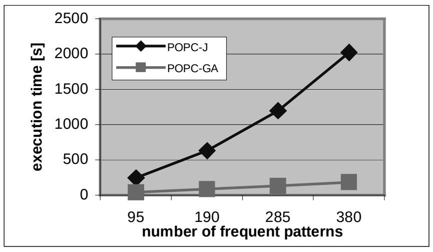

Fig. 5. Execution time for different number of frequent patterns used for clustering

Figure 4 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. Both versions of the algorithm scale linearly with the number of source sequences, which makes them 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 5 illustrates the influence of the number of frequent patterns used for clustering on the execution time of our algorithm. The time requirements of the algorithm vary with the cube of the number of patterns (the maximal possible number of iterations in the Merge Phase is equal to the number of patterns decreased by 1, in each iteration the cluster similarity matrix has to be scanned, the initial size of the matrix is equal to the square of the number of patterns). We performed experiments on a small database consisting of 200 sequences, using from 95 to 380 patterns. The experiments show that in practice the algorithm scales well with the number of patterns. This is true especially for the POPC-GA version of the algorithm, for which the cost of the Initialization Phase dominates the efficient Merge Phase.

Experiments show that both methods are scalable, but POPC-GA significantly outperforms POPC-J thanks to the fact that it does not have to retrieve clusters' contents from the database in the Merge Phase of the algorithm.

The execution times presented in the charts do not include the time needed to discover the set of frequent patterns. The cost of this pre-processing step depends strongly on the data distribution and the support threshold for patterns to be called frequent. Nevertheless, the time required to discover frequent patterns depends linearly on the database size, which preserves the overall linear scalability of the clustering method with respect to the database size. In our experiments we used the GSP algorithm [20] for pattern discovery. The time required for this step varied from 5 to 17 seconds for database sizes from 1000 to 5000 input sequences and the support threshold of 6%. This means that the time needed for pattern discovery does not contribute significantly to the overall processing time.

6 Concluding Remarks

We considered the problem of hierarchical clustering of large volumes of sequences of categorical values. We introduced two variants of the algorithm using different similarity functions to evaluate the inter-cluster similarity, which is a crucial element of the agglomerative hierarchical clustering scheme. Both of the proposed similarity measures were based on the co-occurrence of frequent patterns.

Both versions of the algorithm scale linearly with respect to the size of the source database, which is very important for large data sets. Both methods generate similar sets of clusters but the POPC-GA variant is much more efficient than POPC-J.

An important feature of the algorithm is that it does not only discover the clusters but also delivers the description of each cluster in form of patterns that are "popular" within the set of sequences forming the cluster.

In our approach clusters at any level of a cluster hierarchy can overlap. However, our method can easily be modified to generate disjoint clusters by using such techniques as placing each sequence into a cluster from whose description the sequence supports the highest number or percentage of patterns.

References

1. Agrawal R., Gehrke J., Gunopulos D., Raghavan P.: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (1998)
2. Agrawal R., Srikant R.: Mining Sequential Patterns. Proceedings of the 11th International Conference on Data Engineering (1995)
3. Agrawal, R.; Mehta, M.; Shafer, J.; Srikant, R.; Arning, A.; Bollinger, T.: The Quest Data Mining System. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (1996)
4. Bradley P.S., Fayyad U.M., Reina C.: Scaling Clustering Algorithms to Large Databases. Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (1998)

5. Broder A., Glassman S., Manasse M., Zweig G.: Syntactic clustering of the Web. Computer Networks and ISDN Systems 29, Proceedings of the 6th International WWW Conference (1997)
6. Ester M., Kriegel H-P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (1996)
7. Fisher D.H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning 2 (1987)
8. Ganti V., Gehrke J., Ramakrishnan R.: CACTUS-Clustering Categorical Data Using Summaries. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999)
9. Gibson D., Kleinberg J.M., Raghavan P.: Clustering Categorical Data: An Approach Based on Dynamical Systems. Proceedings of the 24th International Conference on Very Large Data Bases (1998)
10. Guha S., Rastogi R., Shim K.: CURE: An Efficient Clustering Algorithm for Large Databases. Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (1998)
11. Guha S., Rastogi R., Shim K.: ROCK: A Robust Clustering Algorithm for Categorical Attributes. Proceedings of the 15th International Conference on Data Engineering (1999)
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. Proceedings of the Workshop on Research Issues on Data Mining and Knowledge Discovery (1997)
14. Jain A.K., Dubes R.C.: Algorithms for Clustering Data. Prentice Hall (1988)
15. Kaufman L., Rousseeuw P.: *Finding Groups in Data*. John Wiley & Sons, New York (1989)
16. Ketterlin A.: Clustering Sequences of Complex Objects. Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (1997)
17. Lesh N., Zaki M.J., Ogihara M.: Mining Features for Sequence Classification. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999)
18. Perkowitz M., Etzioni O.: Towards Adaptive Web Sites: Conceptual Framework and Case Study. Computer Networks 31, Proceedings of the 8th International WWW Conference (1999)
19. Ramkumar G. D., Swami A.: Clustering Data Without Distance Functions. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol.21 No. 1 (1998)
20. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proceedings of the 5th International Conference on Extending Database Technology (1996)
21. Wang K., Xu C., Liu B.: Clustering Transactions Using Large Items. Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management (1999)
22. Zhang T., Ramakrishnan R., Livny M.: Birch: An efficient data clustering method for very large databases. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (1996)