2009 21st IEEE International Conference on Tools with Artificial Intelligence

Clustering Relational Data: A Transactional Approach

Gianni Costa, Alfredo Cuzzocrea, Giuseppe Manco, Riccardo Ortale
ICAR-CNR, Italy
{costa,cuzzocrea,manco,ortale}@icar.cnr.it

Abstract

A methodology for clustering multi-relational data is proposed. Initially, tuple linkages in the database schema of the multi-relational entities are leveraged to virtually *organize the available relational data into as many transactions, i.e. sets of feature-value pairs. The identified transactions are then partitioned into homogeneous groups. Each discovered cluster is equipped with a* representative*, that provides an explanation of the corresponding group of transactions, in terms of those feature-value pairs that are most likely to appear in a transaction belonging to that particular group. Outlier data are placed into a* trash cluster*, that is finally partitioned to mitigate the dissimilarity between the trash cluster and the previously generated clusters.*

1 Introduction

Clustering plays a fundamental role in the knowledge discovery process, being targeted at the unsupervised identification of valid, novel, potentially useful and, ultimately, understandable patterns in data [8]. This is achieved by grouping a collection of unlabeled objects into meaningful clusters with the requirement that the resulting groups are *homogeneous* (i.e. pairs of objects in a same cluster are highly similar) and *neatly separated* (i.e. objects within distinct clusters are very dissimilar). A wealth of clustering techniques [1, 7, 9, 11, 25] have been extensively studied in the literature. However, research efforts have predominantly focused on the development of single-table methods, which assume that some separate preprocessing task has previously organized data into a materialized table, from which to extract patterns in the attribute-value form. A major issue in this respect is that it is computationally infeasible to include all attributes, potentially useful for the specific case of clustering, into a single table, since a tuple in a relation is typically joinable with thousands of further tuples within other relations. Also, the integration process flattens the normalized organization of relational databases [22]. This originates several issues, such as the lost of both key information and original table meaning, the exacerbation of data redundancy (responsible for data integrity problems), as well as the introduction of logical and structural problems, leading to data (update/insertion/deletion) anomalies.

Relational clustering allows to suitably exploit connections among the individual data instances, to define relational patterns, i.e. patterns that involve data fragments stored across multiple relations of a relational database [6]. More precisely, in contrast to the case of single-table clustering, where a domain entity is only representable by means of a single tuple (thus raising the requirement to summarize all entity properties by means of tuple attributes), in the context of relational clustering, that domain entity can be conceptually described as an object interrelated with other entities, that syntactically translates into a set of relational database tuples, connected by linkages defined in the underlying schema (e.g. primary and foreign keys). Relational clustering offers two primary advantages w.r.t. single-table clustering. Firstly, the identification of the most appropriate reorganization of the available data, that guarantees the extraction of patterns of interest, is demanded to the specific relational clustering scheme. Secondly, background knowledge can be exploited to assist clustering.

In the present paper, we propose a novel methodology for grouping relational data, based on transactional clustering, that consists of two main steps. Initially, tuple linkages, defined in the database schema, are leveraged to preprocess the relational data, so that to *virtually* organize it into transactions, i.e. into sets of items with a specific structure. At the second step, a transactional clustering algorithm is exploited to suitably partition the identified transactions into homogeneous clusters. Every cluster is equipped with a *representative*, that provides an actionable explanation of the corresponding group of transactions, in terms of those items that are most likely to appear in a transaction within that particular cluster. Our methodology also identifies and properly manages outliers. Their corresponding transactions are placed into a *trash cluster*, that is finally partitioned to mitigate the dissimilarity between the trash cluster and the previously generated clusters.

1082-3409/09 $26.00 © 2009 IEEE
DOI 10.1109/ICTAI.2009.19

2 The Formal Framework

We here fix some notations used throughout the paper. We are given a collection of database tables $R_M, R_1, R_2, \ldots, R_m$, such that

- R_M denotes the main table, i.e. the one including the main information concerning the domain entities to be clustered;
- $R_1, \ldots, R_m$ are supplementary tables storing further details as well as domain-specific background knowledge about the entities.

The information stored within the generic table R_t conforms to the following scheme: $A_1^t, A_2^t, \ldots, A_{n_t}^t$, where n_t indicates the number of features (i.e. attributes) within R_t.

As a running example, consider the toy example depicted in Fig. 1. The figure exemplifies an unsupervised pattern recognition scenario, adapted from the Bongard's problems [2], where 28 distinct objects, divided into squares, triangles and circles, are placed within 9 cells of a plane. Objects differ in their shape, color and size. Cells occupy a certain area of the plane and correspond to specific placement instances. Precisely, every cell includes a number of objects, each of which is placed in a certain point of the cell surface. The placement process can site an object, within the generic cell, into three alternative ways: either on a portion of the cell surface, that is mutually disjoint with the individual areas occupied by the other objects in the same cell; exactly inside the cell portion of some larger object; or on an area, that partly overlaps the one occupied by another object. Besides establishing a position for each object, the placement process also orientates triangles either upward or downward.

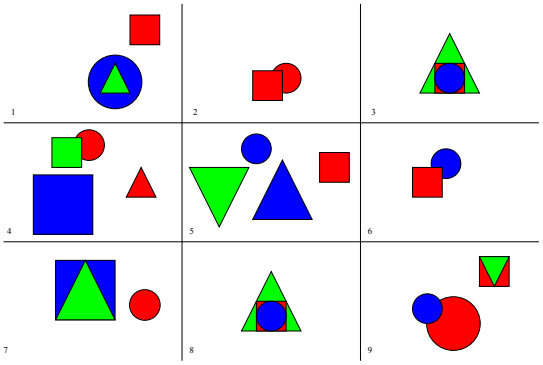
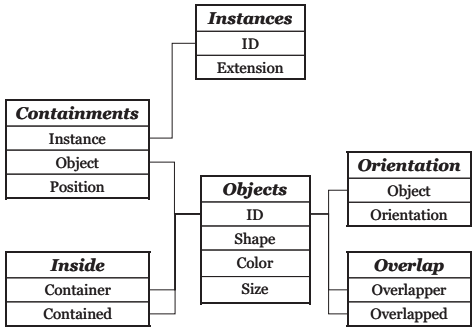


Figure 1. A toy clustering scenario

Figures 2(a) and 2(b) illustrate, respectively, the schema and some instances of the foresaid pattern recognition scenario. Notice that the relation *Containments* stores the associations between each instance and its objects, *Inside* lists inclusions between pairs of objects and *Orientation* provides the orientation of triangle objects.



(a) Schema

Containments

Instance	Object	Position
#I1	#O1	Center
#I1	#O2	Center
#I1	#O3	North East
#I3	#O6	Center
#I3	#O7	Center
#I3	#O8	Center
#I8	#O22	Center
#I8	#O23	Center
#I8	#O24	Center

Objects

ID	Shape	Color	Size
#O1	Circle	Blue	Large
#O2	Triangle	Green	Small
#O3	Square	Red	Small
#O6	Triangle	Green	Large
#O7	Square	Red	Small
#O8	Circle	Blue	Small
#O22	Triangle	Green	Large
#O23	Square	Red	Small
#O24	Circle	Blue	Small

Inside

Container	Contained
#O1	#O2
#O6	#O7
#O7	#O8
#O22	#O23
#O23	#O24

Instances

ID	Extension
#I1	Large
#I3	Medium
#I8	Medium

Orientation

Object	Orientation
#O2	Up
#O6	Up
#O22	Up

(b) Instances

Figure 2. Relational schema and data for fig.1

In this context, a meaningful clustering target might be discovering similar instances, i.e. placement patterns involving objects with homogeneous features.

The objective of clustering can hence be informally stated as finding a partition of the tuples in R_M into homogenous subgroups, where homogeneity between two entities $e_i, e_j \in R_M$ is also influenced by the information about such entities collected within $R_1, \ldots, R_m$. For example, entities #I2 and #I6 exhibit a high degree of homogeneity, represented by the fact that they contain objects with

similar shape and size, placed in the same way. But the information about shape, size and displacement of objects is spread among different tables. Thus, it is crucial in this context to provide, for a generic entity e_i, a global representation of the whole set of properties of interest. Thus, our first objective is to provide a representation of e_i by means of a set (a transaction) $\mathbf{x}_i = \{f_{i_1}, \ldots, f_{i_{n_i}}\}$ where a generic f_i represents a feature extracted either from R_M or from any other R_j. Roughly speaking, transaction $\mathbf{x}_i$ groups all information fragments (i.e. tuples), stored across tables $R_M, R_1, R_2, \ldots, R_m$, that are relative to the *i*-th domain entity of R_M. Thus, f_i represents a relation between a tuple $e_i \in R_M$ and a tuple $d \in R_t$, and is made of three main components:

1. the attribute A^{t_i} involved;
2. the *join path* relating the tuple $d \in R_t$ to the entity $e_i \in R_m$;
3. the value v that d exhibits over A^{t_i}.

In the following, we assume that v is in a range of nominal values: numeric values can be managed by resorting to appropriate discretization techniques, that can be included either as a preprocessing step, or by a tighter integration within the proposed approach.

Henceforth, for ease of notation, we refer to transactions in D rather than to relational data within $R_M, R_1, R_2, \ldots, R_m$. However, we underline that data actually exists only as tuples within $R_M, R_1, R_2, \ldots, R_m$ and that the transactional view of a domain entity denotes the focussed selection of linked tuples from such tables, i.e. the ones annotated with the identifier of the specific transaction.

In principle, various schemes for clustering transactional data can now be applied to the partition D. However, there are some issues that make traditional approaches unsuitable to this purpose, such as the enormous amount of transactions, the huge number of distinct items in a single transaction, as well as the inherent difficulty of classical algorithms at effectively cluster discrete-valued data. As an example, an important semantic feature in this context is the notion of cluster representative: namely, a transaction subsuming the characteristics of the transactions belonging to the cluster.

Transactional clustering exhibits several challenges, such as processing sets of variable length and dealing with categorical values. In particular, the latter aspect is responsible for the lack of an inherent order on the domains of the individual attributes, which prevents the definition of a notion of similarity, that catches resemblance between categorical data objects. Clearly, this imposes difficulties at devising a suitable clustering quality, that are not encountered in the case of numeric attributes where, instead, object similarity naturally follows from the geometric properties of the data. Furthermore, transactional clustering is often high-dimensional, thus requiring methods actually capable of scaling with dimensionality. The aforementioned issues are further exacerbated in a relational setting.

Agglomerative clustering [12] has been used to cope with such issues in a unified way. Though often presented in the literature as the best-quality clustering approaches, hierarchical methods are computationally inefficient, due to their quadratic complexity over the number of transactions under consideration. Moreover, it is difficult to generate a representative providing an easy interpretation of the discovered clusters. An alternative to hierarchical methods is that of exploiting the well-known partitioning approaches such as, e.g., *K-Means* and its variants [15], that have a linear scalability on the size of the dataset. In such approaches, each object $\mathbf{x}_i$ is assigned to a cluster j according to its distance $d(\mathbf{x}_i, \mathbf{m}_j)$ from a value $\mathbf{m}_j$ representing the cluster itself. $\mathbf{m}_j$ is called the *centroid* (or *representative*) of the cluster. Unfortunately, traditional partitioning methods do not supply an adequate formalism for tuples of variable size containing categorical data. Further approaches focus on the problem of clustering transactional-like data [19, 20]. Precisely, these works propose solutions, that use cosine [19] or Jaccard similarity [20], based on presence of common items and directly applicable to variable size sets. Their major limitation consists in the exploitation of the cluster centroid to approximate the cluster center. This vanishes most of the performance improvements obtained by using a compact representation of the transactions. In this paper, we adopt the *Transactional K-Means* scheme [10] to find a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$, of D, such that:

1. each C_i groups similar transactions and is associated to a centroid $\mathbf{m}_i$;
2. $\mathbf{x}_i \in C_j$ if $d(\mathbf{x}_i, \mathbf{m}_j) \leq d(\mathbf{x}_i, \mathbf{m}_l)$ for $1 \leq l \leq k$, $j \neq l$;
3. the partition $\mathcal{C}$ minimizes $\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} d^2(\mathbf{x}_j, \mathbf{m}_i)$.

The main idea of the adopted approach is to define a new notion of cluster centroid, that represents the common properties of cluster elements. Similarity inside a cluster is, hence, measured by using such a cluster representative, that becomes also a natural tool for finding an explanation of the cluster population. The specific definition of cluster centroid is based on a compact data representation model. Each transaction is represented as a boolean vector, that takes into account only the presence and absence of individual items. It has been shown in [10] that, by using the foresaid notion of cluster centroid, in conjunction with Jaccard distance, it is possible to obtain results, that are comparable in quality with those yielded by other approaches typically used for transactional clustering, but with better performances in

terms of execution time. Moreover, the cluster representative provides an immediate explanation of clusters features. A particular remark is deserved to the performance of the partitioning approach, which is relevant in case of real-time applications.

The preprocessing and clustering steps are discussed, respectively, at sections 3 and 4.

3 Virtual Transactions

As already anticipated, the time and space cost for joining tables $R_M, R_1, R_2, \ldots, R_m$ is prohibitively high. To avoid physically joining the original tables, we employ the *tuple ID propagation* technique [24], a mechanism that virtually joins tables $R_M, R_1, R_2, \ldots, R_m$, by propagating the IDs (i.e., primary keys) of the tuples within the main table, R_M, to the tuples stored in the other tables, $R_1, R_2, \ldots, R_m$. The referential integrity constraint of available tuples is exploited, so that propagation flows along the links among tuples, represented by the references between foreign and primary keys. In this paper, we inherit from [24] the original tuple ID propagation technique, and we apply this technique with the goal of "transforming" relational data into transactional ones.

In details, the propagation mechanism consists in the identification of a fixed join order for the tables, and in the annotation of each individual tuple within the supplementary tables, taken one by one according to that order. To better clarify the propagation mechanism, we discuss it in the context of a running example on the discovery of similar instances within the relational pattern-recognition database of figure 2(a).

In such a case, we consider *Instances* as the main table of our analysis, and we fix the join order as in the foreign-key-linkage (FKL)-graph of figure 3(b).

The tuple ID propagation then starts from the entry point in the FKL-graph, which represents the analysis target (in the hypothesized case, the *Instances* table), and moves towards the other tables accordingly. For each tuple t within any supplementary table R_i ($i = 1, \ldots, m$), annotation augments t by a list of identifiers, which refer tuples of R_M joinable with t (i.e., having foreign keys that point to t, or viceversa). The propagation process is incremental: the annotations of a tuple t_i within table $R_i (\neq R_M)$ are propagated to tuple t_j within $R_j (\neq R_M)$, if t_i has a foreign key that references t_j (or viceversa).

In the above example, edges in the FKL-graph drive the propagation process. As a result, the table instances in figure 2(b) are annotated as shown in figure 3(a). For example, the first tuple of *Containments* is annotated with $\langle[\texttt{Inst}], \{1\}\rangle$, as it joins with the tuple of *Instances* whose ID is 1. Similarly, the second tuple of the *Objects* table is annotated with $\langle[\texttt{Inst.Cont}], \{1\}\rangle$, since it joins with the

Containments

Instance	Object	Position	TID
#I1	#O1	Center	<[Inst],{1}>
#I1	#O2	Center	<[Inst],{1}>
#I1	#O3	North East	<[Inst],{1}>
#I3	#O6	Center	<[Inst],{3}>
#I3	#O7	Center	<[Inst],{3}>
#I3	#O8	Center	<[Inst],{3}>
#I8	#O22	Center	<[Inst],{8}>
#I8	#O23	Center	<[Inst],{8}>
#I8	#O24	Center	<[Inst],{8}>

Objects

ID	Shape	Color	Size	TID
#O1	Circle	Blue	Large	<[Inst.Cont],{1}>,<[Inst.Cont.Obj.Ins],{1}>
#O2	Triangle	Green	Small	<[Inst.Cont],{1}>
#O3	Square	Red	Small	<[Inst.Cont],{1}>
#O6	Triangle	Green	Large	<[Inst.Cont],{3}>
#O7	Square	Red	Small	<[Inst.Cont],{3}>,<[Inst.Cont.Obj.Ins],{3}>
#O8	Circle	Blue	Small	<[Inst.Cont],{3}>,<[Inst.Cont.Obj.Ins],{3}>
#O22	Triangle	Green	Large	<[Inst.Cont],{8}>
#O23	Square	Red	Small	<[Inst.Cont],{8}>,<[Inst.Cont.Obj.Ins],{8}>
#O24	Circle	Blue	Small	<[Inst.Cont],{8}>,<[Inst.Cont.Obj.Ins],{8}>

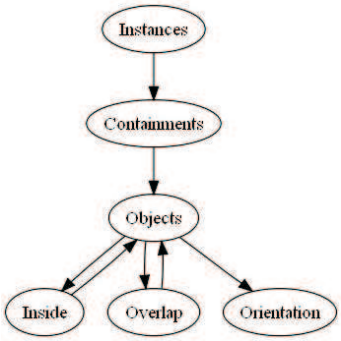
Inside

Container	Contained	TID
#O1	#O2	<[Inst.Cont.Obj],{1}>
#O6	#O7	<[Inst.Cont.Obj],{3}>
#O7	#O8	<[Inst.Cont.Obj],{3}>
#O22	#O23	<[Inst.Cont.Obj],{8}>
#O23	#O24	<[Inst.Cont.Obj],{8}>

Orientation

Object	Orientation	TID
#O2	Up	<[Inst.Cont.Obj.Ins.Obj],{1}>
#O6	Up	<[Inst.Cont.Obj.Ins.Obj],{3}>
#O22	Up	<[Inst.Cont.Obj.Ins.Obj],{8}>

(a) Tuple ID Propagation



(b) FKL graph

Figure 3. (a) Instances marked with virtual transaction IDs. (b) FKL-graph

tuple of *Instances*, whose ID is 1 throughout the second tuple of *Containments*.

Notice that, for each propagated ID, information concerning the join path is annotated as well. This allows us to effectively manage multiple references to a same tuple, due to cycles, within the FKL-graph. In figure 3(a), the first tuple of the *Objects* table is annotated twice by ID 1, as there are two join paths (`Inst.Cont` and `Inst.Cont.Obj.Ins`, respectively) which link the two tuples.

At the end of identifier propagation, semantic connections between main and supplementary tables are established, so that the transactional representation $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of the original instance data can be easily constructed by a simple linear scan of each relation (i.e. without physically joining tables $R_M, R_1, R_2, \ldots, R_m$). Each item of a transaction in D is modeled as *JoinPath.AttributeName.Value*, such that:

- *JoinPath* is the join path originating the actual item;
- *AttributeName* is the name of the attribute considered;
- *Value* is the value of such attribute.

In particular, the *i*-th transaction collects all items of those tuples annotated with value i. In the resutling transactional representation, both primary and external key attributes are ignored, as they do not provide significant information content. Figure 4 shows the transactional representation of some instances in figure 2(b), at the end of the propagation process.

$\mathbf{x}_1$={*Inst.Extension.Large*, *Inst.Cont.Position.Center*, *Inst.Cont.Position.NorthEast*, *Inst.Cont.Obj.Shape.Circle*, *Inst.Cont.Obj.Color.Blue*, *Inst.Cont.Obj.Size.Large*, *Inst.Cont.Obj.Shape.Triangle*, *Inst.Cont.Obj.Color.Green*, *Inst.Cont.Obj.Size.Small*, *Inst.Cont.Obj.Shape.Square*, *Inst.Cont.Obj.Color.Red*, *Inst.Cont.Obj.Ins.Obj.Shape.Triangle*, *Inst.Cont.Obj.Ins.Obj.Color.Green*, *Inst.Cont.Obj.Ins.Obj.Size.Small*, *Inst.Cont.Obj.Ins.Obj.Orie.Orientation.Up*}

$\mathbf{x}_3$={*Inst.Extension.Medium*, *Inst.Cont.Position.Center*, *Inst.Cont.Obj.Shape.Triangle*, *Inst.Cont.Obj.Color.Green*, *Inst.Cont.Obj.Size.Large*, *Inst.Cont.Obj.Shape.Square*, *Inst.Cont.Obj.Color.Red*, *Inst.Cont.Obj.Size.Small*, *Inst.Cont.Obj.Shape.Circle*, *Inst.Cont.Obj.Color.Blue*, *Inst.Cont.Obj.Ins.Obj.Shape.Square*, *Inst.Cont.Obj.Ins.Obj.Color.Red*, *Inst.Cont.Obj.Ins.Obj.Size.Small*, *Inst.Cont.Obj.Ins.Obj.Shape.Circle*, *Inst.Cont.Obj.Ins.Obj.Color.Blue*, *Inst.Cont.Obj.Ins.Obj.Orie.Orientation.Up*}

$\mathbf{x}_8$={*Inst.Extension.Medium*, *Inst.Cont.Position.Center*, *Inst.Cont.Obj.Shape.Triangle*, *Inst.Cont.Obj.Color.Green*, *Inst.Cont.Obj.Size.Large*, *Inst.Cont.Obj.Shape.Square*, *Inst.Cont.Obj.Color.Red*, *Inst.Cont.Obj.Size.Small*, *Inst.Cont.Obj.Shape.Circle*, *Inst.Cont.Obj.Color.Blue*, *Inst.Cont.Obj.Ins.Obj.Shape.Square*, *Inst.Cont.Obj.Ins.Obj.Color.Red*, *Inst.Cont.Obj.Ins.Obj.Size.Small*, *Inst.Cont.Obj.Ins.Obj.Shape.Circle*, *Inst.Cont.Obj.Ins.Obj.Color.Blue*, *Inst.Cont.Obj.Ins.Obj.Orie.Orientation.Up*}

Figure 4. Transactional representation of some instances of fig.2(b)

4 Clustering Process

We adopt the transactional K-Means algorithm [10] to $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ into K homogeneous groups. The scheme is an effective and efficient enhancement of the basic K-Means algorithm, capable to process volumes of (high-dimensional) categorical data. In the following, we discuss how to derive transactional K-Means from traditional K-Means.

The K-Means algorithm works as follows. First of all, K objects are randomly selected from D. Such objects correspond to some initial cluster centroids, and each remaining object in D is assigned to the cluster with the nearest centroid. Next, the algorithm iteratively recomputes the centroid of each cluster and re-assigns each object to the cluster of the nearest centroid. The algorithm terminates when the centroids do not change anymore. In that case, in fact, $\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} d^2(\mathbf{x}_j, rep(C_i))$ is minimized.

The general schema of K-Means is parametric w.r.t. both d and rep, that respectively denote the notions of distance measure and cluster centroid. Such concepts in turn are parametric to the domain of D. By suitably defining d and rep, so that they fit to the transactional domain, we can obtain an effective clustering scheme. In particular, we choose to compare transactions by means of the *Jaccard distance* [14] (denoted by d_J in the following), that measures the degree of overlap (i.e., the number of common feature-value pairs) between two transactions. Within a high-dimensional feature space, this has a main advantage of automatically preventing to consider irrelevant features for the comparison.

4.1 Cluster Representative

Transactional data requires the computation of a cluster representative. In general, given a transaction domain $\mathcal{U}$ equipped with a distance function $d : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$ and a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\} \subseteq \mathcal{U}$, the centroid of $\mathcal{S}$, $rep(\mathcal{S})$, is the element that minimizes the sum of the squared distances $rep(\mathcal{S}) = min_{\mathbf{v} \in \mathcal{U}} \sum_{i=1}^{m} d^2(\mathbf{x}_i, \mathbf{v})$. In general, computing a cluster representative is a computationally expensive process. Moreover, optimal cluster centroids are not necessarily unique. To overcome such issues, we here exploit a general property of representatives of transaction clusters, grouped based on d_J, according to which frequent feature-value pairs across transactions in a cluster are very likely to belong to the representative of that cluster [10]. This enables the exploitation of the greedy heuristic rep_H, sketched in fig. 5, which initially computes an approximation of the representative of a given cluster as the intersection of all transactions in that cluster. The approximation is iteratively refined by adding the most frequent feature-value pairs in the cluster, until the sum of the distances can be minimized.

Computing rep_H can still be expensive, since feature-value pairs have to be sorted on the basis of their frequencies. A more efficient approximation, rep_γ, can be defined by introducing a user-defined threshold value γ, representing the minimum occurrence frequency that a feature-value pair must have to be inserted into the approximation of the cluster representative. Given a specific dataset D, determining the parameter γ suitable to D is a critical research challenge that, however, is orthogonal to our work, and left as future work.

Definition 1 *Given a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ of transactions and a threshold $\gamma \in [0, 1]$, an approximate representative of $\mathcal{S}$, $rep_\gamma(\mathcal{S})$, can be defined as $rep_\gamma(\mathcal{S}) =$*

Algorithm $rep_H(\mathcal{S})$

Input : A set of transactions $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.

Output : A transaction $\mathbf{m}$ that minimizes $f(\mathbf{m}) = \sum_{\mathbf{x} \in \mathcal{S}} {d_J}^2(\mathbf{x}, \mathbf{m})$.

Method :

- sort $\bigcup \mathcal{S}$ by increasing frequency, obtaining the list $a_1, \ldots, a_m$ such that $freq(a_i, \mathcal{S}) > freq(a_{i+1}, \mathcal{S})$;
- let initially $\mathbf{m} = \bigcap_{\mathbf{x} \in \mathcal{S}} \mathbf{x}$;
- while $f(\mathbf{m})$ decreases
  - add a_h to $\mathbf{m}$, for increasing values of h.

Figure 5. Cluster-representative computation

$\{\mathbf{v} \in \bigcup_i \mathbf{x}_i \mid freq(\mathbf{v}, \mathcal{S})/m \geq \gamma\}$ *where* $freq(\mathbf{v}, \mathcal{S}) = |\{\mathbf{x}_i | \mathbf{v} \in \mathcal{S}\}|$. □

The approaches $rep_H(\mathcal{S})$ and $rep_\gamma(\mathcal{S})$ represent two viable alternatives. rep_γ represents an approximation that is extremely efficient to compute. However, it is influenced by the value of γ. Greater γ values correspond to a stronger intra-cluster similarity, less populated clusters and low-cardinality representatives. By the converse, lower γ values correspond to a weaker intra-cluster similarity, huge clusters and high-cardinality representatives. On the other side, rep_H yields a less efficient but parameter-free clustering scheme.

4.2 The Transactional K-Means Algorithm

The main scheme of the transactional K-Means algorithm is shown in Figure 6. The algorithm divides into two main phases. In the first phase, it computes $k + 1$ clusters. Transactions are assigned to the first k clusters, according to the distance measure d_J. The $(k+1)$-th cluster is referred to as trash, since it includes outlier transactions, i.e. those transactions of D that have empty intersection with the representatives of first k clusters. According to the adopted notion of Jaccard distance, such transactions are equally distant from clusters $C_1, \ldots, C_k$, which means that their assignment to any such a cluster is not significant. Therefore, outliers are placed within the trash.

The second phase is targeted at appropriately mitigating the effects of the high dissimilarity between the trash cluster and the other clusters previously generated. The basic idea consists in trying to reiterate the partitional scheme of the first phase on the trash cluster, in order to split the latter into l further clusters. Of course, the resulting final partition may include clusters with a single element, since substantially different transactions can remain within the trash, until they are chosen as cluster centroids.

Algorithm TrK**-Means(**D**,**k**,**γ**)**

Input : A dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of transactions, the desired number k of clusters. A cluster representative threshold value γ.

Output : A partition $\mathcal{C} = \{C_1, \ldots, C_{k+l}\}$ of D in $k + l$ clusters, where $l \geq 0$.

Method :

- randomly choose $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ and set $\mathbf{m}_j = \mathbf{x}_{i_j}$ for $1 \leq j \leq k$;
- repeat
  - for each j, set $C_j = \{\mathbf{x}_i | d_J(\mathbf{x}_i, \mathbf{m}_j) < d_J(\mathbf{x}_i, \mathbf{m}_l), 1 \leq l \leq k\}$;
  - set $C_{k+1} = \{\mathbf{x}_i |$ for each j $d_J(\mathbf{x}_i, \mathbf{m}_j) = 1\}$;
  - set $\mathbf{m}_j = rep(C_j)$ for $1 \leq j \leq k$;

  until m_j do not change;
- recursively apply the algorithm to C_{k+1}, producing a partition of C_{k+1} in l clusters.

Figure 6. The Transactional K-Means scheme

5 Experimental Analysis

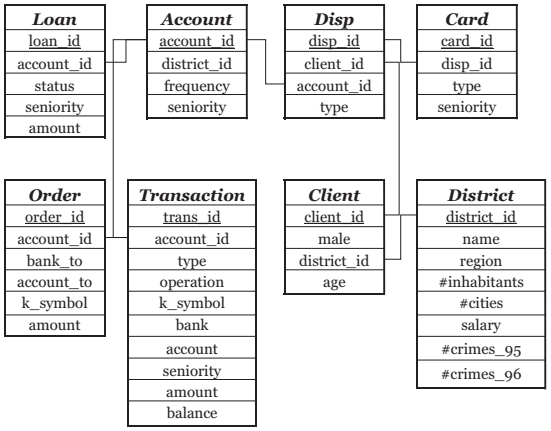
Here, the behavior of the proposed approach is preliminarily investigated. The analysis is performed to provide a taste of the quality of the results. Since clustering algorithms define structures that are not known a priori, irrespective of the clustering methods, the final partition of data requires some kind of evaluation [13]. The evaluation is useful to measure the adequacy of the discovered structure so that it can be interpreted objectively. The adequacy of a clustering structure refers to the extent in which the clustering structure provides true information about the data: a clustering structure is *valid* if it fits the dataset, i.e., if the discovered clusters correspond to the actual homogeneous groups in the dataset.

Clustering quality is assessed by measuring intra-cluster dissimilarity. Based on the definition of clustering considered in this paper, a natural quality measure associated to an instance of the K-Means approach that produces a partition $\mathcal{C} = \{C_1, \ldots, C_{k+1}\}$ of the input dataset D, such that C_{k+1} contains unclustered objects, is the following:

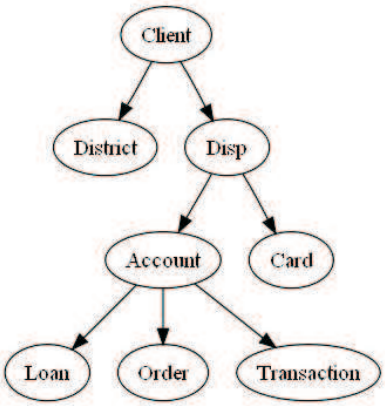
$$\mathcal{Q}_d = Avg_{i=1\ldots k} \frac{Avg_{\mathbf{x},\mathbf{y} \in \mathcal{C}_i} d(\mathbf{x}, \mathbf{y})}{Avg_{\mathbf{x},\mathbf{y} \in D} d(\mathbf{x}, \mathbf{y})}$$

$\mathcal{Q}_d$ averages the average percentages of intra-cluster distances with respect to the total distance between two transactions. A limitation of this measure is represented by the fact that large quantities of unclustered objects cause a better quality value. In other words, high γ values can cause a large trash quantity, but they can also cause an higher intra-cluster similarity. Without going into more details, we assert that in order to compute $\mathcal{Q}_d$ measure one should also carefully consider the number of unclustered objects.

To study the behavior of our approach, we conducted experiments over the financial database employed in the context of PKDD CUP 1999. We manipulated the original database in order to discretize numerical attributes. The schema of the resulting database is shown in figure 7(a). Table 1 provides details on the overall size of the database, in terms of tuples stored within the individual tables.



(a) The Financial Database Schema



(b) The FKL Graph

Figure 7. The real-life data employed for testing and its associated propagation graph

Data in the original tables was suitably preprocessed. Precisely, within table `Loan`, the attribute *seniority* was introduced to indicate the amount of time elapsed since loan arrangement: it takes ordinal values into the set $\{short, medium, long\}$. This attribute also retains a similar semantics within the `Account`, `Transaction` and `Card` tables. The attribute *amount* indicates the countervalue of `Orders` and `Transactions`. It takes values

Table	Tuple Number
`Loan`	682
`Account`	4.500
`Disp`	5.369
`Card`	892
`Order`	6.471
`Transaction`	1.056.320
`Client`	5.369
`District`	77

Table 1. Details about the financial database

into the set $\{low, medium, high\}$. The *balance* attribute within the `Transaction` table. A prior analysis of its numerical values revealed four major categories of balances, namely $\{$*negative*, *slightly positive*, *moderately positive*, *conspicuously positive*$\}$. In the `District` table, the attributes *#inhabitants*, *#cities*, *salary*, *#crimes_95* and *#crimes_96* were discretized too. In particular, the latter two are crime rates, that originally indicated the ratio between the number of committed crimes w.r.t. to the overall population, respectively, in 1995 and 1996. Both attributes were discretized so that to assume the following ordinal values $\{low, average, worrysome, high\}$. The attribute *salary* identifies three annual wage bands in local currency, i.e. $\{< 1.000.000, 1.000.000\%1.100.000, > 1.100.000\}$. Inhabitants are partitioned into four bands, i.e. $\{< 50.000, 50.000\%200.000, 200.000\%600.000, > 600.000\}$. Cities in a district belong to the set $\{< 5, 5\%8, > 8\}$. As far as the propagation of tuple IDs is concerned, we conformed to the FKL graph of figure 7(b).

We compare the quality measures obtained by the transactional K-Means algorithm in its two variants, namely the greedy-based and the threshold-based heuristics, with other K-Means methods that adopt the mean vector with Jaccard and Cosine distances. We use the real-life financial database (see figure 7(a)).

Figure 8 shows quality measures for the financial database according to the comparison algorithms and for several values of the cluster number k. As we can see, intra-class homogeneity is high. also, no unclustered objects are detected, even with high values of the γ parameter.

Interestingly, the greedy approach to computing the representative seems to perform better than the frequency-based approach. Also, large values of the γ representative seem to worsen the performance (with the exception of the first experiment). Clearly, low values of γ allow to include a higher number of feature within the representative cluster, which clearly positively affect the homogeneity among entities. This is a clear effect of the benefits of relational clustering w.r.t. traditional clustering methods. Even the high profitability of greedy approach finds an explanation

in this setting, the difference being that the greedy procedure allows the clustering process to self-tune the size of the representative.

Particularly significant is the absence of unclustered objects, even with high values of the γ parameter. Within a relational setting, tuples in the main table are very likely to share some properties according to some join path. hence, it is quite rare to observe objects equally distant from each cluster. Once again, this testifies the benefits of considering relational properties when clustering, as opposed to fixed schemes based on a single view.

k	*Measure*	*Greedy*	$\gamma = 0.025$	$\gamma = 0.05$	$\gamma = 0.1$	$\gamma = 0.2$	$\gamma = 0.5$
4	Q_d	0.912796	0.828066	0.819579	0.8048	0.85486	0.876241
	Trash	0	0	0	0	0	0
6	Q_d	0.850012	0.828066	0.817364	0.825472	0.81046	0.778616
	Trash	0	0	0	0	0	0
8	Q_d	0.789604	0.859053	0.810132	0.755044	0.747546	0.75916
	Trash	0	0	0	0	0	0

Figure 8. Quality results

6 Conclusions and Future Work

We preliminarily discussed a hybrid methodology for clustering multi-relational data. The latter is organized into virtual transactions, which are eventually partitioned into homogeneous groups.

A major direction of future research is to comparatively study the empirical behaviour of the method.

References

[1] R. Agrawal et al. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *ACM SIGMOD*, 94–105, 1998.

[2] M. Bongard. *Pattern Recognition*. Spartan Books, 1970.

[3] M. Craven et al. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *ML*, 43: 97–119, 2001.

[4] S. Džeroski et al. Classification of River Water Quality Data using Machine Learning. *ENVIROSOFT*, 129–137, 1994.

[5] S. Džeroski et al. Detecting Traffic Problems with ILP. *ILP*, 281–290, 1998.

[6] S. Džeroski. *Relational Data Mining*. Springer-Verlag, 2001.

[7] M. Ester et al. Incremental Clustering for Mining in a Data Warehousing Environment. *VLDB*, 323–333, 1998.

[8] U.M. Fayyad et al. *Advances in Knowledge Discovery and Data Mining*. The IT Press, 1996.

[9] D. Fisher. Knowledge Acquisition via Incremental Conceptual Clustering. *ML*, 2: 139–172, 1987.

[10] C. Gozzi et al. Clustering Transactional Data. *PKDD*, 175–187, 2002.

[11] S. Guha et al. CURE: An Efficient Clustering Algorithm for Large Databases. *ACM SIGMOD*, 73–84, 1998.

[12] S. Guha et al. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *IS*, 25(3): 345–366, 2000.

[13] M. Halkidi et al. Cluster Validity Methods. *SIGMOD Record*, 31(1-2): 175–187, 2002.

[14] J. Han et al. *Data Mining: Concepts and Techniques*. Morgan Kaufman, 2001.

[15] Z. Huang. Extensions to the K-Means Algorithm for Clustering Large Data Sets with Categorical Values. *DAMI*, 2(3): 283–304, 1998.

[16] R.D. King et al. Relating Chemical Activity to Structure: An Examination of ILP Successes. *NGC*, 13: 411–433, 1995.

[17] M. Kirsten et al. Extending K-Means Clustering to First-Order Representations. *ILP*, 112–129, 2000.

[18] F. Mizoguchi et al. Using Inductive Logic Programming to Learn Classification Rules that Identify Glaucomatous Eyes. *IDAMP*, 227–242, 1997.

[19] M. Steinbach et al. A Comparison of Document Clustering Techniques. *ACM SIGKDD TMW*, 2000.

[20] A. Strehl et al. Impact of Similarity Measures on Webpage Clustering. *AAAI AIW*, 58–64, 2000.

[21] M. Turcotte et al. The Effect of Relational Background Knowledge on Learning of Protein Three-Dimensional Fold Signatures. *ML*, 43(1-2): 81–96, 2001.

[22] J.D. Ullman et al. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[23] X. Xin et al. CrossClus: User-Guided Multi-Relational Clustering. *DAMI*, 15(3): 321–348, 2007.

[24] X. Yin et al. Efficient Classification across Multiple Database Relations: A CrossMine Approach. *IEEE TKDE*, 18(6): 770–783, 2006.

[25] T. Zhang et al. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD*, 103–114, 1996.