Data Min Knowl Disc
DOI 10.1007/s10618-011-0218-x

Using trees to mine multirelational databases

Aída Jiménez · Fernando Berzal · Juan-Carlos Cubero

Received: 29 October 2009 / Accepted: 7 March 2011
© The Author(s) 2011

Abstract This paper proposes a new approach to mine multirelational databases. Our approach is based on the representation of multirelational databases as sets of trees, for which we propose two alternative representation schemes. Tree mining techniques can thus be applied as the basis for multirelational data mining techniques, such as multirelational classification or multirelational clustering. We analyze the differences between identifying induced and embedded tree patterns in the proposed tree-based representation schemes and we study the relationships among the sets of tree patterns that can be discovered in each case. This paper also describes how these frequent tree patterns can be used, for instance, to mine association rules in multirelational databases.

Keywords Multirelational databases · Frequent itemset mining · Association rules · Tree pattern mining

---

Responsible editor: Eamonn Keogh.

A. Jiménez (✉) · F. Berzal · J.-C. Cubero
Department of Computer Science and Artificial Intelligence, ETSIIT—University of Granada, 18071 Granada, Spain
e-mail: aidajm@decsai.ugr.es

F. Berzal
e-mail: fberzal@decsai.ugr.es

J.-C. Cubero
e-mail: jc.cubero@decsai.ugr.es

Published online: 18 March 2011

1 Introduction

Data mining techniques have been developed to extract potentially useful information from databases. Classification, clustering, and association rules have been widely used. However, many existing techniques usually require all the interesting data to be in a single table.

Several techniques have been proposed in the literature to handle several tuples at once. Some algorithms explore tuples that are somehow related, albeit still in the same table (Tung et al. 2003; Lee and Wang 2007). Other algorithms, however, are able to extract information from multirelational databases, i.e., they take into account not just a single table, but all the tables that are connected to it (Džeroski 2003). For instance, these algorithms have been used for multirelational classification (Yin et al. 2004) and multirelational clustering (Yin et al. 2005).

In this paper, we propose two alternative representation schemes for multirelational databases. Our representation schemes are based on trees, so that we can apply existing tree mining techniques to identify frequent patterns in multirelational databases. We also compare the proposed representation schemes to determine which one should be used depending on the information we would like to obtain from the database.

Figure 1 shows an instance of the prototypical multirelational database with two relations: `basket` and `item`. Our approach consists of transforming this database into a set of trees, using one tree for representing each tuple in a particular relation from the multirelational database. The result of this transformation, starting from the `basket` relation, is shown in Fig. 2.

A typical solution to multirelational database mining problems consists of joining all the relations of our interest into a single relation, usually called universal relation (Fagin et al. 1982; Maier and Ullman 1983; Maier et al. 1984; Ullman 1990). Then, classical data mining techniques can be applied to this universal relation. However, join-based techniques suffer from a serious disadvantage: they do not preserve the proper support counts. Figure 3 shows the result of joining the `basket` and `item`

basket

id	date	customer	zipcode
1	Thursday	Anna	60608
2	Saturday	Peter	60607
3	Thursday	John	60611

item

basket	product	qty	price
1	toner	2	75
1	printer	1	199
1	netbook	1	299
2	toner	3	75
3	toner	3	75
3	printer	2	199

Fig. 1 Simple multirelational database example

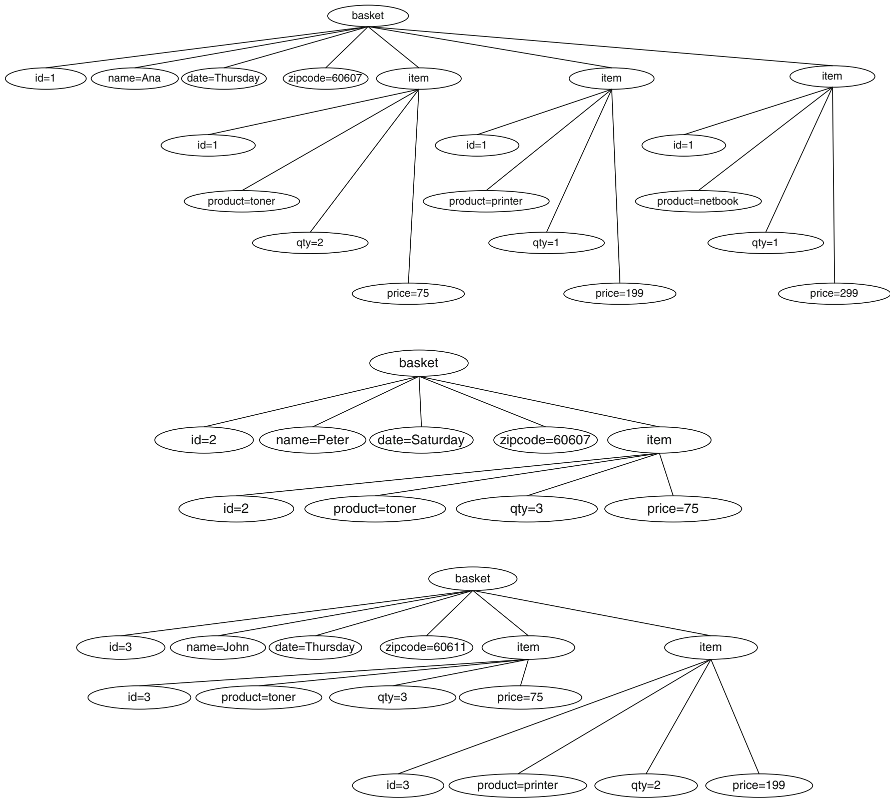

Fig. 2 Tree representation of the multirelational database in Fig. 1

item $\bowtie$ basket

basket	product	qty	price	date	customer	zipcode
1	toner	2	75	Thursday	Anna	60608
1	printer	1	199	Thursday	Anna	60608
1	netbook	1	299	Thursday	Anna	60608
2	toner	3	75	Saturday	Peter	60607
3	toner	3	75	Thursday	John	60611
3	printer	2	199	Thursday	John	60611

Fig. 3 Joined 'universal' relation from the multirelational database in Fig. 1

relations. In the resulting relation, the support of Thursday is 83% and the support of the 60608 zip code is 50%, whereas, if we look at the original `basket` relation, we can see that the actual support of Thursday is 66% and the actual support of the 60608 zip code is 33%. Using our tree-based proposal, we do not introduce such distortions in our support counts and we always preserve their original values.

Our paper is organized as follows. We introduce some standard terms and related work in Sect. 2. Section 3 describes two different schemes for representing multirelational databases as sets of trees. Some implementation issues when deriving trees from

actual multirelational databases are addressed in Sect. 4. Section 5 studies the kinds of patterns that can be identified from such trees, while Sect. 6 explains how association rules we can extract association rules defined over the trees obtained from a multirelational database, as well as the constraints that can be used to improve the performance of the association rule mining process. Finally, we present some experimental results in Sect. 7 and we end our paper with some conclusions in Sect. 8.

2 Background

In this section, we introduce some basic concepts and published results on tree pattern mining and multirelational data mining.

2.1 Tree pattern mining

A *labeled tree* is a connected acyclic graph that consists of a vertex set V, an edge set $E \subseteq V \times V$, an alphabet Σ for vertex and edge labels, and a labeling function $L : V \cup E \rightarrow \Sigma \cup \varepsilon$, where ε stands for the empty label. The size of a tree is defined as the number of nodes it contains. Optionally, a tree can also have a predefined root, v_0, and a binary ordering relationship $\leq$ defined over its nodes (i.e. '$\leq$' $\subseteq V \times V$).

A *canonical tree representation* is an unique way of representing a labeled tree. This canonical representation makes the problems of tree comparison and subtree enumeration easier. Several alternatives have been proposed in the literature to represent trees as strings (Chi et al. 2003). In this paper, we will use a depth-first-based codification. In a depth-first codification, the string representing a tree is built by adding the label of each tree node in a depth-first order. A special symbol $\uparrow$, which is not in the label alphabet, is used when the sequence comes back from a child to its parent.

Many frequent tree pattern mining algorithms have been proposed in the literature (Jimenez et al. 2010a). These algorithms are usually derived from either Apriori (Agrawal and Srikant 1994) or FP-Growth (Han et al. 2004). Most tree pattern mining algorithms follow the Apriori iterative pattern mining strategy, where each iteration is broken up into two distinct phases: candidate generation and support counting. Some of these algorithms are FreqT (Abe et al. 2002), TreeMiner (Zaki 2005b), SLEUTH (Zaki 2005a), and POTMiner (Jimenez et al. 2010b). Other algorithms follow the FP-Growth pattern growth approach and they do not explicitly generate candidates. For instance, the PathJoin algorithm (Xiao et al. 2003) uses compacted structures called FP-Trees to encode input data, while Chopper and XSpanner (Wang et al. 2004) codify trees as sequences and identify frequent subtrees by discovering frequent subsequences.

2.2 Multirelational data mining

Multirelational data mining techniques look for patterns that involve multiple relations (tables) in a relational database. In a relational database, a relation can be defined as $r = (A^r, K^r, FK^r)$ where $A^r = \{A_1^r, \ldots, A_n^r\}$ is a set of attributes, $K^r \subseteq A^r$ represents the primary key of the relation and $FK^r = \{FK_1^r, \ldots, FK_m^r\}$ is the set of

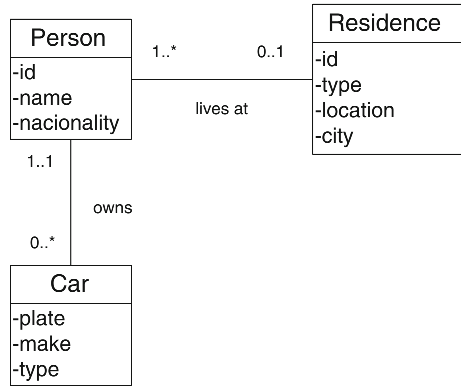

Fig. 4 Multirelational database schema in UML notation

Person			
id	**name**	**nationality**	**houseID**
1	Peter	US	5
…	…	…	

Residence			
id	**type**	**location**	**city**
5	apartment	suburb	London
…	…	…	

Car			
plate	**make**	**type**	**ownerID**
1234BCD	Toyota	SUV	1
5678JKL	Chevrolet	sedan	1
…	…	…	

Fig. 5 Multirelational representation of the database schema in Fig. 4

foreign keys in r. Each foreign key can be defined as $FK_i^r = (F_i^r, s_i)$ where $F_i^r \subseteq A^r$ and s_i is the relation whose primary key K^{s_i} is referenced by FK_i^r.

Figure 4 depicts the conceptual schema of a multirelational database using the UML notation (Booch et al. 2005). Given such a conceptual schema, we can derive the suitable logical data model for a relational database (Garcia-Molina et al. 2008; Silberschatz et al. 2001). The relations obtained from the conceptual model in Fig. 4 are shown in Fig. 5 with some example tuples.

Many different techniques have been proposed in the literature to deal with multirelational databases in a machine learning context. We have classified the proposed techniques into three broad categories: inductive logic programming, tuple ID propagation, and tree-based algorithms.

– Many multirelational data mining algorithms come from the field of *Inductive Logic Programming* (Džeroski 2003).
  RELAGGS is a database-oriented approach based on aggregations that collects data from adjacent tables (Krogel and Wrobel 2003). ACORA (Perlich and Provost

2006) also employs aggregations to obtain a single table for classification problems.

Another relevant proposal that deals with multirelational structures is WARMR (King et al. 2001). WARMR employs Datalog, a logic programming language (Ullman 1988), and it is a level-wise algorithm like Apriori (Agrawal and Srikant 1994). It has been used to discover frequent patterns in chemical compounds.

Caraxterix (Turmeaux et al. 2003) was proposed to mine characteristic rules, where characterization consists of discovering properties that characterize objects by taking into account their properties and also properties of the objects linked to them.

– *Tuple ID propagation* is an efficient approach for virtually joining relations, as well as searching and propagating information among them. This technique propagates the IDs of target tuples to other relations, together with their associated class labels where appropriate. A classifier, dubbed CrossMiner (Yin et al. 2004), and a clustering algorithm, called CrossClus (Yin et al. 2005), have been proposed using this technique.
– *Tree-based algorithms* have also been used for multirelational data mining. FAT-miner (De Knijf 2007), for instance, is a frequent tree pattern mining algorithm that seems to implicitly employ a representation similar to our object-based representation scheme, albeit the particular algorithm employed to derive the trees from a multirelational database is not explicitly described in De Knijf (2007) nor De Knijf (2006).

Trees have been used in many other data mining areas, although, to the best of our knowledge, our use of trees in this paper has not been formally addressed elsewhere. Decision trees, for example, have been applied for classification and clustering. TILDE (Blockeel and Raedt 1998) is an ILP-based decision tree inducer. Another algorithm, MRDTL (Leiva et al. 2002), applies decision trees to multirelational databases. In the field of statistical learning, we can also find many other examples, such as probability trees (Neville et al. 2003) and spatiotemporal relational probability trees (McGovern et al. 2008). The former, as our proposal, uses foreign keys to mine multirelational databases for classification purposes.

3 Multirelational database tree representation

In this section, we describe how we can obtain a set of trees from a given multirelational database using two different representation schemes.

The analysis of multirelational databases typically starts from a particular relation. This relation, which we will call *target relation* (or target table), is selected by the end user according to her specific goals.

We introduce two different schemes for representing multirelational databases as sets of trees. The key-based tree representation scheme draws from the concept of identity in the relational model while the object-based representation scheme is based on the concept of identity in object-oriented models. Relational databases rely on primary keys to ensure that each tuple can be univocally referenced within a given relation. In the object model, however, each object is already unique and no specific key is needed. In object databases, each object is automatically assigned an unique

ID, which means that you can create objects that have identical field values but are still different objects (Paterson et al. 2006).

The main idea behind our two representation schemes is building a tree from each tuple in the target table and following the links between tables (i.e. the foreign keys) to collect all the information related to each particular tuple in the target table. In both representation schemes, we will use the name of the target table as the label at the root of the trees.

From a database point of view, data can be classified into two types: atomic and compound. Atomic data cannot be decomposed into smaller pieces by the database management system (DBMS). Compound data, consisting of structured combinations of atomic data, can be decomposed by the DBMS (Codd 1990). We will represent the value a_i of an atomic attribute A_i as $A_i = a_i$ in a tree node and, for compound attributes, as $(A_1, \ldots, A_n) = (a_1, \ldots, a_n)$. In the following sections, with the aim of clarifying the notation, we will suppose that all the attributes are atomic, assuming that, if they were compound, the notation $(A_1, \ldots, A_n) = (a_1, \ldots, a_n)$ should be used instead.

3.1 Key-based tree representation

The key-based tree representation scheme is based on the concept of identity in the relational model, i.e., each tuple is identified by its primary key. Therefore, the root node of the key-based tree representing a tuple in the target relation r will have, as its unique child, the value of the primary key of the tuple in the target relation that is represented by the tree. The children of this primary key node will be the remaining attribute values from the tuple in the target relation. The rest of the tree is then built by exploring the foreign keys that connect the target relation to other relations in the database. From a conceptual point of view, we have two different scenarios:

- When we have a one-to-one or many-to-one relationship between two relations, we will have a foreign key in the target table r pointing to another relation s. That results in a subtree with the data from the tuple in s.
- When we have a one-to-many relationship, we will have a foreign key in the table s that refers to the primary key of table r. In this case, many tuples in s may point to the same tuple in r. A different subtree results from the data for each tuple in s that points to the target tuple in r.

Formally, the algorithm needed to build a key-based tree starting from each tuple in the target relation $r = (A^r, K^r, FK^r)$ is the following:

- Build the root node, whose label is the name of the target relation, i.e. r.
- Add a child node to the root corresponding to the primary key of the tuple in the target relation using the notation $r.K^r = k^r$.
- For each attribute $A_i^r \in A^r$, add a child node to the primary key node using the notation $r.A_i^r = a_i^r$.
- For each foreign key $FK^r = (F^r, s)$ pointing to another relation, s, create a key-based tree representation for the tuple in s that is referenced by the tuple in r, i.e.:

  - Add a child node to the primary key node using the notation $r.F^r = f^r$.
  - For each attribute $A_i^s \in A^s$ from the s relation, add a child node to the $r.F^r = f^r$ node using the notation $r.F^r.A_i^s = a_i^s$.
- For each foreign key $FK^s = (F^s, r)$ in another relation, s, pointing to our target relation, r, create a key-based tree representation for each tuple in s that points to our tuple in r:
  - Add a child node to the primary key node with the name of the two relations, the foreign key, and the primary key of s using the notation: $r.s[F^s].K^s = k^s$.
  - For each attribute $A_i^s \in A^s$ in the s relation, add a child node to $r.s[F^s].K^s = k^s$ using the notation $r.s[F^s].A_i^s = a_i^s$.
- This algorithm is recursively applied, taking into account the foreign keys in s that point to other relations in the database, as well as the foreign keys pointing to s from other relations in the multirelational database.

Let us suppose that, in the multirelational database of Fig. 4, our target table is `person`, its primary key is *id*, and its only attributes are as shown in Fig. 5. If we have the tuple {1, Peter, US, 5}, its key-based tree representation will be the one we see in Fig. 6a. In textual form (see Sect. 2.1), this tree can be represented as follows:

person
    person.id=1
        person.name=Peter ↑
        person.email=US ↑
        person.houseID=5 ↑↑

When we consider the links (foreign keys) between tables in our example database, we find the two situations we have described above:

- The relationship between `person` and `residence` is one-to-one. Therefore, we have a foreign key in the `person` table that refers to the `house` of each `person`. This relationship leads to the subtree in Fig. 6b whose nodes are depicted with vertical lines in their background.
- The relationship between `person` and `car` is one-to-many. Therefore, the `car` table includes a foreign key that refers to the car owner. The key-based tree representation of this relationship is shown by the nodes shaded with horizontal lines in Fig. 6b.

3.2 Object-based tree representation

The object-based tree representation scheme is based on the concept of object identity in an object-oriented model. In this representation scheme, we will use intermediate nodes as roots of the subtrees representing each tuple in the multirelational database. All the attribute values within the tuple, including the primary key attributes, will be children of the root node representing the tuple in the tree.

As in the previous section, the tree is built by exploring the relationships between the target relation and other relations in the multirelational database:

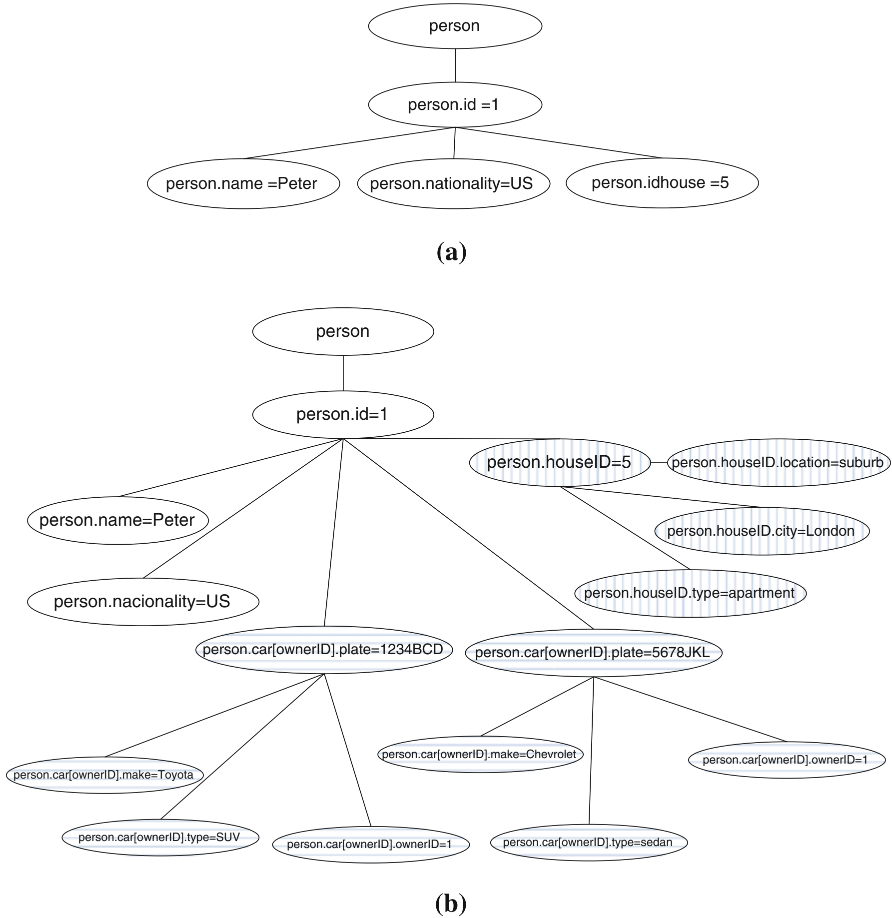

Fig. 6 Key-based tree representation of the multirelational database shown in Fig. 5

- The case of one-to-one and many-to-one relationships between two relations, r and s, where r has a foreign key pointing to s, is now addressed by adding the attributes of s as children of the intermediate node labeled with the names of the foreign key attributes.
- When the relationship is one-to-many, i.e. when the relation s has a foreign key that refers to the target relation r, a new subtree is built for each tuple in s that points to the same tuple in r. These subtrees will have root nodes labeled with the name of both tables and the foreign key involved in the relation, while their children will contain all the attribute values from the tuples in s.

Formally, the algorithm to build a object-based tree starting from each tuple of the target relation $r = (A^r, K^r, F^r)$ is:

- Build the root node, whose label is the name of the target relation, i.e. r.

- For each attribute $A_i^r \in A_r$, including the primary key attributes, add a child node to the root using the notation $r.A_i^r = a_i^r$.
- For each foreign key $FK^r = (F^r, s)$ pointing to another relation, s, create a subtree with the data from the tuple in s that is referenced by the tuple in r:
  - Add a child node to the root using the notation $r.F^r$.
  - For each attribute $A_i^s \in A^s$ in the s relation, including primary key attributes, add a child node to the intermediate node we have just created using the notation $r.F^r.A_i^s = a_i^s$.
- For each foreign key $FK^s = (F^s, r)$ in another relation, s, pointing to our target relation, r, create a subtree for each tuple in s that points to the target tuple in r:
  - Add a child node to the root with the name of both relations and the foreign key attribute names using the notation: $r.s[F^s]$.
  - For each attribute $A_i^s \in A^s$ in the s relation, including the primary key attributes in K^s, add a child node to the $r.s[F^s]$ node that we have just created using the notation $r.s[F^s].A_i^s = a_i^s$.
- This procedure is recursively applied, taking into account the foreign keys in s that point to other relations in the database, as well as the foreign keys pointing to s from other relations in the multirelational database.

The example from Fig. 6 is now displayed in Fig. 7 using the object-based representation scheme. The nodes decorated with vertical lines in Fig. 7b illustrate the many-to-one relationship between `person` and `residence`, while the one-to-many relationship between `person` and `car` is depicted by the nodes with horizontal lines in their background.

The main difference between the object-based tree representation scheme and the key-based one is that, in the object-based representation scheme, primary key attribute values and non-prime attribute values appear at the tree level. Using the key-based representation scheme, however, non-prime attribute values within each tuple appear as children of the node representing the primary key.

It should also be noted that the object-based tree representation scheme generates trees with more nodes than the key-based one, since it introduces ancillary intermediate nodes, i.e., those nodes that do not contain values and just represent the start of a new tuple within the tree.

However, tree depth is typically lower in the object-based tree representation scheme than in the key-based one. When representing tuples from the target relation, no key nodes are needed in the object-based representation (i.e., the depth of the resulting tree is 2, as in Fig. 6a). In the key-based representation, however, the primary-key node adds a new level to the tree (i.e., the depth of the resulting tree is 3, as shown in Fig. 7a).

Section 5 will show how the use of these two different tree-based representation schemes will be useful to identify different kinds of patterns, but first we should address some implementation issues that arise in practice.

4 Deriving trees from a multirelational database

In this section, we discuss how we can traverse the foreign keys that connect individual relations in a multirelational database.

Fig. 7 Object-based tree representation of the multirelational database shown in Fig. 5

4.1 Exploration depth

The connections between relations in a given multirelational database can be represented as a graph whose nodes are relations and whose edges represent foreign keys connecting pairs of relations. Starting from the target relation, we can traverse such a graph. The tree resulting from this traversal will grow each time we visit a new graph node (relation) or revisit an already-discovered one. From the multirelational database perspective, the size and depth of the resulting tree depends on the number of links that we follow, starting from the target relation.

When the multirelational database is complex, the relations that are far away from the target relation might not always be interesting for us. Therefore, in practice, we should select the relations we want to represent in the resulting trees or, at the very least, bound the resulting tree depth.

We can define the *exploration depth* for the tree-based representation of multirelational databases as the length of the longest paths from the target relation to the other relations represented within the trees.

For example, in Figs. 6a and 7a, exploration depth is 0, since only the target relation is represented in those trees. In Figs. 6b and 7b, however, exploration depth is 1 because we have followed all the foreign keys that connect our target relation to other relations in our database.

4.2 Relationship traversal

When we are building the trees corresponding to each tuple in our target relation, all the foreign keys between the relations represented in the tree are traversed forward starting from the target relation. Apart from this, we must consider whether it is interesting to go back through a foreign key that is already represented in the tree, i.e. whether to traverse it backwards or not.

To solve this problem, we focus on the relationships in our (high-level) conceptual database schema rather than on the (low-level) foreign keys that appear in our relational data model. When relationships are one-to-one or one-to-many, it is not necessary to traverse them backwards because we would just obtain the same data that we have already have included in the tree. However, if a relation is many-to-one or many-to-many, we should traverse that relationship backwards to obtain all the tuples that are connected to the tuple in our target relation that we are representing in tree format.

For example, consider the object-based tree in Fig. 7b. When we reach the node labeled $person.car[ownerID].ownerID = 5$, we have represented the information about Peter and his cars. Therefore, it is not necessary to go back through the `person–car` relationship because we would obtain the information about Peter that we already have in the tree. However, if the target table were `car`, as shown in Fig. 8, we would first represent the information of the Toyota car. Next, we would traverse the `car–person` relationship to obtain the information about the owner of the car (Peter). Finally, we should go back through the `person–car` relationship to represent all the cars that Peter owns, not just the Toyota we started with.

5 Identifying frequent patterns in multirelational databases

The use of tree-based representation schemes for multirelational databases lets us apply tree mining techniques to identify frequent patterns in multirelational databases. Many algorithms have been proposed in the literature to identify frequent tree patterns, including TreeMiner (Zaki 2005b), SLEUTH (Zaki 2005a), and POTMiner (Jimenez et al. 2010b). Using these algorithms on the tree-based representation schemes we have introduced in Sect. 3, we will be able to discover different kinds of patterns.

Fig. 8 Object-based tree derived from the multirelational database in Fig. 5 using `car` as the target relation

In this section, we will analyze them and we will study the relationships among the sets of patterns that can be discovered from a multirelational database.

5.1 Identifying different kinds of patterns

When working with tree pattern mining algorithms, different kinds of subtrees can be defined depending on the way we define the matching function between the pattern and the tree it derives from (Chi et al. 2005; Jimenez et al. 2010a):

- A *bottom-up subtree* T' of T, with root v, can be obtained by taking one vertex v from T with all its descendants and their corresponding edges.
- An *induced subtree* T' can be obtained from a tree T by repeatedly removing leaf nodes from a bottom-up subtree of T.
- An *embedded subtree* T' can be obtained from a tree T by repeatedly removing nodes, provided that ancestor relationships among the vertices of T are not broken.

Bottom-up subtrees are a special case of induced subtrees. Likewise, induced subtrees are a special case of embedded subtrees. Frequent tree pattern mining algorithms usually focus on identifying induced or embedded subtrees as the ones shown in Fig. 9. In the following subsections, we will examine the induced and embedded patterns we can discover when using both the key-based and the object-based tree representation schemes.

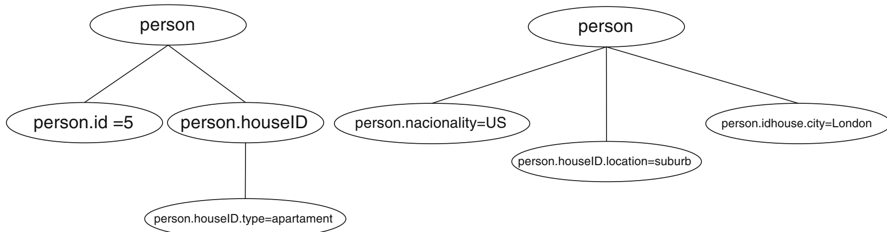

Fig. 9 An induced subtree (*left*) and an embedded subtree (*right*) from the tree shown in Fig. 7b

5.1.1 Induced key-based patterns

The key-based representation scheme uses primary keys as root nodes for the different subtrees that represent each tuple in the tree. Since those primary key nodes are not frequent in the tree database resulting from the multirelational database and induced patterns preserve all the nodes as in their original trees, no induced patterns starting at, or including, a primary key node will be identified using this representation scheme. However, it is certainly possible that we can identify induced patterns starting at foreign keys, since they might be frequent in the tree database. It should be noted, however, that all the information they contain will usually be from the same relation in our multirelational database.

For example, the `person.id=1` node in Fig. 6b will not be frequent, since `person.id` is the primary key of the target table and it appears in just one database tree. Hence, no induced patterns starting at this node will be identified. It is possible, however, that induced patterns starting at node `person.houseID=5` may be frequent if there were more people sharing the same residence.

5.1.2 Embedded key-based patterns

If we are interested in obtaining patterns involving data from different relations in the key-based representation scheme, we will have to resort to embedded patterns. These patterns will miss some intermediate nodes, such as the primary key nodes, but they will be able to combine information from different relations in our original multirelational database.

For example, we could obtain the pattern in Fig. 10 from the tree in Fig. 6b. This pattern represents that people living in suburbs and owning a sedan car are frequent in our database.

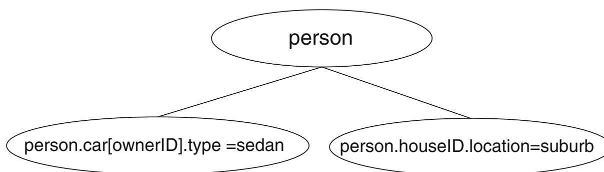

Fig. 10 Embedded subtree obtained from the key-based tree in Fig. 6b

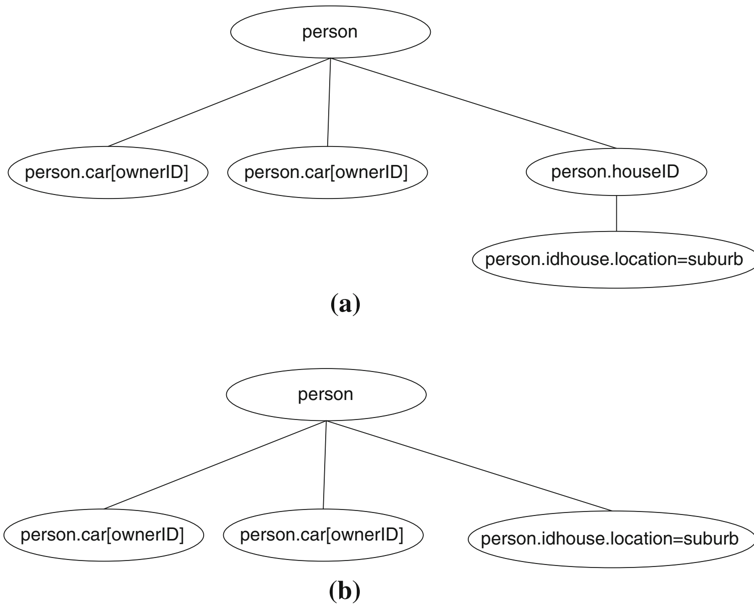

Fig. 11 Subtrees from the object-based tree in Fig. 7

5.1.3 *Induced object-based patterns*

The object-based representation scheme uses intermediate nodes to represent references to the different tuples represented within the trees. Using this representation scheme, it is possible to identify larger induced patterns than before: Induced object-based patterns can contain information from different relations, something that was not possible when we used the key-based representation scheme.

When appearing as part of frequent patterns, the intermediate nodes provide additional information. They tell us how many tuples in one relation are related to a given tuple in our target relation. For example, the tree pattern in Fig. 11a shows that people living in suburbs and owning (at least) two cars are frequent in our database, without considering car details. It should be noted that this kind of patterns cannot be identified using the key-based representation scheme.

5.1.4 *Embedded object-based patterns*

When mining embedded patterns using the object-based representation scheme, we will obtain all the patterns that were identified when we used the key-based representation scheme, as well as those patterns that contain intermediate nodes.

As happened with induced object-based patterns, the use of intermediate nodes will let us identify patterns that were not discoverable using the key-based representation scheme, such as the one shown in Fig. 11b, which is extracted from the tree in Fig. 7b.

However, it should be noted that number of patterns including intermediate nodes could be high because these nodes are typically frequent in the trees representing the

multirelational database. Hence, their discovery comes at a price that we will analyze in our experiments in Sect. 7.

5.2 Induced versus embedded patterns

Induced and embedded patterns provide us different information about the multirelational database.

In some sense, induced patterns describe the database in fine detail. Induced patterns preserve the structure of the original trees in the database by maintaining the relationships among all their nodes as they appear in the tree-based representation of the multirelational database. This causes that, in order to obtain useful information from the multirelational database, the identification of large patterns is often necessary. Unfortunately, this need to unveil large patterns might involve a large computational effort.

Embedded patterns are typically smaller than the induced patterns required to represent the same kind of information. However, if we use embedded patterns, some of the relationships among the nodes in the original trees are not preserved. In other words, we might not be able to rebuild the original tree structure from an embedded tree pattern.

For example, Fig. 12 shows some patterns obtained from the object-based tree representation scheme of the multirelational database in Fig. 4. The induced pattern shown on the left tells us that some people in our database have a Toyota *and* a sedan car, while the induced pattern on the right tells us that people in our database have a Toyota *that* is also a sedan car. The embedded pattern shown in the same figure illustrates that some people have a Toyota car and a sedan car, but we do not know if it is the same car (a Toyota sedan) or they own two cars (a Toyota and a sedan car, which is different from the Toyota). We cannot be sure of the original tree that led to the discovered embedded pattern. In other words, embedded patterns can introduce some ambiguity in their interpretation.

5.3 Key-based versus object-based patterns

The key-based and the object-based tree representation schemes also provide us different information about the multirelational database.

When we use the key-based representation scheme, no induced patterns with information about the target table can be identified. Induced patterns might contain, however, information about tuples in other tables, those that are frequently related to the tuples in the target table.

When we use the object-based representation scheme, induced patterns with information about the target table can now be obtained. Therefore, the object-based representation scheme is our only choice if we are interested in induced patterns, which preserve the original structure of the trees in the database.

On the other hand, using the object-based representation scheme to discover embedded patterns is useful only if we are interested in patterns that show that a particular object in a given relation is related to at least n objects in another relation. An example

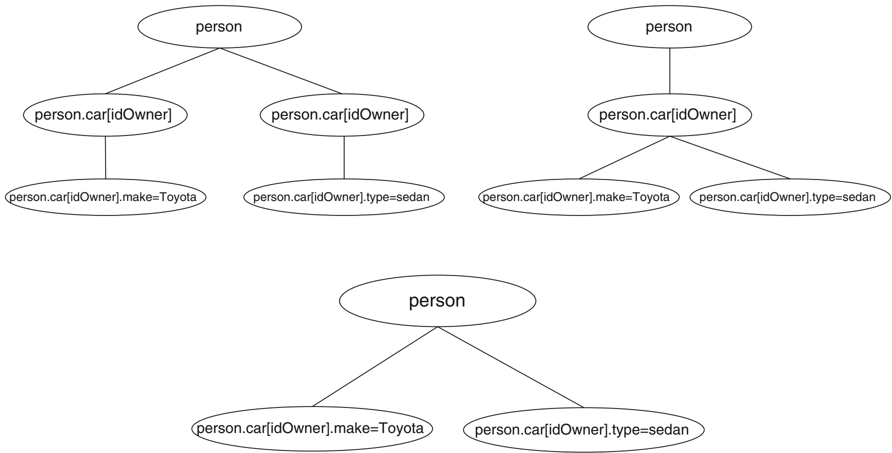

Fig. 12 Embedded and induced patterns from the object-based tree representation of the multirelational database in Fig. 5

of this kind of pattern is shown in Fig. 11b. That pattern indicates that people living in suburbs with two cars ($n = 2$ in this example) are frequent in our database, without any references to particular car features. This kind of pattern cannot be identified using the key-based representation scheme, since all the nodes in a key-based tree necessarily involve attribute values.

In the object-based representation scheme, however, the presence of intermediate nodes increases the number of identified patterns and, therefore, the computational effort needed to discover them. Hence, we should only resort to the object-based representation scheme when we are interested in patterns similar to the one in Fig. 11b. Otherwise, the key-based representation scheme provides faster results in the discovery of embedded patterns.

5.4 Relationships between kinds of patterns

Once we have discussed the kind of patterns we can obtain using each representation scheme, we will study the relationships between the different sets of patterns that we can identify within a multirelational database.

First of all, we will define an equivalence relationship between key-based trees and object based-trees when they represent the same information from a logical point of view. Let $prefix$ be a substring $t_1 \dots t_m$ of the string representing the label of a node $t_1 \dots t_n$ where $m \leq n$.

Definition 1 Equivalence between key-based and object-based trees.

We consider two scenarios to define the equivalence relationship between key-based and object-based trees:

(a) When the relation r contains a foreign key that points to the relation s, which might correspond to a one-to-many or a many-to-one relationship, we will say that a key-based tree T_1 =

$$prefix.F^r = K^s$$

$$
\begin{aligned}
&\quad prefix.F^r.A_1^s = a_1^s \uparrow \\
&\quad prefix.F^r.A_2^s = a_2^s \uparrow \ldots \\
&\quad prefix.F^r.A_n^s = a_n^s \uparrow\uparrow
\end{aligned}
$$

is *equivalent* ($=_{eq}$) to the object-based tree T_3=

$$
\begin{aligned}
&prefix.F^r \\
&\quad prefix.F^r.K^s = k^s \uparrow \\
&\quad prefix.F^r.A_1^s = a_1^s \uparrow \\
&\quad prefix.F^r.A_2^s = a_2^s \uparrow \ldots \\
&\quad prefix.F^r.A_n^s = a_n^s \uparrow\uparrow
\end{aligned}
$$

(b) When a foreign key in s points to r, we will say that a key-based tree T_2 =

$$
\begin{aligned}
&prefix.s[F^s].K^s = k^s \\
&\quad prefix.s[F^s].A_1^s = a_1^s \uparrow \\
&\quad prefix.s[F^s].A_2^s = a_2^s \uparrow \ldots \\
&\quad prefix.s[F^s].A_n^s = a_n^s \uparrow\uparrow
\end{aligned}
$$

is *equivalent* ($=_{eq}$) to the object-based tree T_4=

$$
\begin{aligned}
&prefix.s[F^s] \\
&\quad prefix.s[F^s].K^s = k^s \uparrow \\
&\quad prefix.s[F^s].A_1^s = a_1^s \uparrow \\
&\quad prefix.s[F^s].A_2^s = a_2^s \uparrow \ldots \\
&\quad prefix.s[F^s].A_n^s = a_n^s \uparrow\uparrow
\end{aligned}
$$

Definition 2 Equivalence inclusion for sets of tree patterns.

We say that $A \subseteq_{eq} B$ if and only if, for each element $a \in A$, there exists an element $b \in B$ such that $a =_{eq} b$.

As we have seen before, we can identify four different sets of patterns using either induced or embedded subtrees with both representation schemes: induced key-based patterns (IK), embedded key-based patterns (EK), induced object-based patterns (IO), and embedded object-based patterns (EO). We can identify some relationships among those four sets of patterns by using the following list of properties, whose proofs can be found in the Appendix:

1. All induced key-based patterns are embedded key-based patterns, i.e.,
   Induced key-based patterns $\subseteq$ Embedded key-based patterns.
2. All induced object-based patterns belong to the set of embedded object-based pattern, i.e.,
   Induced object-based patterns $\subseteq$ Embedded object-based patterns.
3. Every induced key-based pattern is equivalent to one pattern that belongs to the set of induced object-based patterns, i.e.,
   Induced key-based patterns $\subset_{eq}$ Induced object-based patterns.

4. Every embedded key-based pattern is equivalent to one pattern that belongs to the set of induced object-based patterns, i.e.,
   Embedded key-based patterns $\subset_{eq}$ Induced object-based patterns.
5. Every embedded key-based pattern is equivalent to one pattern that belongs to the set of embedded object-based patterns., i.e.,
   Embedded key-based patterns $\subset_{eq}$ Embedded object-based patterns.
6. Every induced key-based pattern is equivalent to one pattern that belongs to the set of embedded object-based patterns, i.e.,
   Induced key-based patterns $\subset_{eq}$ Embedded object-based patterns.

In short, we can summarize the relationships among induced key-based, embedded key-based, induced object-based, and embedded object-based patterns as:

$$IK \subseteq EK \subset_{eq} IO \subseteq EO$$

6 Extracting association rules from tree patterns

An association rule is defined as follows for transactional databases: Let $I = i_1, i_2, \ldots, i_m$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. We say that a transaction T contains X, a set of some items in I, if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$ (Agrawal and Srikant 1994).

In multirelational databases, we define D as a set of trees. We say that a tree contains X when X is a subtree of T. Let P be a frequent pattern in D. An association rule in a multirelational database is an implication of the form $X \Rightarrow P$, where X is a subtree of P.

6.1 Tree rules

In this section, we explain how to obtain association rules from frequent tree patterns. The kind of rules we obtain (and, therefore, the knowledge they provide us) will depend on the kind of patterns we mine and the tree-based representation scheme we choose, as we analyzed in Sects. 5.2 and 5.3.

In association rule mining, once all frequent itemsets are identified, we iterate through all their subitemsets in order to enumerate all potentially-interesting association rules. For each subitemset S of a frequent itemset I of size n, a rule $S \Rightarrow I - S$ can be obtained.

When dealing with tree patterns, we have to enumerate all the subtrees of each frequent tree pattern:

– If we are working with embedded patterns, its embedded subtrees can be obtained by repeatedly removing nodes from the original pattern (all but the root node because, if we removed the root node, we would break the tree).

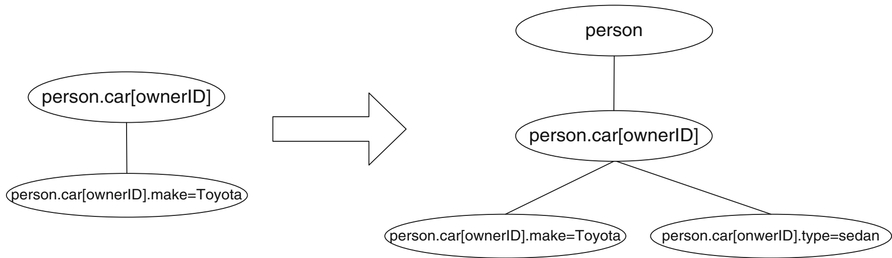

Fig. 13 Rule obtained from the second induced pattern in Fig. 12

– In the case of induced patterns, only leaf nodes can be removed in the process, since we must guarantee that the resulting subtree is also induced.

It should be noted that, when working with tree patterns, we cannot represent rules as in transactional databases ($S \Rightarrow I - S$) because:

– $I - S$ may not be a tree (for example, when S includes the root node), and
– we may not know how to match the S subtree with the $I - S$ subtree (in a similar vein to our discussion on matching trees in Fig. 12a).

Therefore, we will represent the whole pattern I in the consequent of the association rule, albeit its meaning is still analogous to $I - S$ in the traditional sense, since the presence of the pattern I involves the presence of the pattern $I - S$. This way, we obtain rules where both the antecedent and the consequent are trees. Figure 13 shows an example rule that can be obtained from the second induced pattern in Fig. 12.

6.2 Rule mining constraints

The number of rules that can be obtained from a multirelational database can be huge and most of those rules might not be useful for the end user. In this section, we study how to apply constraints (Pei and Han 2002) to tree rules in order to reduce the number of rules to be considered.

6.2.1 Rule-specific constraints

Rule-specific constraints (Bayardo 2004) are based on the measures employed to determine the interestingness of an association rule, like confidence, lift, or certainty factor (Berzal et al. 2002).

A threshold can be established for one or several of these interestingness measures to reduce the number of resulting rules. This constraint lets us reject those rules that do not reach the established thresholds during the rule generation process. Accepting only those rules with confidence above 0.7 (i.e. 70%) is a typical example of this kind of constraint.

6.2.2 Item constraints

An item constraint specifies which groups of items must be present (or not) in the patterns we are looking for. They are typically used to prune the rules so that we obtain only those rules that have a predefined attribute (or attribute value) in their consequent. The use of these constraints in conjunction with length constraints, for instance, is common in associative classification models (Berzal et al. 2004).

Item constraints can also be employed to prune the set of candidate patterns in Apriori-based pattern mining algorithms. When we know which items must be present in every pattern, we can discard the patterns that do not contain the required items (Srikant et al. 1997).

When we apply item constraints at the end of the rule mining process, we can also reduce the number of rules by specifying the items to be present in the antecedent or the consequent of the rules. For example, in order to obtain rules involving the make of a car from the multirelational database in Fig. 4, we could prune the rule set by considering only those rules that have the attribute `person[ownerID].car.make` in their consequent.

A special case of item constraints does not only consider one item but rather its relationships to other items: model-based constraints. These special constraints guide our search for patterns that are sub or super-patterns of some given patterns. For instance, in the database from Fig. 5, we could look for frequent patterns related to Toyota sedan cars.

6.2.3 Length constraints

A length constraint specifies the number of items in the patterns. It can also be applied, in a more restrictive way, by specifying the number of elements in the consequent of the resulting rules. Length constraints can be useful, for instance, when association rules are used for building classification models, where they should have a single element in their consequent.

Maximum length constraints can be applied during the pattern mining phase to reduce the number of generated patterns. Antecedent or consequent maximum length constraints, however, can only be applied during the rule generation process.

7 Experimental results

In this section, we present some experimental results that we have obtained using both the key-based and the object-based tree representations schemes. In these experiments, we have used both synthetic and actual datasets to study the feasibility and performance of our approach to multirelational database mining.

The synthetic datasets have been created using the database generator provided by Yin at his web page: http://research.micro\discretionary-soft.com/en-us/people/xyin/. The parameters of his generator are the number of relations in the database (r), the expected number of tuples in each relation (t), and the expected number of foreign keys in each relation (f). Given particular values for those parameters, the generator

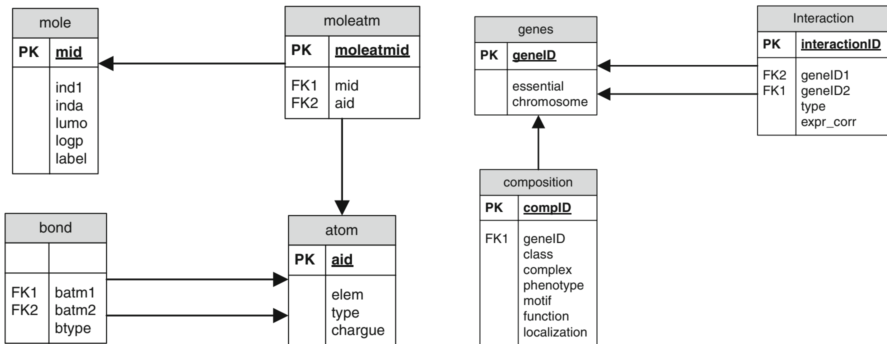

Fig. 14 Schemata of the mutagenesis (*left*) and genes (*right*) databases

produces a relation schema of r relations, one of them being the target relation. The number of attributes in each relation obeys an exponential distribution whose expectation is 5, with a minimum of 2 attributes. The number of different values for each attribute obeys an exponential distribution whose expectation is 10, with a minimum of two different attribute values. The number of foreign keys in each relation also obeys an exponential distribution, with expectation f. Finally, the target relation has exactly t tuples and the number of tuples in each nontarget relation obeys an exponential distribution with expectation t and a minimum of 50 tuples (Yin et al. 2004).

For the experiments with actual datasets, we have used three multirelational databases: `mutagenesis`, `loan`, and `genes`. Figures 14 and 15 depict their schemata.

The `mutagenesis` database (Srinivasan et al. 1994) is a frequently-used ILP benchmark. It contains four relations and 15,218 tuples. The target relation (`mole`) contains 188 tuples.

The `loan` database was used in the PKDD CUP'09. The database contains eight relations with 75,982 tuples in total. The target relation (`loan`) contains 400 tuples.

The `mutagenesis` and `loan` databases were adapted by Yin et al. for their experiments with CrossMine (Yin et al. 2004) and can be downloaded from: http://research.micro\discretionary-soft.com/en-us/people/xyin/.

The `genes` database was proposed in the KDD CUP'01. This database can be used to predict the function of genes, thus we have chosen `function` as its class label for our experiments. The target table in the `genes` database is `composition`, which includes 4,636 tuples.

The original `genes` database contained only two relations called `genes` and `interaction`, but the `genes` relation was not normalized, i.e., the table contained 862 different genes but there were several rows in the table for some genes. The normalization of the database was achieved, as proposed by Leiva et al. (2002), by creating two tables as follows: attributes in the `genes-relation` table that did not have unique values for each gene were placed in the `composition` table and the rest of the attributes were placed in the `gene` table. The `gene_id` attribute is the primary key in the `gene` table and a foreign key in the `composition` table. The `interaction` relation has not changed with respect to the original one.

Fig. 15 Schema of the loan database

The difference between our normalization and the one proposed by Leiva et al. is that we found that the attribute `localization` has two different values for the gene with `gene_id`=G239017. Therefore, we have placed this attribute in the `composition` table instead of keeping it in the `genes` table.

Using the proposed tree-based representation schemes for multirelational databases, tree databases can be obtained from each multirelational database. When the exploration depth is 2, we obtain the following tree databases from the original multirelational databases:

- The `mutagenesis` tree database contains 188 trees. Trees have an average of 163 nodes using the key-based representation scheme (30,674 nodes in total) and an average of 189 nodes per tree using the object-based one (35,567 nodes in total).
- The tree database obtained from the `loan` database contains 400 trees. Trees have an average of 166 nodes using the key-based representation scheme (66,466 nodes in total) and an average of 186 nodes using the object-based one (75,785 nodes in total).

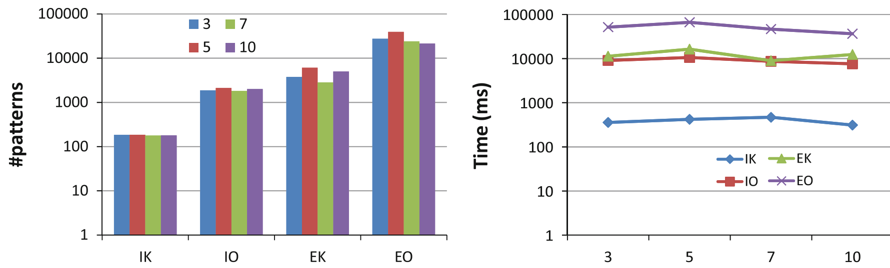

Fig. 16 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the number of relations in the synthetic database

– The tree dtabase obtained from the `genes` database contains 4,636 trees. Trees have an average of 113 nodes using the key-based representation scheme (491,315 nodes in total) and an average of 127 nodes using the object-based one (555,192 nodes in total).

7.1 Identifying induced and embedded patterns

In this section, we present the experiments we have performed to identify frequent patterns in both synthetic and actual databases. We have used the POTMiner tree pattern mining algorithm (Jimenez et al. 2010b) to discover induced and embedded subtrees from the tree-based representation of multirelational databases.

7.1.1 Identifying induced and embedded patterns in synthetic databases

In order to test the performance and scalability of our algorithm, we have performed some series of experiments by varying the parameters of our algorithm (exploration depth, minimum support, and maximum pattern size), as well as the parameters of the synthetic multirelational databases themselves (i.e., number of relations, number of tuples, and number of foreign keys). We have used the following base configuration: #relations = 5, #tuples = 200, #foreign keys = 1, exploration depth = 2, support = 10%, and maxsize (maximum pattern size) = 4.

For each experiment series, we graphically depict the number of identified patterns (up to maxsize), both induced (I) and embedded (E), for each representation scheme, i.e. key-based (K) and object-based (O). We also indicate POTMiner execution time for each case. It should be noted that a logarithmic scale has been used for the Y-axis in all the figures within this section.

First, we have performed some experiments by varying the number of relations: four databases were created with 3, 5, 7, and 10 relations, respectively. Figure 16 shows the results obtained in these experiments. As it can be seen, the number of identified patterns, as well as POTMiner execution time, is more or less independent of the number of relations. The variation in the number of relations does not affect POTMiner execution time because we are not changing the number of trees, which is constant, nor their average size.

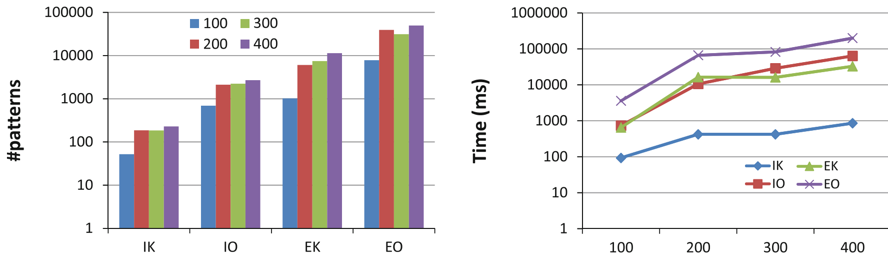

Fig. 17 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the number of tuples in the relations of the synthetic database

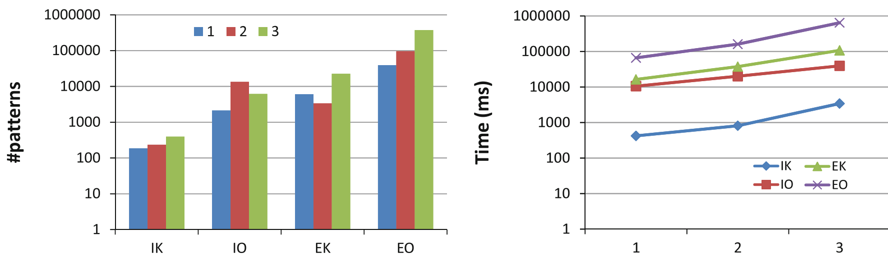

Fig. 18 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the number of foreign keys in the relations of the synthetic database

We have also performed some experiments by varying the number of tuples in each relation: four databases were created with 100, 200, 300, and 400 tuples, respectively. As we increase the number of tuples, while keeping the number of attribute values constant, the number of frequent patterns exponentially increases as shown in Fig. 17. It should also be noted that the number of trees depends on the number of tuples in the target relation of the multirelational database and POTMiner execution time is asymptotically linear with respect to the number of trees in the tree database (Jimenez et al. 2010b).

In our experiments varying the number of foreign keys in each relation, shown in Fig. 18, three databases were created with 1, 2, and 3 expected foreign keys for each relation. When increasing the number of foreign keys in each relation, tree size increases and more patterns are identified. Then, as in a more traditional frequent pattern mining setting, POTMiner time is proportional to the number of identified patterns.

With respect to the parameters of our algorithm, we first vary the exploration depth (see Sect. 4.1). We have generated four databases with exploration depths 1, 2, 3, and 4. As shown in Fig. 19, the number of identified patterns increases exponentially when we increase the exploration depth because the size of the trees is also increased (we are including information about more relations within them). As in the previous experiment involving the number of foreign keys, POTMiner execution time is proportional to the number of examined patterns (and, hence, exponential with respect to the exploration depth, since the number of patterns exponentially grows with respect to the exploration depth).

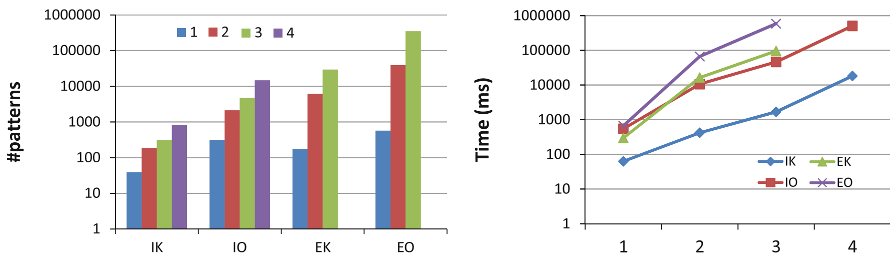

Fig. 19 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the exploration depth in the synthetic database

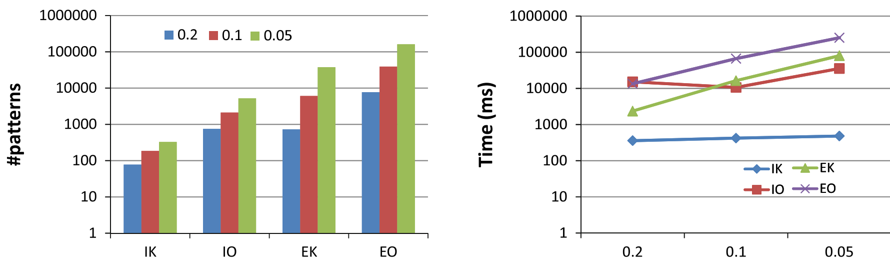

Fig. 20 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the minimum support threshold for the frequent patterns in the synthetic database

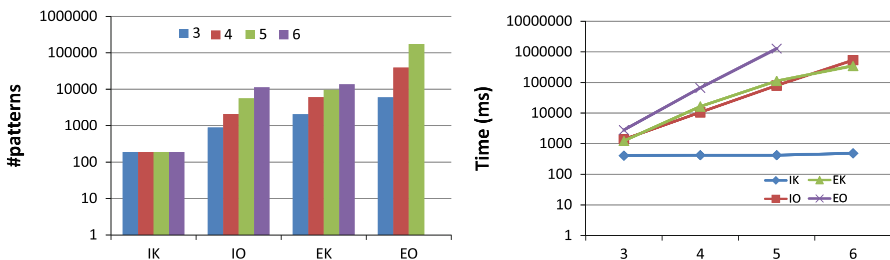

Fig. 21 Number of identified patterns (*left*) and POTMiner execution time (*right*) when varying the maximum size of the identified patterns within the synthetic database

The results of our experiments varying the minimum support of frequent patterns are shown in Fig. 20. We have extracted patterns using 20, 10, and 5% minimum support thresholds. As expected, the number of identified patterns increases when the support decreases and POTMiner execution time is proportional to the increase in the number of identified patterns.

Finally, we have performed a series of experiments by varying the maximum size of the identified patterns. Figure 21 shows the results of these experiments including the number of patterns up to maxsize, which goes from 3 to 6. It should be noted that, in the case of induced key-based patterns, no patterns of size greater than 1 were identified. In the remaining cases, the number of identified patterns and, consequently, POTMiner execution time exponentially increases with the pattern size.

POTMiner execution time is linear with respect to the number of trees in the database and also proportional to the number of identified patterns (Jimenez et al. 2010b). In the case of multirelational databases, the number of tuples in the target relation determines the number of trees in the database, hence POTMiner is linear with respect to the number of tuples in the target relation. Therefore, our algorithm is asymptotically optimal for the problem of mining frequent patterns from multirelational databases using tree-based representation schemes.

However, as in the classical frequent pattern mining problem, the number of identified patterns within a multirelational database can be exponential. The number of foreign keys, the exploration depth, the minimum support threshold, and the maximum pattern size can generate a combinatorial explosion in the number of frequent patterns. Since POTMiner execution time is proportional to the number of identified patterns (Jimenez et al. 2010b), our algorithm execution time can be exponential with respect to the aforementioned parameters. Care should be taken while setting the right values for those parameters in a real-world situation.

7.1.2 Identifying induced and embedded patterns in actual databases

In our experiments with three actual databases (`mutagenesis`, `loan`, and `genes`), we have identified induced and embedded patterns including up to six nodes, i.e., $maxsize = 6$ in POTMiner.

Figure 22 shows the number of induced patterns discovered using different minimum support thresholds for the three datasets, each one obtained with exploration depths 1, 2, and 3, for both the key-based (top) and the object-based (bottom) tree representation schemes.

It should be noted that the number of key-based induced patterns is low and, in most cases, only patterns of size 1 are identified. This is due to the use of primary keys as internal nodes within the trees, which are rarely frequent, as we mentioned in Sect. 5.3.

The number of embedded frequent patterns in the three databases is shown in Fig. 23. The number of discovered patterns using the object-based tree representation scheme is larger than the number of identified patterns using the key-based representation scheme. This is mainly due to the use of intermediate nodes to represent each tuple from the database and the fact that these intermediate nodes are usually frequent.

Figure 24 shows two patterns that have been identified in those databases. The induced object-based pattern at the top is from the `genes` database: 21% of the genes in this database are considered to be *Non-Essential* yet they are involved in *Physical* interactions. The embedded object-based pattern at the bottom of Fig. 24 is from the `mutagenesis` database: 47% of the molecules in this database include, at least, three oxygen atoms (in this particular database, all chemical compounds include at least two).

Figure 25 compares the time required to identify induced and embedded patterns using the object-based representation scheme. As we explained in the previous section, POTMiner execution time is proportional to the number of patterns that are examined (Jimenez et al. 2010b). The discovery of induced patterns is faster than the discovery of embedded patterns because there is a lower number of them. Likewise, mining

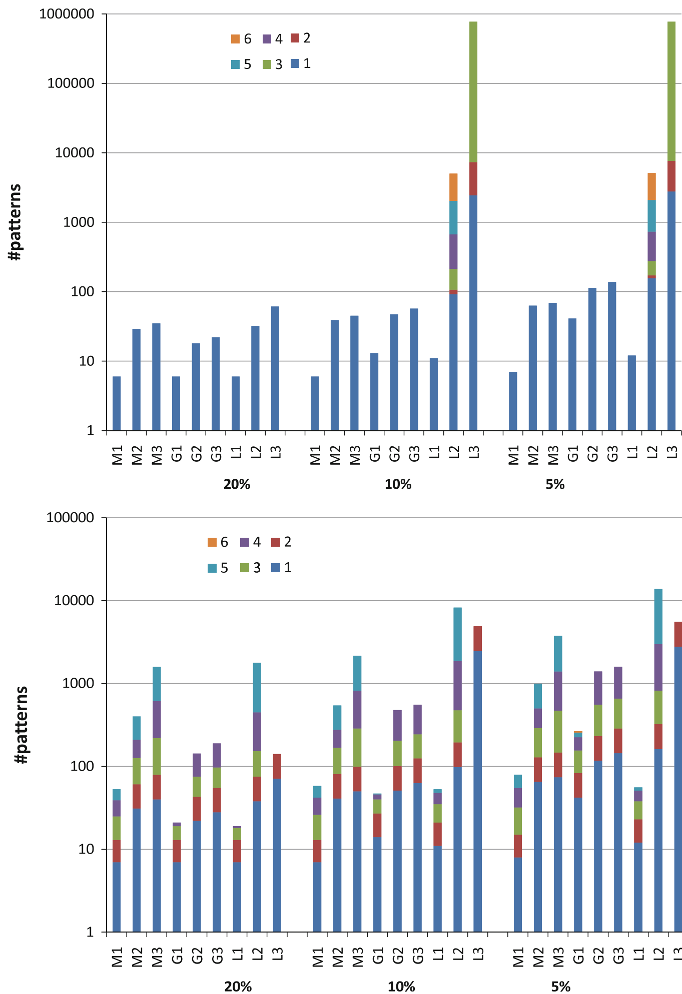

Fig. 22 Number of induced patterns using the key-based (*top*) and the object-based (*bottom*) representation schemes for the muta (M), loan (L), and genes (G) databases

object-based patterns requires more execution time because a greater number of patterns is considered, as we analyzed in Sect. 5.2.

7.2 Extracting rules from frequent tree patterns

In these series of experiments, we have identified induced and embedded frequent patterns of size 4 using both the key-based and the object-based representation schemes, exploration depths from 1 to 3, and a 10% minimum support threshold. We have then used those frequent patterns to extract association rules by varying the minimum confidence threshold (using 0.7, 0.8, and 0.9 as threshold values).

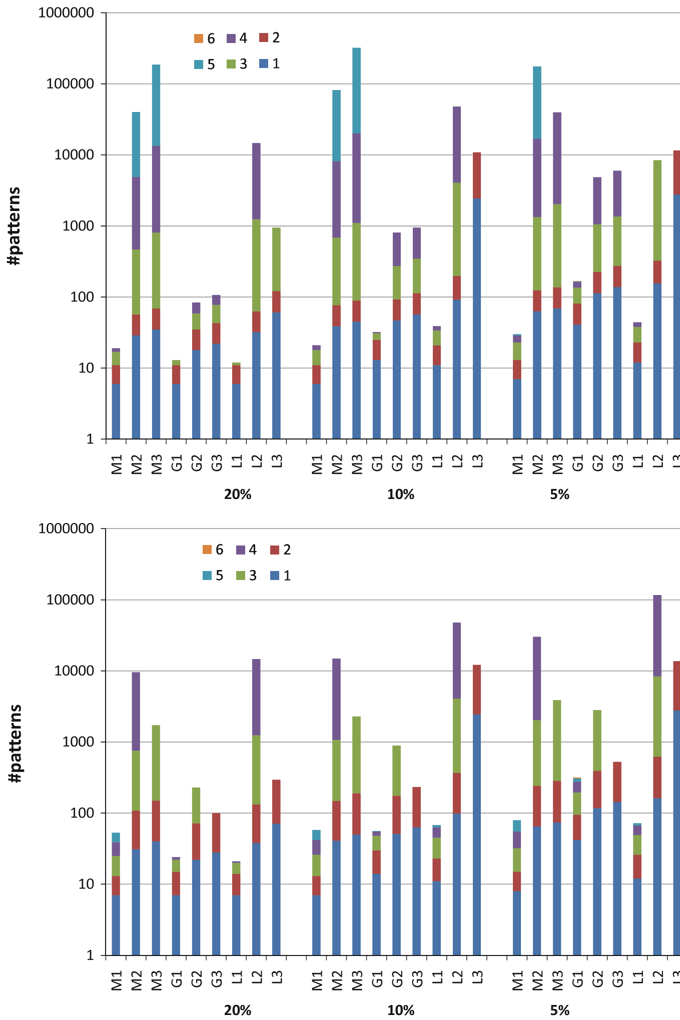

Fig. 23 Number of embedded patterns using the key-based (*top*) and the object-based (*bottom*) representation schemes for the muta (M), loan (L), and genes (G) databases

7.2.1 Extracting rules from frequent tree patterns in synthetic databases

In these experiments, we have used the patterns identified in the synthetic database using the base configuration described in the previous section.

Figure 26 shows the number of discovered rules and highlights the rules involving the `class` attribute in the target relation. We compare the number of rules that include the `class` attribute in their consequent with respect to the total number of discovered rules. The `only class` value shows the number of rules that have the `class` attribute as their unique consequent with respect to the number of rules that include the status attribute in their consequent but not necessarily alone. These constraints, as we

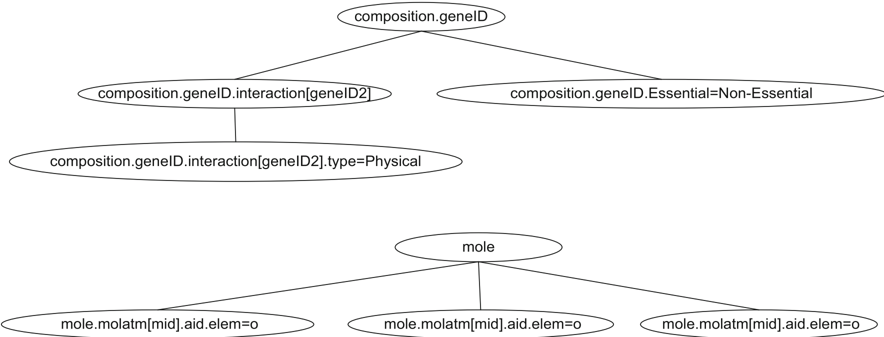

Fig. 24 Patterns identified in the genes (*top*) and mutagenesis (*bottom*) databases

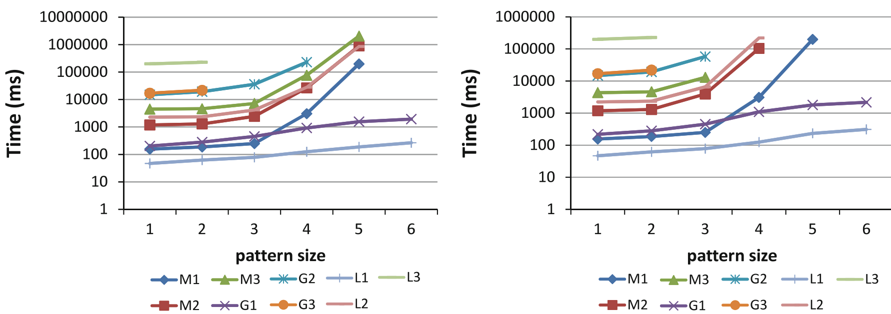

Fig. 25 POTMiner execution time for identifying induced (*left*) and embedded (*right*) patterns using the object-based representation scheme for the muta (M), loan (L), and genes (G) databases

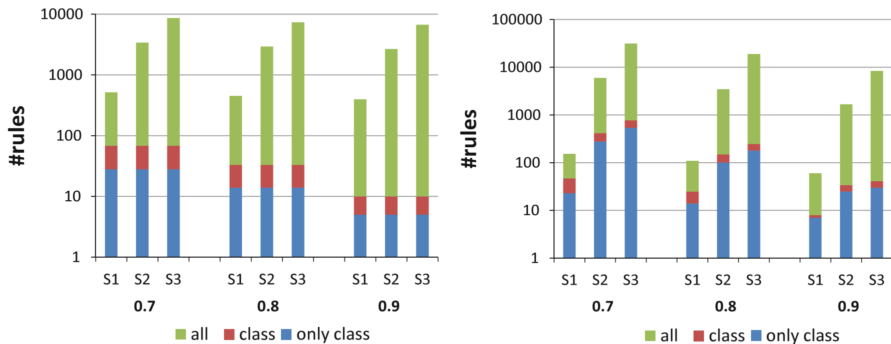

Fig. 26 Number of association rules from the synthetic database using induced object-based patterns (*left*) and embedded key-based patterns (*right*) with different minimum confidence thresholds

explained in Sect. 6.2, are useful when we use association rules to build classification models.

The number of rules obtained from induced patterns in the object-based representation is similar for all the exploration depths in the synthetic datasets. Since induced patterns preserve the original structure of the database trees, the discovered patterns of size 4 are more or less the same regardless of the chosen exploration depth and,

therefore, the resulting rules are also similar. No rules were obtained from induced key-based patterns because all of them were of size 1. More rules were obtained from embedded patterns than from induced patterns, since embedded patterns are a superset of induced patterns.

Albeit not shown in the figures, it should be noted that, once the frequent patterns are identified, the process of extracting rules from them is very fast, just a few seconds for a complete multirelational database.

7.2.2 *Extracting rules from frequent tree patterns in actual databases*

In our experiments with actual datasets, we have extracted rules from the `mutagenesis`, `loan`, and `genes` databases. We have considered `label`, `status`, and `function`, respectively, as their class attribute.

Figure 27 shows the number of rules resulting from the induced object-based patterns and the embedded key-based patterns derived from the actual databases. Albeit not shown in the figure, as happened with the synthetic datasets, no rules were obtained from the induced key-based patterns for the `mutagenesis` and `genes` databases because no pattern of size greater than 1 was obtained (see Sect. 7.1). Only a few rules, which did not contain the class attribute, were obtained from the induced key-based representation of the `loan` database.

Figure 28a depicts one rule obtained from induced object-based patterns in the `loan` database. This rule, with 65% support and 82% confidence, can be interpreted as: “If we have a `loan` whose associated `account` has a monthly `frequency` of issuance of statements, then the `loan` usually has no problems (i.e. `status`=1)”. In other words, of all the loans in our database that have an associated `account` with a monthly `frequency` of issuance of statements, 82% of them have no problems.

Rules derived from embedded patterns provide information that cannot be obtained from the induced patterns. For instance, Fig. 28b shows a rule from the embedded key-based patterns in the `loan` database. This rule, with 20% support and 82% confidence, states that “if we have a `loan` whose `account` has a `permanent order` of *house* type and its district has one municipality with more than 10000 inhabitants (`num_gt_10000` = 1), then the `loan` usually has no problems (`status`=1)”. Hence, it is less frequent in our database to find loans without problems associated to accounts in a district with more than 10,000 inhabitants with a `permanent order` of *house* type than loans without problems that have an associated account with monthly `frequency`. However, our confidence about not having problems with those loans is the same in both situations.

The number of association rules from object-based patterns is 3–10 times higher than the number of rules derived from key-based patterns. Since, every embedded key-based pattern has an equivalent embedded object-based pattern, as discussed in Sect. 5.4, all embedded key-based rules have their counterpart in the set of embedded object-based rules. Therefore, a rule equivalent to the rule in Fig. 28b will be found in the set of embedded object-based rules.

Object-based rules will be specially useful if we are interested in the kind of knowledge we described with the example in Fig. 11 from Sect. 5.1.3. In our experiments, we have obtained a similar rule from the `loan` database, as shown in Fig. 28c: “Within

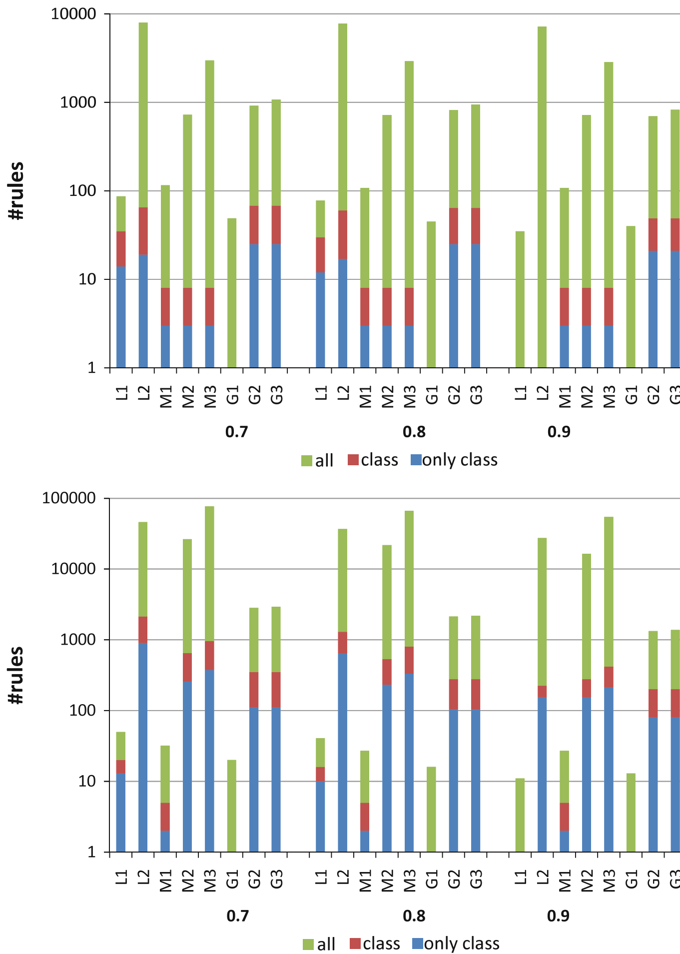

Fig. 27 Number of association rules from the mutagenesis, loan, and genes databases using induced object-based patterns (*top*) and embedded key-based patterns (*bottom*) with different minimum confidence thresholds

the set of `accounts` with an associated `loan` and two `orders`, 83% of them have a monthly `frequency` of issuance of statements" (support=67%, confidence=83%).

8 Conclusions

This paper proposes a new approach to mine multirelational databases. Our approach is based on representing multirelational databases as sets of trees.

We have designed two alternative tree representation schemes for multirelational databases. The main idea behind both of them is building a tree representing each tuple in the target table (i.e., the most interesting table for the user) by following the foreign keys that connect tables in the relational databases.

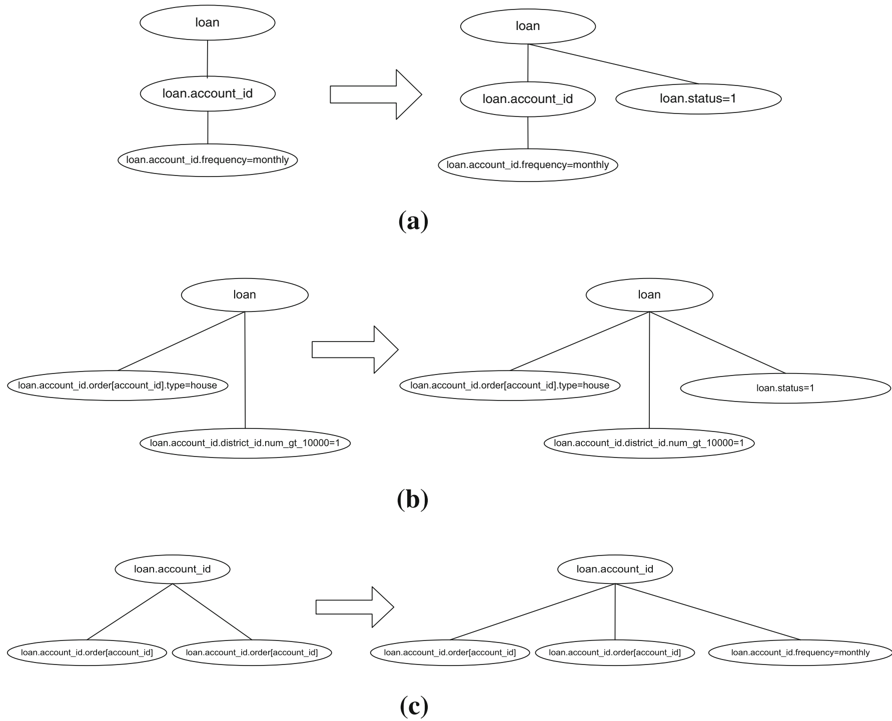

Fig. 28 Some rules obtained during our experiments

The key-based representation scheme uses primary keys at the roots of the subtrees representing each tuple. In contrast, the object-based representation scheme uses generic intermediate nodes to include new tuples in the trees.

We have identified frequent patterns in the trees derived from multirelational databases and we have studied the differences that arise from identifying induced or embedded patterns in both the key-based and the object-based representation schemes. As we explain in Sect. 5.4 and prove in the appendix, every key-based pattern is equivalent to a pattern that belongs to the set of object-based patterns.

We have also described how tree patterns can be employed to discover association rules from multirelational databases. The performance of this rule mining process can be improved by the use of constraints, which are also useful for reducing the number of rules that are returned to the user.

Our experiments with synthetic and actual datasets show that our approach is feasible in practice, since it can rest on our previous tree pattern mining algorithm, POTMiner, which is linear with respect to the number of trees in the database and whose execution time is proportional to the number of examined patterns, an asymptotically optimal solution for the frequent tree pattern mining problem.

The discovery of induced patterns combined with the object-based representation scheme is often enough to mine multirelational databases. Embedded patterns, when used with the key-based representation scheme, let us reach data that is farther from

the target relation, although such patterns do not always preserve the structure of the original database trees and can introduce some ambiguities in their interpretation. Embedded object-based patterns, the most general case we have studied, can discover situations where the number of related tuples is important, regardless of their particular attribute values.

Acknowledgments We would like to thank the anonymous referees for their valuable comments and suggestions, which gave us the chance to improve the quality of this manuscript. This work has been partially supported by research projects TIN2006-07262 (Spanish Ministry of Science and Innovation), TIN2009-08296 (Spanish Ministry of Science and Innovation), and P07-TIC-03175 (Junta de Andalucía).

Appendix: Relationships between kinds of patterns

Property 1 *All induced key-based patterns are embedded key-based patterns, i.e.,*
Induced key-based patterns $\subseteq$ Embedded key-based patterns.

Proof (a) $\forall p \in IK \Rightarrow p \in EK$. By definition, an induced pattern is an embedded pattern where all the parent–children relationships are preserved. Therefore, $\forall p \in IK$, $p \in EK$, i.e., $IK \subseteq EK$.

(b) For patterns of size ≤ 2, $EK = IK$. All key-based patterns of size ≤ 2 have to preserve their only parent–children relationship, when it exists. Therefore $\forall q \in EK$ of size$(q) \leq 2$, $q \in IK$, i.e., $EK = IK$.

(c) For patterns of size > 2, $EK \not\subset IK$. Let $q \in EK$ be an embedded key-based pattern that does not preserve all the parent–children relationships. Then, $q \in EK$ but $q \notin IK$. Therefore, $q \in EK \not\Rightarrow q \in IK$. Hence, $EK \not\subset IK$.

Therefore, $IK \subseteq EK$. □

Property 2 *All induced object-based patterns belong to the set of embedded object-based pattern, i.e.,*
Induced object-based patterns $\subseteq$ Embedded object-based patterns.

Proof Analogous to the proof of Property 1. Therefore, $IO \subseteq EO$. □

Property 3 *Every induced key-based pattern is equivalent to one pattern that belongs to the set of induced object-based patterns, i.e.,*
Induced key-based patterns $\subset_{eq}$ Induced object-based patterns.

Proof (a) $IK \subset IO$: $\forall p \in IK \Rightarrow \exists p' \in IO$, $p =_{eq} p'$. By induction on the number of foreign keys in the tree:

1. Suppose that the pattern p does not contain any foreign keys, i.e., $\#FK(p) = 0$. Then, $p \in IK$ is of the form

$$
\begin{array}{l}
r \\
\quad r.K^r = k^r \\
\quad\quad r.A_1^r = a_1^r \uparrow \ldots \\
\quad\quad r.A_n^r = a_n^r \uparrow\uparrow\uparrow \ldots
\end{array}
$$

and $p' \in IO$ is of the form

r

$r.K^r = k^r \uparrow$
$R.A_1^r = a_1^r \uparrow \ldots$
$r.A_n^r = a_n^r \uparrow\uparrow \ldots$

Therefore, by Definition 1, $p =_{eq} p'$.

2. Suppose that $q \in IK$ contains k foreign keys, i.e., $\#FK(q) = k$, and $\exists q' \in IO, q =_{eq} q'$. Let q be a subtree of p, which contains $k+1$ foreign keys; i.e., $\#FK(p) = k+1$. Then, we have to consider two scenarios depending on the kind of foreign key added to q in order to obtain the pattern p.
   (a) If the foreign key is in the relation r pointing to another relation s, then a subtree of the form T_1 will be added to a node with label $prefix$ in q in order to generate the pattern p. Let $p' \in IO$ be a pattern generated by adding a subtree with the form of T_3 to the pattern q' at the node with label $prefix$. As $T_1 =_{eq} T_3, q =_{eq} q'$, and T_1 is attached to q at the same node where T_3 is attached to q', we have $p =_{eq} p'$.
   (b) If the foreign key is in the relation s pointing to our relation, then a tree of the form T_2 will be added to a node with label $prefix$ in q in order to generate the pattern p. Let $p' \in IO$ be a pattern generated by adding a subtree with the form of T_4 to the pattern q' at the node with label $prefix$. As $T_2 =_{eq} T_4, q =_{eq} q'$, and T_2 is attached to q at the same node where T_4 is attached to q', we have $p =_{eq} p'$.

Therefore, $\forall p \in IK, \exists p' \in IO, p =_{eq} p'$. Hence, $IK \subset IO$.

(b) $IO \not\subset_{eq} IK$. As a counterexample, the special object-based patterns that let us know how many tuples are related to a given one cannot be identified using the key-based representation (as the example shown in Fig. 11a). Hence, $IO \not\subset_{eq} IK$.

Therefore, $IK \subset_{eq} IO$. □

Property 4 *Every embedded key-based pattern is equivalent to one pattern that belongs to the set of induced object-based patterns, i.e.,*
Embedded key-based patterns $\subset_{eq}$ *Induced object-based patterns.*

Proof (a) $EK \subset IO$: $\forall p \in EK \Rightarrow \exists p' \in IO, p =_{eq} p'$.

By induction on the number of foreign keys in the tree:

1. Suppose that the pattern p does not contain any foreign keys, i.e., $\#FK(p) = 0$. Then, $p \in EK$ is of the form

r

$r.K^r = k^r$
$r.A_1^r = a_1^r \uparrow \ldots$
$r.A_n^r = a_n^r \uparrow\uparrow\uparrow \ldots$

and $p' \in IO$ is of the form

r

$r.K^r = k^r \uparrow$

$$r.A_1^r = a_1^r \uparrow \ldots$$
$$r.A_n^r = a_n^r \uparrow\uparrow\uparrow \ldots$$

Therefore, by Definition 1, $p =_{eq} p'$.

2. Suppose that $q \in EK$ contains k foreign keys, i.e., $\#FK(q) = k$, and $\exists q' \in IO, q =_{eq} q'$. Let q be a subtree of p, which contains $k+1$ foreign keys, i.e., $\#FK(p) = k+1$. Then, we have to consider two scenarios depending on the kind of foreign key added to q in order to obtain the pattern p. Furthermore, as p is an embedded pattern, we have to consider whether the subtree f corresponding to the foreign key conserves its root node or note.
   - If the foreign key is in the relation r pointing to another relation s.
     (a) When the subtree f conserves its root node in p (i.e., the node $prefix.A_{FK} = k^s$), then f is of the form T_1 and it will be added to a node with label $prefix$ in q to generate the pattern p. Let $p' \in IO$ be a pattern generated by adding a subtree with the form of T_3 to the pattern q' at the node with the label $prefix$. As $T_1 =_{eq} T_3, q =_{eq} q'$, and T_1 is attached to q at the same node that T_3 is attached to q', then $p =_{eq} p'$.
     (b) When the subtree f has no root node in p (i.e., we add only some leaf nodes with their attribute values), then:

$$prefix.F^r.A_1^s = a_1^s \uparrow$$
$$prefix.F^r.A_2^s = a_2^s \uparrow \ldots$$
$$prefix.F^r.A_n^s = a_n^s \uparrow\uparrow$$

     are the nodes that have been added as children to the node with the label $prefix$ in q to form p. Then, if we add the same leaf nodes as children to the node with label $prefix$ in q', we will obtain a pattern $p' \in IO$. Therefore, $p =_{eq} p'$.
   - If the foreign key is in the relation s pointing to the relation r:
     (a) When the subtree f conserves its root node in p (i.e., the node $prefix.s[F^s].K^s = k^s$), then f is of the form T_2 and it will be added to a node with label $prefix$ in q to generate the pattern p. Let $p' \in IO$ be a pattern generated by adding a subtree with the form of T_4 to the pattern q' at the node with the label $prefix$. As $T_2 =_{eq} T_4, q =_{eq} q'$, and T_2 is attached to q at the same node that T_4 is attached to q', then $p =_{eq} p'$.
     (b) When the subtree f has no root node in p (i.e., we add only some leaf nodes with their attribute values), then:

$$prefix.s[F^s].A_1^s = a_1^s \uparrow$$
$$prefix.s[F^s].A_2^s = a_2^s \uparrow \ldots$$
$$prefix.s[F^s].A_n^s = a_n^s \uparrow\uparrow$$

     are the nodes that have been added as children to the node with the label $prefix$ in q to form p. Then, if we add the same leaf nodes

as children to the node with the label $prefix$ in q', we will obtain a pattern $p' \in IO$. Therefore, $p =_{eq} p'$.

Hence, $\forall p \in EK, \exists p' \in IO, p =_{eq} p'$. Therefore, $EK \subset IO$.

(b) $IO \not\subset_{eq} EK$: As a counterexample, the induced object-based patterns that let us know how many tuples are related to a given one cannot be identified using the key-based representation (as the example shown in Fig. 11a). Hence, $IO \not\subset_{eq} EK$.

Therefore, $EK \subset_{eq} IO$. □

Property 5 *Every embedded key-based pattern is equivalent to one pattern that belongs to the set of embedded object-based patterns., i.e., Embedded key-based patterns* $\subset_{eq}$ *Embedded object-based patterns.*

Proof (a) $EK \subset_{eq} EO$: We have proved that $EK \subset_{eq} IO$ in Property 4 and that $IO \subseteq EO$ in Property 2. Therefore, $EK \subset_{eq} IO \subseteq EO$. By transitivity, $EK \subset_{eq} EO$.

(b) $EO \not\subset_{eq} EK$: As a counterexample, the embedded object-based patterns that let us know how many tuples are related to a given one cannot be identified using the key-based representation (as the example shown in Fig. 11b). Hence, $EO \not\subset_{eq} EK$.

Therefore, $EK \subset_{eq} EO$. □

Property 6 *Every induced key-based pattern is equivalent to one pattern that belongs to the set of embedded object-based patterns, i.e., Induced key-based patterns* $\subset_{eq}$ *Embedded object-based patterns.*

Proof (a) $IK \subset_{eq} EO$: We have proved that $IO \subseteq EO$ in Property 2 and also that $IK \subset_{eq} IO$ in Property 3. Therefore, $IK \subset_{eq} IO \subseteq EO$. By transitivity, $IK \subset_{eq} EO$.

(b) $EO \not\subset_{eq} IK$: As a counterexample, the embedded object-based patterns that let us know how many tuples are related to a given one cannot be identified using the key-based representation (as the example shown in Fig. 11b). Hence, $EO \not\subset_{eq} IK$.

Therefore, $IK \subset_{eq} EO$. □

References

Abe K, Kawasoe S, Asai T, Arimura H, Arikawa S (2002) Efficient substructure discovery from large semi-structured data. In: Proceedings of the 2nd SIAM international conference on data mining, pp 158–174

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases, 12–15 Sept, pp 487–499

Bayardo RJ (2004) The hows, whys, and whens of constraints in itemset and rule discovery. In: Constraint-based mining and inductive databases, lecture notes in artificial intelligence, pp 1–13

Berzal F, Blanco I, Sánchez D, Vila MA (2002) Measuring the accuracy and interest of association rules: a new framework. Intell Data Anal 6(3):221–235

Berzal F, Cubero JC, Sánchez D, Serrano JM (2004) ART: a hybrid classification model. Mach Learn 54(1):67–92

Blockeel H, Raedt LD (1998) Top-down induction of first-order logical decision trees. Artif Intell 101 (1–2):285–297

Booch G, Rumbaugh J, Jacobson I (2005) The unified modeling language user guide, 2nd edn. Addison-Wesley Professional, New York

Chi Y, Yang Y, Muntz RR (2003) Indexing and mining free trees. In: Proceedings of the 3rd IEEE international conference on data mining, pp 509–512

Chi Y, Muntz RR, Nijssen S, Kok JN (2005) Frequent subtree mining—an overview. Fundam Inform 66(1–2):161–198

Codd EF (1990) The relational model for database management, version 2. Addison-Wesley, New York

De Knijf J (2006) FAT-miner: mining frequent attribute trees. Tech. Rep. UU-CS-2006-053, Department of Information and Computing Sciences, Utrecht University

De Knijf J (2007) FAT-miner: mining frequent attribute trees. In: Proceedings of the 2007 ACM symposium on applied computing. ACM, New York, pp 417–422

Džeroski S (2003) Multi-relational data mining: an introduction. SIGKDD Explor Newsl 5(1):1–16

Fagin R, Mendelzon AO, Ullman JD (1982) A simplied universal relation assumption and its properties. ACM Trans Database Syst 7:343–360

Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book. Pearson Education, Boston

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Discov 8(1):53–87

Jimenez A, Berzal F, Cubero JC (2010a) Frequent tree pattern mining: a survey. Intell Data Anal 14(6):603–622

Jimenez A, Berzal F, Cubero JC (2010b) POTMiner: mining ordered, unordered, and partially-ordered trees. Knowl Inform Syst 23(2):199–224

King RD, Srinivasan A, Dehaspe L (2001) Warmr: a data mining tool for chemical data. J Comput-Aided Mol Des 15(2):173–181

Krogel MA, Wrobel S (2003) Facets of aggregation approaches to propositionalization. In: Horvath T, Yamamoto A (eds) Work-in-progress track at the thirteenth international conference on inductive logic programming

Lee AJT, Wang CS (2007) An efficient algorithm for mining frequent inter-transaction patterns. Inform Sci 177(17):3453–3476

Leiva HA, Gadia S, Dobbs D (2002) MRDTL: a multi-relational decision tree learning algorithm. In: Proceedings of the 13th international conference on inductive logic programming. Springer-Verlag, pp 38–56

Maier D, Ullman JD (1983) Maximal objects and the semantics of universal relation databases. ACM Trans Database Syst 8:1–14

Maier D, Ullman JD, Vardi MY (1984) On the foundations of the universal relation model. ACM Trans Database Syst 9:283–308

McGovern A, Hiers NC, Collier M, II DJG, Brown RA (2008) Spatiotemporal relational probability trees: an introduction. In: Proceedings of the 8th IEEE international conference on data mining. IEEE Computer Society, pp 935–940

Neville J, Jensen D, Friedland L, Hay M (2003) Learning relational probability trees. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining, pp 625–630

Paterson J, Edlich S, Hörning H, Hörning R (2006) The definitive guide to db4o. Apress, New York

Pei J, Han J (2002) Constrained frequent pattern mining: a pattern-growth view. SIGKDD Explor Newsl 4(1):31–39

Perlich C, Provost F (2006) Distribution-based aggregation for relational learning with identifier attributes. Mach Learn 62:65–105

Silberschatz A, Korth HF, Sudarshan S (2001) Database systems concepts. McGraw-Hill, New York

Srikant R, Vu Q, Agrawal R (1997) Mining association rules with item constraints. In: Proceedings of the 3rd international conference of knowledge discovery and data mining, pp 63–73

Srinivasan A, Muggleton SH, King R, Sternberg M (1994) Mutagenesis: ILP experiments in a non-determinate biological domain. In: Proceedings of the 4th international workshop on inductive logic programming, vol 237 of GMD-Studien, pp 217–232

Tung AKH, Lu H, Han J, Feng L (2003) Efficient mining of intertransaction association rules. IEEE Trans Knowl Data Eng 15(1):43–56

Turmeaux T, Salleb A, Vrain C, Cassard D (2003) Learning characteristic rules relying on quantified paths. In: Proceedings of the 7th European conference on principles and practice of knowledge discovery in databases, pp 471–482

Ullman JD (1988) Principles of database and knowledge-base systems, vol I: classical database systems. Computer Science Press Inc., New York

Ullman JD (1990) Principles of database and knowledge-base systems, vol II: the new technologies. W. H. Freeman & Co., New York

Wang C, Hong M, Pei J, Zhou H, Wang W, Shi B (2004) Efficient pattern-growth methods for frequent tree pattern mining. In: Proceedings of the 8th Pacific-Asia conference on knowledge discovery and data mining. Lecture Notes in Computer Science, vol 3056, Springer, pp 441–451

Xiao Y, Yao JF, Li Z, Dunham MH (2003) Efficient data mining for maximal frequent subtrees. In: Proceedings of the 3rd IEEE international conference on data mining, pp 379–386

Yin X, Han J, Yang J, Yu PS (2004) CrossMine: efficient classification across multiple database relations. In: Proceedings of the 20th international conference on data engineering, pp 399–410

Yin X, Han J, Yu PS (2005) Cross-relational clustering with user's guidance. In: Proceedings of the 12th international conference on knowledge discovery and data mining, pp 344–353

Zaki MJ (2005a) Efficiently mining frequent embedded unordered trees. Fundam Inform 66(1–2):33–52

Zaki MJ (2005b) Efficiently mining frequent trees in a forest: algorithms and applications. IEEE Trans Knowl Data Eng 17(8):1021–1035