Inductive Querying with Virtual Mining Views

Hendrik Blockeel, Toon Calders, Elisa Fromont, Bart Goethals, Adriana Prado, Céline Robardet

Inductive Querying with Virtual Mining Views

Hendrik Blockeel, Toon Calders, Élisa Fromont, Bart Goethals, Adriana Prado, and Céline Robardet

Abstract In an inductive database, one can not only query the data stored in the database, but also the patterns that are implicitly present in these data. In this chapter, we present an inductive database system in which the query language is traditional SQL. More specifically, we present a system in which the user can query the collection of all possible patterns as if they were stored in traditional relational tables. We show how such tables, or mining views, can be developed for three popular data mining tasks, namely itemset mining, association rule discovery and decision tree learning. To illustrate the interactive and iterative capabilities of our system, we describe a complete data mining scenario that consists in extracting knowledge from real gene expression data, after a pre-processing phase.

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
Leiden Institute of Advanced Computer Science, Universiteit Leiden, The Netherlands e-mail: hendrik.blockeel@cs.kuleuven.be

Toon Calders
Technische Universiteit Eindhoven, The Netherlands e-mail: t.calders@tue.nl

Élisa Fromont · Adriana Prado
Université de Lyon (Université Jean Monnet), CNRS, Laboratoire Hubert Curien, UMR5516, F-42023 Saint-Etienne, France e-mail: {elisa.fromont,adriana.bechara.prado}@univ-st-etienne.fr

Bart Goethals
Universiteit Antwerpen, Belgium e-mail: bart.goethals@ua.ac.be

Céline Robardet
Université de Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621, France e-mail: celine.robardet@insa-lyon.fr

1 Introduction

Data mining is an interactive process in which different tasks may be performed sequentially. In addition, the output of those tasks may be repeatedly combined to be used as input for subsequent tasks. For example, one could (a) first learn a decision tree model from a given dataset and, subsequently, mine association rules which describe the misclassified tuples with respect to this model or (b) first look for an interesting association rule that describes a given dataset and then find all tuples that violate such rule.

In order to effectively support such a knowledge discovery process, the integration of data mining into database systems has become necessary. The concept of *Inductive Database Systems* has been proposed in [1] so as to achieve such integration. The idea behind this type of system is to give to the user the ability to query not only the data stored in the database, but also patterns that can be extracted from these data. Such database should be able to store and manage patterns as well as provide the user with the ability to query them.

In this chapter, we show how such an inductive database system can be implemented in practice, as studied in [2, 3, 4, 5, 6, 7]. To allow the users to query patterns as well as standard data, several researchers proposed extensions to the popular query language SQL as a natural way to express such mining queries [8, 9, 10, 11, 12, 13]. As opposed to those proposals, we present here an inductive database system in which the query language is traditional SQL. We propose a relational database model based on what we call *virtual mining views*. The mining views are relational tables that virtually contain the complete output of data mining tasks. For example, for the itemset mining task, there is a table called *Sets* virtually storing all itemsets. As far as the user is concerned, all itemsets are stored in table *Sets* and can be queried as any other relational table. In reality, however, table *Sets* is empty. Whenever a query is formulated selecting itemsets from this table, the database system triggers an itemset mining algorithm, such as Apriori [14], which computes the itemsets in the same way as normal views in databases are only computed at query time. The user does not notice the emptiness of the tables; he or she can simply assume their existence and query accordingly. Therefore, we prefer to name these special tables virtual mining views.

In this chapter, we show how such tables, or virtual mining views, can be developed for three popular data mining tasks, namely itemset mining, association rule discovery and decision tree learning. To make the model as generic as possible, the output of these tasks are represented by a unifying set of mining views. In Section 2, we present these mining views in detail.

Since the proposed mining views are empty, they need to be filled (materialized) by the system once a query is posed over them. The mining process itself needs to be performed by the system in order to answer such queries. Note that the user may impose certain constraints in his or her queries, asking for only a subset of all possible patterns. As an example, the user may

query from the mining view *Sets* all frequent itemsets with a certain support. Therefore, the entire set of patterns does not always need to be stored in the mining views, but only those that satisfy the constrains imposed by the user. In [2], Calders et al. present an algorithm that extracts from a query a set of constraints relevant for association rules to be pushed into the mining algorithm. We have extended this constraint extraction algorithm to extract constraints from queries over decision trees. The reader can refer to [7] for the details on the algorithm.

All ideas presented here, from querying the mining views and extracting constraints from the queries to the actual execution of the data mining process itself and the materialization of the mining views, have been implemented into the well-known open source database system PostgreSQL[1]. Details of the implementation are given in Section 3.

We have therefore organized the rest of this chapter in the following way. The next section is dedicated to the virtual mining views framework. We also present how the 4 prototypical tasks described in the previous chapter can be executed by SQL queries over the mining views. The implementation of the system along with an extended illustrative data mining scenario is presented in Section 3. Finally, the conclusions of this chapter are presented in Section 4, stressing the main contributions and pointing to related future work.

2 The Mining Views Framework

In this section, we present the mining views framework in detail. This framework consists of a set of relational tables, called mining views, which virtually represent the complete output of data mining tasks. In reality, the mining views are empty and the database system finds the required tuples only when they are queried by the user.

2.1 The Mining View Concepts

We assume to be working in a relational database which contains the table $T(A_1, \ldots, A_n)$, having only categorical attributes. We denote the domain of A_i by $dom(A_i)$, for all $i = 1 \ldots n$. A tuple of T is therefore an element of $dom(A_i) \times \ldots \times dom(A_n)$. The active domain of A_i of T, denoted by $adom(A_i, T)$, is defined as the set of values that are currently assigned to A_i, that is, $adom(A_i, T) := \{t.A_i \mid t \in T\}$.

[1] `http://www.postgresl.org/`

In the mining views framework, the patterns extracted from table T are generically represented by what we call *concepts*. We denote a concept as a conjunction of attribute-value pairs that is definable over table T. For example,

$$(Outlook = \text{'Sunny'} \wedge Humidity = \text{'High'} \wedge Play = \text{'No'})$$

is a concept defined over the classical relational data table Playtennis [24], a sample of which is illustrated in Figure 1.

Playtennis

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
...	...	...	...	...	...

Fig. 1 The data table Playtennis.

To represent each concept as a database tuple, we use the symbol '?' as the *wildcard value* and assume it does not exist in the active domain of any attribute of T.

Definition 1. A concept over table T is a tuple $(c_1, \ldots, c_n)$ with $c_i \in adom(A_i) \cup \{\text{'?'}\}$, for all i=1 ... n.

Following Definition 1, the example concept above is represented by the tuple

('?', 'Sunny', '?', '?', 'High', '?', 'No').

We are now ready to introduce the mining view $T\_Concepts$. In the proposed framework, the mining view $T\_Concepts(cid, A_1, \ldots, A_n)$ virtually contains all concepts that are definable over table T. We assume that these concepts can be sorted in lexicographic order and that an identifier can unambiguously be given to each concept.

Definition 2. The mining view $T\_Concepts(cid, A_1, \ldots, A_n)$ contains one tuple $(cid, c_1, \ldots, c_n)$ for every concept defined over table T. The attribute *cid* uniquely identifies the concepts.

In fact, the mining view $T\_Concepts$ represents exactly a *data cube* [25] built from table T, with the difference that the wildcard value "ALL" introduced in [25] is replaced by the value '?'. By following the syntax introduced in [25], the mining view $T\_Concepts$ would be created with the SQL query shown in Figure 2 (consider adding the identifier *cid* after its creation).

```
1. create table T_Concepts
2. select A1, A2,..., An
3. from T
4. group by cube A1, A2,..., An
```

Fig. 2 The data cube that represents the contents of the mining view *T_Concepts*.

2.2 Representing Patterns and Models as Sets of Concepts

We now explain how patterns extracted from the table Playtennis can be represented by the concepts in the mining view *Playtennis_Concepts*. In the remainder of this section, we refer to table Playtennis as T and use the concepts in Figure 3 for the illustrative examples.

Playtennis_Concepts

cid	Day	Outlook	Temperature	Humidity	Wind	Play
...	...	...	...	...	...	...
101	?	?	?	?	?	Yes
102	?	?	?	?	?	No
103	?	Sunny	?	High	?	?
104	?	Sunny	?	High	?	No
105	?	Sunny	?	Normal	?	Yes
106	?	Overcast	?	?	?	Yes
107	?	Rain	?	?	Strong	No
108	?	Rain	?	?	Weak	Yes
109	?	Rain	?	High	?	No
110	?	Rain	?	Normal	?	Yes
...	...	...	...	...	...	...

Fig. 3 A sample of the mining view *Playtennis_Concepts*, which is used for the illustrative examples in Section 2.2.

2.2.1 Itemsets and Association Rules

As itemsets in a relational database are conjunctions of attribute-value pairs, they can be represented as concepts. Itemsets are represented in the proposed framework by the mining view:

$$T\_Sets(cid, supp, sz).$$

The view *T_Sets* contains a tuple for each itemset, where *cid* is the identifier of the itemset (concept), *supp* is its support (the number of tuples

satisfied by the concept), and *sz* is its size (the number of attribute-value pairs in which there are no wildcards).

Similarly, association rules are represented by the view:

$$T\_Rules(rid, cida, cidc, cid, conf).$$

T_Rules contains a tuple for each association rule that can be extracted from table T. We assume that a unique identifier, *rid*, can be given to each rule. The attribute *rid* is the rule identifier, *cida* is the identifier of the concept representing its left hand side (referred to here as antecedent), *cidc* is the identifier of the concept representing its right hand side (referred to here as consequent), *cid* is the identifier of the union of the last two, and *conf* is the confidence of the rule.

Figure 4 shows the mining views *T_Sets* and *T_Rules*, and illustrates how the rule "if outlook is sunny and humidity is high, you should not play tennis" is represented in these views by using three of the concepts given in Figure 3.

T_Sets

cid	supp	sz
102	5	1
103	3	2
104	3	3
...	...	...

T_Rules

rid	cida	cidc	cid	conf
1	103	102	104	100%
...	...	...	...	...

Fig. 4 Mining views for representing itemsets and association rules. The attributes *cida*, *cidc*, and *cid* refer to concepts given in Figure 3.

In Figure 5, queries (A) and (B) are example mining queries over itemsets and association rules, respectively. Query (A) asks for itemsets having support of at least 3 and size of at most 5, while query (B) asks for association rules having support of at least 3 and confidence of at least 80%. Note that these two common data mining tasks and the well known constraints "minimum support" and "minimum confidence" can be expressed quite naturally with SQL queries over the mining views.

2.2.2 Decision Trees

A decision tree learner typically learns a single decision tree from a dataset. This setting strongly contrasts with discovery of itemsets and association rules, which is set-oriented: given certain constraints, the system finds all itemsets or association rules that fit the constraints. In decision tree learning, given a set of (sometimes implicit) constraints, one tries to find one tree that fulfills the constraints and, besides that, optimizes some other criteria, which are again not specified explicitly but are a consequence of the algorithm used.

(A)

```
select C.*, S.supp, S.sz
from T_Concepts C, T_Sets S
where C.cid = S.cid
  and S.supp >= 3
  and S.sz <= 5
  and C.Outlook = 'Sunny'
```

(B)

```
select Ante.*, Cons.*,
       S.supp, R.conf
from T_Sets S, T_Rules R,
     T_Concepts Ante,
     T_Concepts Cons
where R.cid = S.cid
  and Ante.cid = R.cida
  and Cons.cid = R.cidc
  and S.supp >= 3
  and R.conf >= 80
```

Fig. 5 Example queries over itemsets and association rules.

In the inductive databases context, we treat decision tree learning in a somewhat different way, which is more in line with the set-oriented approach. Here, a user would typically write a query asking for all trees that fulfill a certain set of constraints, or optimizes a particular condition. For example, the user might ask for the tree with the highest training set accuracy among all trees of size of at most 5. This leads to a much more declarative way of mining for decision trees, which can easily be integrated into the mining views framework. The set of all trees predicting a particular target attribute A_i from other attributes is represented by the view:

$$T\_Trees\_A_i(treeid, cid).$$

The mining view $T\_Trees\_A_i$ is such that, for every decision tree predicting a particular target attribute A_i, it contains as many tuples as the number of leaf nodes it has. We assume that a unique identifier, *treeid*, can be given to each decision tree. Each decision tree is represented by a set of concepts *cid*, where each concept represents one path from the root to a leaf node.

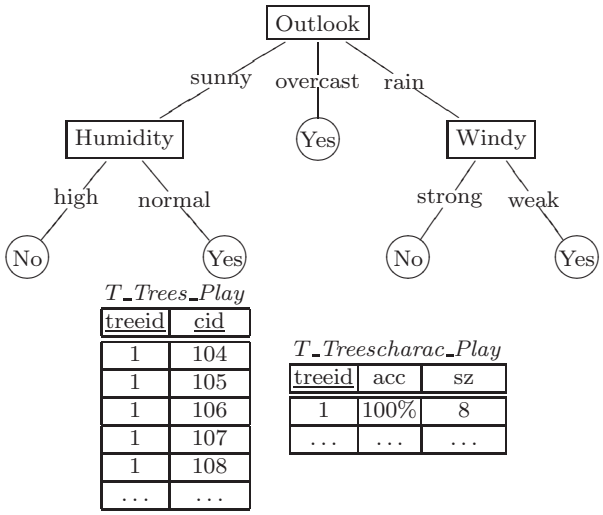
Additionally, a view representing several characteristics of a tree learned for one specific target attribute A_i is defined as:

$$T\_Treescharac\_A_i(treeid, acc, sz).$$

It contains a tuple for every decision tree in $T\_Trees\_A_i$, where *treeid* is the decision tree identifier, *acc* is its corresponding accuracy, and *sz* is its size in number of nodes.

Figure 6 shows how a decision tree that predict the attribute *Play* of table T is represented in the mining views T_Trees_Play and $T\_Treescharac\_Play$ by using the concepts in Figure 3.

In Figure 7, we present some example mining queries over decision trees. Query (C) creates a table called "BestTrees" with all decision trees that predict the attribute *Play*, having maximal accuracy among all possible decision trees of size of at most 5. Query (D) asks for decision trees having a test on



T_Trees_Play

treeid	cid
1	104
1	105
1	106
1	107
1	108
...	...

T_Treescharac_Play

treeid	acc	sz
1	100%	8
...	...	...

Fig. 6 Mining views representing a decision tree which predicts the attribute *Play*. Each attribute *cid* of view *T_Trees_Play* refers to a concept given in Figure 3.

"Outlook=Sunny" and on "Wind=Weak", with a size of at most 5 and an accuracy of at least 80%.

(C)

```
create table BestTrees as
select T.treeid, C.*, D.*
from T_Concepts C,
     T_Trees_Play T,
     T_Treescharac_Play D
where T.cid = C.cid
  and T.treeid = D.treeid
  and D.sz <= 5
  and D.acc =
    (select max(acc)
     from T_Treescharac_Play
     where sz <= 5)
```

(D)

```
select T1.treeid,
       C1.*, C2.*
from T_Trees_Play T1,
     T_Trees_Play T2,
     T_Concepts C1,
     T_Concepts C2,
     T_Treescharac_Play D
where C1.Outlook = 'Sunny'
  and C2.Wind = 'Weak'
  and T1.cid = C1.cid
  and T2.cid = C2.cid
  and T1.treeid = T2.treeid
  and T1.treeid = D.treeid
  and D.sz <= 5
  and D.acc >= 80
```

Fig. 7 Example queries over decision trees.

Prediction

In order to classify a new tuple using a learned decision tree, one simply searches for the concept in this tree (path) that is satisfied by the new tuple. More generally, if we have a test set S, all predictions of the tuples in S are obtained by equi-joining S with the semantic representation of the decision tree given by its concepts. We join S to the concepts of the tree by using a variant of the equi-join that requires that either the values are equal, or there is a wildcard value.

Consider the table BestTrees created after the execution of query (C), in Figure 7. Figure 8 shows a query that predicts the attribute *Play* for all unclassified tuples in an example table Test_Set(*Day*,*Outlook*,*Temperature*, *Humidity*,*Wind*) by using the tree in table BestTrees that has identification number 1.

(E)

```
select S.*, T.Play
from Test_Set S,
     BestTrees T
where (S.Day = T.Day or T.Day = '?')
  and (S.Outlook = T.Outlook or T.Outlook  = '?')
  and (S.Temperature = T.Temperature or T.Temperature = '?')
  and (S.Humidity = T.Humidity or T.Humidity = '?')
  and (S.Wind = T.Wind or T.Wind = '?')
  and T.treeid = 1
```

Fig. 8 An example prediction query.

2.3 Putting It All Together

For every data table $T(A_1, \ldots, A_n)$ in the database, with T having only categorical attributes, the virtual mining views framework consists of a set of relational tables, called virtual mining views, which virtually contain the complete output of data mining tasks executed over T. These mining views are the following:

- $T\_Concepts(cid,A_1,\ldots,A_n)$.
- $T\_Sets(cid,supp,sz)$.
- $T\_Rules(rid,cida,cidc,cid,conf)$.
- $T\_Trees\_A_i(treeid,cid)$, for all i=1 . . . n.
- $T\_Treescharac\_A_i(treeid,\ acc,\ sz)$, for all i=1 . . . n.

As shown in the examples given in this section, in order to retrieve patterns over table T, the user simply needs to write SQL queries over the proposed mining views. The semantics of these queries is the same as that of queries over traditional relational tables. For more example queries over the mining views, we refer the reader to [7].

Another important thing to note is that if the user wants to mine itemsets, association rules, or learn a decision tree from only a portion of table T, he or she should first create a new table T' from T, applying the appropriate selections and (or) projections. Then, the mining views associated with T', which are automatically created, will represent the patterns extracted from that corresponding portion of the data.

2.4 Mining Views vs. Data Mining Tasks

We now present how the 4 prototypical tasks described in the previous chapter can be executed by SQL queries over the mining views.

2.4.1 Discretization task: Discretize attribute Temperature into 3 intervals. The discretized attribute should be used in the subsequent tasks

Since the data mining query language is SQL, our approach does not offer any new operator for pre-processing tasks. The discretization task can thus be performed by creating a new table called "MyPlaytennis" with the SQL CASE query introduced in the previous chapter (when presenting the MINE RULE operator).

2.4.2 Area task: Find all intra-tuple itemsets with relative support of at least 20%, size of at least 2, and area, that is, absolut support $\times$ size, of at least 10

The area task can be performed with an SQL query involving the mining views *MyPlaytennis_Concepts* and *MyPlaytennis_Sets*, which are created automatically after the creation of table MyPlaytennis for the discretization task. The query is shown below. Notice that the property area can be constrained quite naturally in our framework (see line 6), due to the flexibility of ad hoc querying.

```
1. select C.*, S.supp, S.sz,
          S.supp * S.sz as area
2. from MyPlaytennis_Sets S,
        MyPlaytennis_Concepts C
3. where C.cid = S.cid
4.  and S.supp >= 3
5.  and S.sz >= 2
6.  and S.supp * S.sz >= 10
```

2.4.3 Right hand side task: Find all intra-tuple association rules with relative support of at least 20%, confidence of at most 80%, size of at most 3, and a singleton right hand size

Since the next task (lift task) requires a post-processing query over the results output by this one, it is necessary to store these results so that they can be further queried. The SQL query to perform the right hand side task is the following:

```
1. create table MyRules as
2. select Ant.Day as DayA, ... ,Ant.Play as PlayA,
          Con.Day as DayC, ..., Con.Play as PlayC,
          R.conf, SCon.supp/14 as suppC
3. from MyPlaytennis_Sets S, MyPlaytennis_Rules R,
        MyPlaytennis_Concepts Ant,
        MyPlaytennis_Concepts Con,
        MyPlaytennis_Sets SCon
4. where R.cid = S.cid
5.   and Ant.cid = R.cida
6.   and Con.cid = R.cidc
7.   and S.supp >= 3
8.   and R.conf >= 80
9.   and S.sz <= 3
10.  and SCon.cid = R.cidc
11.  and SCon.sz = 1
```

The query above creates a new table called “MyRules”. We also store in this table the confidence of the rules along with the relative supports of their consequents, since they are necessary to perform the lift task (the number 14, which is used to compute the relative supports of the consequents, refers to the total number of tuples in table MyPlaytennis). Observe that the mining views framework does not restrain the user from any format in which the rules are to be stored, thanks again to the flexibility of ad hoc querying.

2.4.4 Lift task: Find, from the result of the right hand side task, rules with attribute Play as consequent that have a lift greater than 1

In order to perform the lift task, one needs to query table MyRules, created for the previous task. The query in question is the one depicted below:

```
1. select M.*, (M.conf/100)/M.suppC as lift
2. from MyRules M
3. where M.PlayC <> '?'
4.   and (M.conf/100)/M.suppC >=1
```

Note that the two constraints required by the lift task can be expressed quite naturally in our framework. In line 3, we assure that the rules in the result have the attribute *Play* as consequent, i.e., it is not a wildcard value. In line 4, we compute the property lift of the rules.

2.5 Conclusions

Observe that the mining views framework is able to perform all data mining tasks described in the previous chapter without any type of pre- or post-processing, as opposed to the other proposals. Also note that the choice of the schema for representing itemsets and association rules implicitly determines the complexity of the queries a user needs to write. For instance, by adding the attributes *sz* and *supp* to the mining views *T_Sets*, the area constraint can be expressed quite naturally in our framework. Without these attributes, one could still obtain their values. Nevertheless, it would imply that the user would have to write more complicated queries.

The addition of the attribute *cid* in the mining view *T_Rules* can be justified by the same argument. Indeed, one of the 3 concept identifiers for an association rule, *cid*, *cida* or *cidc* is redundant, as it can be determined from the other two. However, this redundancy eases query writing. Still with regard to the mining view *T_Rules*, while the query for association rule mining seems to be more complex than the queries for the same purpose in other data mining query languages (e.g., in MSQL), one could easily turn it into a view definition so that association rules can be mined with simple queries over that database view.

It is also important to notice that some types of tasks are not easily expressed with the mining views. For example, if the tuples over which the data mining tasks are to be executed come from different tables in the database, a new table containing these tuples should be created before the mining can start. In DMQL, MINE RULE, SPQL the relevant set of tuples can be specified in the query itself. In the case of DMX, this can be done while training

the model. Another example is the extraction of inter-tuple patterns, which are possible to be performed with DMQL, MINE RULE, SPQL, and DMX. To mine inter-tuple patterns in the mining views framework, one would need to first pre-process the dataset that is to be mined, by changing its representation: the relevant attributes of a group of tuples should be added to a single tuple of a new table. Constraints on the corresponding groups of tuples being considered, which are allowed to be specified in the proposals mentioned above, can be specified in a post-processing step over the results. Our proposal is more related to MSQL and SIQL, as they also only allow the extraction of intra-tuple patterns over a single relation.

Some data mining tasks that can be performed in SIQL and DMX, such as clustering, cannot currently be executed with the proposed mining views. On the other hand, note that one could always extend the framework by defining new mining views that represent clusterings, as studied in [7]. In fact, one difference between our approach and those presented in the previous chapter is the fact that to extend the formalism, it is necessary to define new mining views or simply add new attributes to the existing ones, whereas in other formalisms one would need to extend the language itself.

To finalize, although the mining views do not give the user the ability to express every type of query the user can think of (similarly to any relational database), the set of mining tasks that can be executed by the system is consistent and large enough to cover several steps in a knowledge discovery process.

We now list how the mining views overcome the drawbacks found in at least one of the proposals surveyed in the previous chapter:

2.5.1 Satisfaction of the closure principle

Since, in the proposed framework, the data mining query language is standard SQL, the closure principle is clearly satisfied.

2.5.2 Flexibility to specify different kinds of patterns

The mining views framework provides a very clear separation between the patterns it currently represents, which in turn can be queried in a very declarative way (SQL queries). In addition to itemsets, association rules and decision trees, the flexibility of ad hoc querying allows the user to think of new types of patterns which may be derived from those currently available. For example, in [7] we show how frequent closed itemsets [26] can be extracted from a given table T with an SQL query over the available mining views $T\_Concepts$ and T_Sets.

2.5.3 Flexibility to specify ad hoc constraints

The mining views framework is meant to offer exactly this flexibility: by virtue of a full-fledged query language that allows of ad hoc querying, the user can think of new constraints that were not considered at the time of implementation. An example is the constraint lift, which could be computed by the framework for the execution of the lift task.

2.5.4 Intuitive way of representing mining results

In the mining views framework, patterns are all represented as sets of concepts, which makes the framework as generic as possible, not to mention that the patterns are easily interpretable.

2.5.5 Support for post-processing of mining results

Again, thanks to the flexibility of ad hoc querying, post-processing of mining results is clearly feasible in the mining views framework.

3 An Illustrative Scenario

One of the main advantages of our system is the flexibility of ad hoc querying, that is, the user can iteratively specify new types of constraints and query the patterns in combination with the data themselves. In this section, we illustrate this feature with a complete data mining scenario that consists in extracting knowledge from real gene expression data, after an extensive preprocessing phase. Differently to the scenario presented in [6], here we do not learn a classifier, but mine for non-redundant correct association rules.

We begin by presenting how the implementation of our inductive database system was realized. Next, the aforementioned scenario is presented.

3.1 Implementation

Our inductive database system was developed into the well-known open source database system PostgreSQL[2], which is written in C language. Every time a data table is created into our system, its mining views are automati-

[2] `http://www.postgresl.org/`

cally created. Accordingly, if this data table is removed from the system, its mining views are deleted as well.

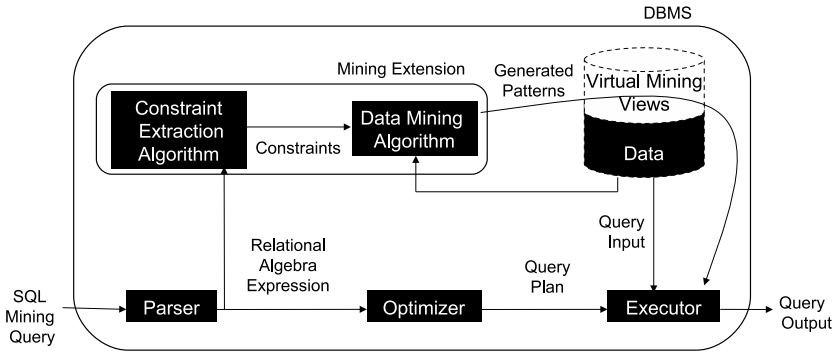


Fig. 9 The proposed inductive database system implemented into PostgreSQL.

The main steps of the system are illustrated in Figure 9. When the user writes a query, PostgreSQL generates a data structure representing its corresponding relational algebra expression. A call to our Mining Extension was added to PostgreSQL's source code after the generation of this data structure. In the Mining Extension, which was implemented in C language, we process the relational algebra structure. If it refers to one or more mining views, we then extract the constraints (as described in detail in [7]), trigger the data mining algorithms and materialize the virtual mining views with the obtained mining results. Just after the materialization (i.e., upon return from the miningExtension() call), the work-flow of the database system continues and the query is executed as if the patterns or models were there all the time. We refer the reader to [5, 7] for more details on the implementation and efficiency evaluation of the system.

Additionally, we adapted the web-based administration tool PhpPgAdmin[3] so as to have a user-friendly interface to the system.

3.2 Scenario

The scenario presented in this section consists in extracting knowledge from the gene expression data which resulted from a biological experimentation concerning the transcription of *Plasmodium Falciparum* [27] during its reproduction cycle (IDC) within the human blood cells.

The Plasmodium Falciparum is a parasite that causes human malaria. The data gather the expression profiles of 472 genes of this parasite in 46 different

[3] `http://phppgadmin.sourceforge.net/`

biological samples.[4] Each gene is known to belong to a specific biological function. Each sample in turn corresponds to a time point (hour) of the IDC, which lasts for 48 hours. During this period, the merozoite (initial stage of the parasite) evolves to 3 different identified stages: Ring, Trophozoite, and Schizont. In addition, due to reproduction, one merozoite leads to up to 32 new ones during each cycle, after which a new developmental cycle is started. Figure 10 shows the percentage of parasites (y-axis) that are at the Ring (black curve), Trophozoite (light gray curve), or Schizont (dark gray curve) stage, at every time point of the IDC (x-axis).

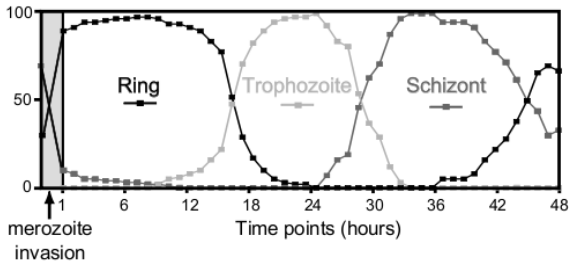


Fig. 10 Major developmental stages of Plasmodium Falciparum parasite (Figure from [27]). The three curves, in different levels of gray, represent the percentage of parasites (y-axis) that are at the Ring (black), Trophozoite (light gray), or Schizont stage (dark gray), at every time point of the IDC (x-axis).

These data were stored into 3 different tables in our system, as illustrated in Figure 11. They are the following:

- GeneFunctions(function_id, function): represents the biological functions. There are in total 12 different functional groups.
- Samples(sample_name, stage): represents the samples themselves. Two data points are missing, namely the 23rd and 29th hours. We added to this table the attribute called *stage*, the values of which are based on the curves illustrated in Figure 10: this new attribute discriminates the samples having at least 75% of the parasites in the Ring (stage=1), Trophozoite (stage=2) or Schizont (stage=3) stage. Samples that contain less than 75% of any parasite stage were assigned to stage 4, a "non-identified" stage. Thus, for our scenario, stage 1 corresponds to time points between 1 and 16 hours; stage 2 corresponds to time points between 18 and 28 hours; and stage 3 gathers time points between 32 and 43 hours.
- Plasmodium(gene_id, function_id, tp_1, tp_2,..., tp_48): represents, for each of the genes, its corresponding function and its expression profile. As proposed in [27], we take the logarithm to the base 2 of the raw expression values.

[4] The data is available at http://malaria.ucsf.edu/SupplementalData.php

Plasmodium

gene_id	function_id	tp_1	tp_2	...	tp_48
1	12	-0.13	0.12	...	0.11
2	12	0.24	0.48	...	-0.03
...	...	...	...	...	...
472	5	1.2	0.86	...	1.15

GeneFunctions

function_id	function
1	Actin_myosin_mobility
2	Cytoplasmic_translation_machinery
...	...
12	Transcription_machinery

Samples

sample_name	stage
tp_1	1
tp_2	1
...	...
tp_48	4

Fig. 11 The Plasmodium data.

Having presented the data, we are now ready to describe the goal of our scenario. In gene expression analysis, a gene is said to be highly expressed, according to a biological sample, if there are many RNA transcripts in the considered sample. These RNA transcripts can be translated into proteins, which can, in turn, influence the expression of other genes. In other words, it can make other genes also highly expressed. This process is called *gene regulation* [27].

In this context, analogously to what the biologists have studied in [27], we want to characterize the parasite's different stages by identifying the genes that are active during each stage. More precisely, we want to identify, for each different stage, the functional groups whose genes have an unusual high level of expression or, as the biologists say, are overexpressed in the corresponding set of samples. By considering the samples corresponding to a specific stage and the genes that are overexpressed within those samples, we might have insights into the regulation processes that occur during the development of the parasite. As pointed out in [27], understanding these regulation processes would provide the foundation for future drug and vaccine development efforts toward eradication of the malaria.

Observe that decision trees are not appropriate for the analysis we want to perform; they are most suited for predicting, which is not our intention here. Therefore, in our scenario we mine for association rules. A couple of pre-processing steps have to be performed initially, such as the discretization of the expression values. These steps are described in detail in the first 3 subsequent subsections. The remaining subsections show how the desired rules can be extracted from the data.

3.2.1 Step 1: Pre-processing 1

Since our intention is to characterize the parasite's stages by means of the functional groups and not of the individual genes themselves, we first create

a view on the data that groups the genes by the function they belong to. The corresponding pre-process query is shown below:[5]

```
1. create view PlasmodiumAvg as
2. select G.function,
3.         avg(p.tp_1) as tp_1,
4.         ...,
5.         avg(p.tp_48) as tp_48
6. from Plasmodium P, GeneFunctions G
7. where P.function_id = G.function_id
8. group by G.function
```

The view called "PlasmodiumAvg" calculates, for every different functional group, the average expression profile (arithmetic mean) over all time points (see lines from 2 to 5).

3.2.2 Step 2: Pre-processing 2

Since we want the functional groups as components of the desired rules (antecedents and/or consequents), it is therefore necessary to transpose the view PlasmodiumAvg, which was created in the previous step. In other words, we need a new view in which the gene functional groups are the columns and the expression profiles are the rows. To this end, we use the PostgreSQL function called *crosstab*[6]. As *crosstab* requires the data to be listed down the page (not across the page), we first create a view on PlasmodiumAvg, called "PlasmodiumAvgTemp", which lists data in such format. The corresponding queries are shown below.

```
1. create view PlasmodiumAvgTemp as
2. select function as tid, 'tp_1' as item,
                      tp_1 as val
3. from PlasmodiumAvg
4. union
5. ...
6. union
7. select function as tid, 'tp_48' as item,
                      tp_48 as val
8. from PlasmodiumAvg
```

[5] For the sake of readability, ellipsis were added to some of the SQL queries presented in this section and in the following ones, which represent sequences of attribute names, attribute values, clauses etc.

[6] We refer the reader to `http://www.postgresql.org/docs/current/static/tablefunc.html` for more details on the *crosstab* function.

```
 9. create view PlasmodiumTranspose as
10. select * from crosstab
11. ('select item, tid, val from PlasmodiumAvgTemp
          order by item',
     'select distinct tid from PlasmodiumAvgTemp
          order by item')
12. as (sample_name text, Actin_myosin_mobility real,
                        ...,
                        Transcription_machinery real)
```

3.2.3 Step 3: Pre-processing 3

Having created the transposed view PlasmodiumTranspose, the third and last pre-processing step is to discretize the gene expression values so as to encode the expression property of each functional group of genes.

In gene expression data analysis, a gene is considered to be overexpressed if its expression value is high with respect to its expression profile. One approach to identify the level of expression of a gene is the method called *x% cut-off*, which was proven to be successful in [28]: a gene is considered overexpressed if its expression value is among the x% highest values of its expression profile, and underexpressed otherwise. With x=50, a gene is tagged as overexpressed if its expression value is above the median value of its profile.

As in this scenario the data are log transformed (very high expression values are deemphasized), the distribution of the data is symmetrical and, therefore, median expression values are very similar to mean values. As computing the mean value is straightforward in SQL and as we are not dealing with genes independently, but with groups of genes, we use a slight adaptation of the *50% cut-off* method: we encode the overexpression property by comparing it to the mean value observed for each group, rather than the median. We first create a view, called "PlasmodiumTransposeAvg", which calculates, for every group of genes, its mean expression value. This computation is performed by the following query:

```
1. create view PlasmodiumTransposeAvg as
2. select avg(Actin_myosin_mobility) as avg_Actin_mm,
3. ...
4. avg(Transcription_machinery) as avg_Tran_m
5. from PlasmodiumTranspose
```

Afterwards, we create the new table named "PlasmodiumSamples" applying the aforementioned discretization rule. The query that performs this discretization step is shown below. Notice that the attribute *stage* is also added to the new table PlasmodiumSamples (see line 2).

```
 1. create table PlasmodiumSamples as
 2. select P.sample_name, S.stage,
 3. case when P.Actin_myosin_mobility > avg_Actin_mm
 4.    then 'overexpressed'
 5.  else
 6.    'underexpressed'
 7.  end as Actin_myosin_mobility,
 8.  ...
 9.  case when P.Transcription_machinery > avg_Tran_m
10.     then 'overexpressed'
11.  else
12.     'underexpressed'
13.  end as Transcription_machinery
14.  from PlasmodiumTranspose P,
          PlasmodiumTransposeAvg,
          Samples S
15.  where P.sample_name = S.sample_name
16.  order by S.stage
```

3.2.4 Step 2: Mining over Association Rules

After creating the table PlasmodiumSamples, in this new step, we search for the desired rules. The corresponding query is shown below:

```
 1. create table RulesStage as
 2. select R.rid, S.sz, S.supp, R.conf,
          CAnt.stage as stage_antecedent
          CCon.*
 3. from PlasmodiumSamples_Sets S,
 PlasmodiumSamples_Sets SAnt,
         PlasmodiumSamples_Concepts CAnt,
         PlasmodiumSamples_Concepts CCon,
         PlasmodiumSamples_Rules R
 4. where R.cid = S.cid
 5.   and CAnt.cid = R.cida
 6.   and CCon.cid = R.cidc
 7.   and S.supp >= 10
 8.   and R.conf = 100
 9.   and R.cida =  SAnt.cid
10.   and SAnt.sz = 1
11.   and CAnt.stage <> '?'
12.  order by Ant.stage
```

As we want to characterize the parasite's stages themselves by means of the gene functions, we look for rules having only the attribute *stage* as the antecedent (see lines 9, 10 and 11) and gene function(s) in the consequent. Additionally, since we want to characterize the stages without any uncertainty, we only look for correct association rules, that is, rules with a confidence of 100% (see line 8). Finally, as the shortest stage is composed of 10 time points in total (not considering the dummy stage), we set 10 as the minimum support (line 7). The 381 resultant rules are eventually stored in the table called "RulesStage" (see line 1).

3.2.5 Step 3: Post-processing

The previous query has generated many redundant rules [29]: for each different antecedent, all rules have the same support and 100% confidence. Notice, however, that as we are looking for all gene groups that are overexpressed according to a given stage, it suffices to analyze, for each different stage (antecedent of the rules), only the rule that has maximal consequent. Given this, all one has to do is to select, for each different stage, the longest rule. The corresponding query is presented below. The sub-query, in lines from 3 to 5, computes, for every antecedent (stage), the maximal consequent size.

```
1. select R.*
2. from RulesStage R,
3. (select max(sz) as max_sz,
          stage_antecedent
4. from RulesStage
5. group by stage_antecedent) R1
6. where R.sz = R1.max_sz
7.   and R.stage_antecedent = R1.stage_antecedent
```

The 3 rules output by the last query are presented in Figure 12. As shown in Figure 13, they are consistent with the conclusion drawn in the corresponding biological article [27]. Each graph in Figure 13, from B to M, corresponds to the average expression profile of the genes of a specific functional group (the names of the functions are shown at the bottom of the figure). The functions are ordered, from left to right, with respect to the time point when there is a peak in their expression profiles (the peak value is shown in parentheses) and they are assigned to the parasite's stage during which this peak occurs (the name of the stages are presented at the top of the figure). Observe that, according to Figure 13, the functions *Early ring transcripts* and *Transcription machinery* are related to the early Ring and Ring stages, which is in fact indicated by the first extracted rule. The *Glycolytic pathway*, *Ribonucleotide synthesis*, *Deoxynucleotide synthesis*, *DNA replication*, and *Proteasome* are related to the early Trophozoite and the Trophozoite stages, which is also

antecedent (stage)	consequent	
	overexpressed	underexpressed
Ring	Early ring transcripts Transcription machinery	Deoxynucleotide synthesis DNA replication machine Plastid genome TCA cycle
Trophozoite	Glycolytic pathway Ribonucleotide synthesis Deoxynucleotide synthesis DNA replication Proteasome	Actin myosin motors Early ring transcripts Merozoite invasion
Schizont	Plastid genome Merozoite invasion Actin myosin mobility	Cytoplasmic translation machinery Ribonucleotide synthesis Transcription machinery

Fig. 12 Correct association rules with maximum consequent.

consistent with the second extracted rule. Finally, *Plastid genome*, *Merozoite Invasion*, and *Actin myosin mobility* have been associated to the Schizont stage by the biologists, which is indeed consistent with the third extracted rule.

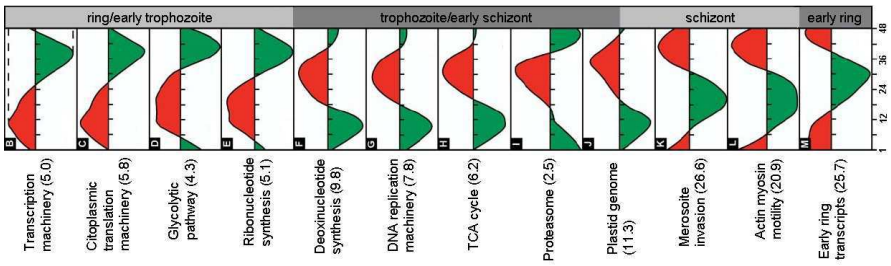


Fig. 13 The temporal ordering of functional groups of genes (an adapted figure from [27]). Each graph, from B to M, corresponds to the average expression profile of the genes of a specific functional group. The biologists of [27] have assigned each functional group to the parasite's stage in which it achieves its highest expression value.

4 Conclusions and Future Work

In this chapter, we described an inductive database system in which the query language is SQL. More specifically, we presented a system in which the user can query the collection of all possible patterns as if they were stored in traditional relational tables. The development of the proposed system was

motivated by the need to (a) provide an intuitive framework that covers different kinds of patterns in a generic way and, at the same time, allows of (b) ad hoc querying, (c) definition of meaningful operations and (d) querying of mining results.

As for future work, we identify the following three directions:

- Currently, the mining views are in fact empty and only materialized upon request. Therefore, inspired by the work of Harinarayan et al. [30], the first direction for further research is to investigate which mining views (or which parts of them) could actually be materialized in advance. This would speed up query evaluation.
- Our system deals with intra-tuple patterns only. To mine inter-tuple patterns, one would need to first pre-process the dataset that is to be mined, by changing its representation. Although this is not a fundamental problem, this pre-processing step may be laborious. For example, in the context of market basket analysis, a table would need to be created in which each transaction is represented as a tuple with as many boolean attributes as are the possible items that can be bought by a customer. An interesting direction for future work would then be to investigate how inter-tuple patterns can be integrated into the system.
- Finally, the prototype developed so far covers only itemset mining, association rules and decision trees. An obvious direction for further work is to extend it with other models, taking into account the exhaustiveness nature of the queries the users are allowed to write.

Acknowledgements This work has been partially supported by the projects IQ (IST-FET FP6-516169) 2005/8, GOA 2003/8 "Inductive Knowledge bases", FWO "Foundations for inductive databases", and BINGO2 (ANR-07-MDCO 014-02). When this research was performed, Hendrik Blockeel was a post-doctoral fellow of the Research Foundation - Flanders (FWO-Vlaanderen), Élisa Fromont was working at the Katholieke Universteit Leuven, and Adriana Prado was working at the University of Antwerp.

References

1. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM **39** (1996) 58–64
2. Calders, T., Goethals, B., Prado, A.: Integrating pattern mining in relational databases. In: Proc. ECML-PKDD. (2006) 454–461
3. Fromont, E., Blockeel, H., Struyf, J.: Integrating decision tree learning into inductive databases. In: ECML-PKDD Workshop KDID (Revised selected papers). (2007) 81–96
4. Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A.: Mining views: Database views for data mining. In: ECML-PKDD Workshop CMILE. (2007)
5. Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A.: Mining views: Database views for data mining. In: Proc. IEEE ICDE. (2008)

6. Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A.: An inductive database prototype based on virtual mining views. In: Proc. ACM SIGKDD. (2008)
7. Prado, A.: An Inductive Database System Based on Virtual Mining Views. PhD thesis, University of Antwerp, Belgium (December 2009)
8. Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: DMQL: A data mining query language for relational databases. In: ACM SIGMOD Workshop DMKD. (1996)
9. Imielinski, T., Virmani, A.: Msql: A query language for database mining. Data Mining Knowledge Discovery **3**(4) (1999) 373–408
10. Meo, R., Psaila, G., Ceri, S.: An extension to sql for mining association rules. Data Mining and Knowledge Discovery **2**(2) (1998) 195–224
11. Wicker, J., Richter, L., Kessler, K., Kramer, S.: Sinbad and siql: An inductive databse and query language in the relational model. In: Proc. ECML-PKDD. (2008) 690–694
12. Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., Trasarti, R.: A constraint-based querying system for exploratory pattern discovery information systems. Information System (2008) Accepted for publication.
13. Tang, Z.H., MacLennan, J.: Data Mining with SQL Server 2005. John Wiley & Sons (2005)
14. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. VLDB. (1994) 487–499
15. Botta, M., Boulicaut, J.F., Masson, C., Meo, R.: Query languages supporting descriptive rule mining: A comparative study. In: Database Support for Data Mining Applications. (2004) 24–51
16. Han, J., Kamber, M.: Data Mining - Concepts and Techniques, 1st ed. Morgan Kaufmann (2000)
17. Han, J., Chiang, J.Y., Chee, S., Chen, J., Chen, Q., Cheng, S., Gong, W., Kamber, M., Koperski, K., Liu, G., Lu, Y., Stefanovic, N., Winstone, L., Xia, B.B., Zaiane, O.R., Zhang, S., Zhu, H.: Dbminer: a system for data mining in relational databases and data warehouses. In: Proc. CASCON. (1997) 8–12
18. Srikant, R., Agrawal, R.: Mining generalized association rules. Future Generation Computer Systems **13**(2–3) (1997) 161–180
19. Meo, R., Psaila, G., Ceri, S.: A tightly-coupled architecture for data mining. In: Proc. IEEE ICDE. (1998) 316–323
20. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1**(3) (1997) 241–258
21. Ng, R., Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: Proc. ACM SIGMOD. (1998) 13–24
22. Pei, J., Han, J., Lakshmanan, L.V.S.: Mining frequent itemsets with convertible constraints. In: Proc. IEEE ICDE. (2001) 433–442
23. Bistarelli, S., Bonchi, F.: Interestingness is not a dichotomy: Introducing softness in constrained pattern mining. In: Proc. PKDD. (2005) 22–33
24. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
25. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. Data Mining and Knowledge Discovery (1996) 152–159
26. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Proc. ICDT. (1999) 398–416
27. Bozdech, Z., Llinás, M., Pulliam, B.L., Wong, E.D., Zhu, J., DeRisi, J.L.: The transcriptome of the intraerythrocytic developmental cycle of plasmodium falciparum. PLoS Biology **1**(1) (2003) 1–16

28. Becquet, C., Blachon, S., Jeudy, B., Boulicaut, J.F., Gandrillon, O.: Strong association rule mining for large-scale gene-expression data analysis: a case study on human SAGE data. Genome Biology **12** (2002)
29. Zaki, M.J.: Generating non-redundant association rules. In: Proc. ACM SIGKDD. (2000) 34–43
30. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: Proc. ACM SIGMOD. (1996) 205–216