Multi-Relational Data Mining
(paper id: 294)

Arno J. Knobbe
Syllogic B.V.
Hoefseweg 1
3821 AE Amersfoort
The Netherlands
a.knobbe@syllogic.com

Hendrik Blockeel
Katholieke Universiteit Leuven
Department of CS
Celestijnenlaan 200A
3001 Heverlee Belgium
hendrik.blockeel@cs.kuleuven.ac.be

Arno Siebes
CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
arno@cwi.nl

Daniël M.G. van der Wallen
Syllogic B.V.
Hoefseweg 1
3821 AE Amersfoort
The Netherlands
d.van.der.wallen@syllogic.com

Abstract

An important aspect of data mining algorithms and systems is that they should scale well to large databases A consequence of this is that most data mining tools are based on machine learning algorithms that work on data in attribute-value format. Experience has proven that such 'single-table' mining algorithms indeed scale well. The downside of this format is, however, that more complex patterns are simply not expressible in this format and, thus, cannot be discovered.
One way to enlarge the expressiveness is to generalize, as in ILP, from one-table mining to multiple table mining, i.e., to support mining on full relational databases. The key step in such a generalization is to ensure that the search space does not explode and that efficiency and, thus, scalability are maintained. In this paper we present a framework and an architecture that provide such a generalization.
In this framework the semantic information in the database schema, e.g., foreign keys, are exploited to prune the search space and, in the architecture, database primitives are defined to ensure efficiency. Moreover, the framework induces a canonical generalization of algorithms, i.e., if the generalized algorithms are run on a single table database, they give the same results as their single-table counterparts. The framework is illustrated by the Warmr algorithm, which is a multi-relational generalization of the Apriori algorithm.

Introduction

An important aspect of data mining algorithms is that they should scale well to large databases. Paying a lot of attention to efficiency is especially necessary in the case of databases that may contain very complex patterns. In such databases the search space of patterns may be so large that scaling to a large database is impossible. This is precisely the reason why mining large relational databases containing more than one table has been given very little attention. In this paper we show that analysing multiple tables efficiently is very well feasible, and present a multi-relational data mining framework that exploits the extensive information in the data model for optimisation. This knowledge can be used to significantly prune the search space, and thus prevent the combinatorial explosion that would otherwise occur.

Most of the data mining algorithms which are currently available are based on an attribute-value setting which restricts their use to datasets consisting of a single table (or relation). The attribute-value paradigm only allows the analysis of fairly simple objects. It requires that each object can be described by a fixed set of attributes each of which can only have a single (unstructured) value. To be able to represent more complex and structured objects, one has to employ a relational database containing multiple tables. Each object can be described by multiple records in multiple tables. To be able to analyse relational databases containing multiple relations properly, specific algorithms will have to be written that cope with the structural information that occurs in relational databases. The multi-relational data mining framework described in this paper can be used to base many multi-relational data mining algorithms on.

The framework supports a range of multi-relational data mining algorithms, which are direct generalisations of common attribute-value induction algorithms. We will explain how many of the concepts used by attribute-value algorithms can be generalised to a multi-relational setting. The idea is for these multi-relational data mining algorithms to be full generalisations of their attribute-value counterparts, which means that if they were run on a database containing a single table, the result would be as if the single table version of the algorithm was run. An important extension of attribute-value learning that is achieved by multi-relational data mining can be found in the language that is used to describe patterns. In attribute-value learning, such a language is based on sets of conditions on the attributes of the table, which describe a particular selection of objects. In multi-relational data mining, one can not only have conditions on the values of an attribute, but also on the existence of related records in other tables. This way, one can also include statements about the structural information of objects in the selections.

The set of patterns derived from a relational database is potentially much bigger than the set of patterns which can be derived from a single table. Therefore, a lot of attention will have to be given to reducing the search space and to efficiently evaluating potentially interesting patterns. We will show how a lot of information which is stored in the data model can be used to prune the search space, and thus make the set of candidate patterns manageable. The candidate patterns that are valid according to the data model, and that will have to be evaluated, are sent to an efficient server which validates patterns against the data. This server supports a small number of primitives, which can be used to provide the necessary statistics about candidate patterns. These primitives are direct generalisations of those used in many data mining architectures based on the attribute-value paradigm [7, 8, 9, 11]. These primitives can be expressed in SQL and sent to a conventional RDBMS, or can be supported by a dedicated server which is optimised to compute the primitives efficiently. We will briefly sketch the architecture of such a server, and the primitives it supports. This architecture is a direct generalisation of an existing data mining architecture which is successfully being used in several commercial products [8, 9, 13].

When using a relational database containing multiple tables, it is not always strictly necessary to use a multi-relational data mining algorithm. There are ways of 'moulding' a relational database into a single table format, such that traditional attribute-value algorithms will be able to work with the database. One way of doing this, is to create a universal relation, which involves joining all tables to form a single table. The resulting universal relation can be extremely large and impractical to handle. The second way of transforming a relational database into a single table involves the creation of new attributes in the central fact table that summarise or aggregate information which can be found in other tables. This method is used in the LINUS system [14, 15] among others. However, it produces very wide tables with lots of data being repeated. Although more data is produced, a lot of information about how the data was originally structured is lost, and along with that the main source for efficiency in multi-relational data mining. Also, the creation of useful, understandable and informative new attributes may require a substantial amount of domain knowledge, which may defy the purpose of KDD. Therefore, neither approach is very attractive, especially from an efficiency point of view, and a proper way of dealing with multiple relations is necessary.

The idea of mining from multiple tables is not a new one. It is being studied extensively in the field of Inductive Logic Programming (ILP) [6, 15]. However, these approaches are mostly based on data stored as Prolog programmes, and little attention has been given to data stored in relational database and to how knowledge of the data model can help to guide the search process [1, 16, 21]. Nor has a lot of attention been given to efficiency and scalability issues. Our approach combines the achievements of the KDD field with some of those of the ILP field. We will demonstrate how existing ILP algorithms [2, 4], which have shown their practical applicability, can be implemented as special instances of our multi-relational data mining framework. Where ILP can be seen as learning from a set of predicates, multi-relational data mining can be seen as learning from a relational database. Extensional predicates, i.e. those predicates for which only ground facts exist, are the counterparts of tables in a relational database. Intentional predicates, i.e. those predicates for which also rules are given, correspond to the concept of views. However, multi-relational data mining differs from ILP in three aspects. Firstly, it is restricted to the discovery of non-recursive patterns. Secondly, the semantic information in the database is exploited explicitly. Thirdly, the emphasis on database primitives ensures efficiency.

The outline of this paper is as follows. In **Multi-Relational Data Mining** we describe the basic problem we intend to solve. A description is given of how to model data in a relational database, and how high-level knowledge about the database can be given. In the section **Framework** we describe the basic multi-relational data mining framework, and specifically how knowledge from the data model is being used in to prune the search space. An efficient architecture to support the multi-relational data mining framework is presented in **Architecture**. To demonstrate how our framework can be used to implement multi-relational data mining algorithms, we give an algorithm for finding association rules in multiple tables, in section **Instance**. We end with some conclusions and future work, in **Conclusion**.

Multi-relational data mining

We will assume that the data to be analysed is stored in a relational database [3, 20]. A relational database consists of a set of tables and a set of associations (i.e. constraints) between pairs of tables describing how records in one table relate to records in another table. Both tables and associations are also known as relations, so we will use the former terminology to be able to distinguish between the two concepts. An association between two tables describes the relationships between records in both tables. The nature of this relationship is characterised by the *multiplicity* of the association. The multiplicity of an association determines whether several records in one table relate to single or multiple records in the second table. Also, the multiplicity determines whether every record in one table needs to have at least one corresponding record in the second table. More formally, we define the following two predicates related to the multiplicity of an association *A* between two tables *P* and *Q*.

Definition 1 *Multiple*(*A*, *P*) iff every record in *Q* may correspond to multiple records in *P*.

Definition 2 *Zero*(*A*, *P*) iff a record in *Q* may have no corresponding record in *P*.

Note that these predicates are defined both for *P* and *Q*, and both predicates may or may not hold for either *P* or *Q*. This way there are 16 different possible multiplicities for association *A*.

A special case of an association between two tables is a *foreign key relation*. A foreign key relation from a foreign key in table *P* to a primary key in table *Q* is an association for which the following hold:

Multiple(*A*, *P*), *Zero*(*A*, *P*), not(*Multiple*(*A*, *Q*)), not(*Zero*(*A*, *Q*))

Most associations in a physical data model will be foreign key relations.

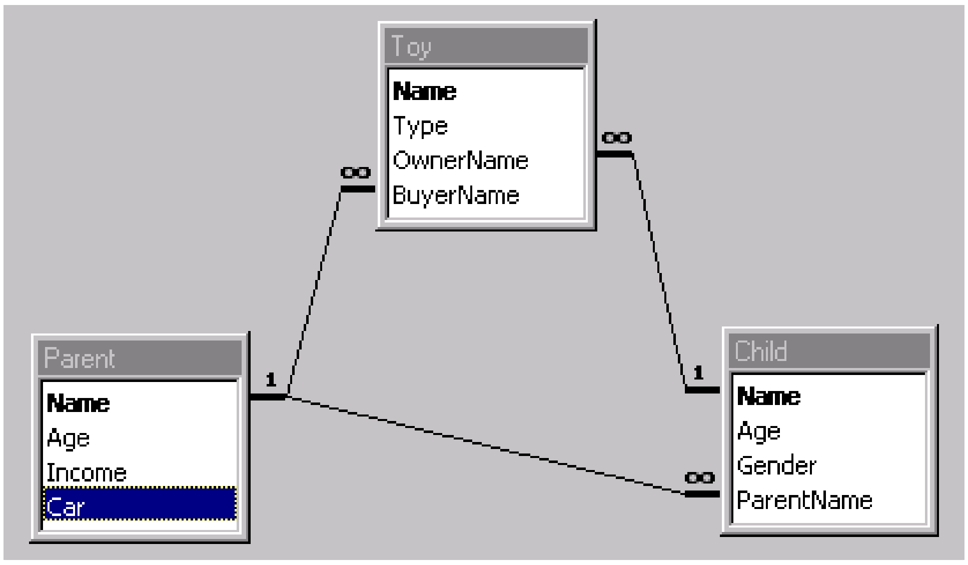
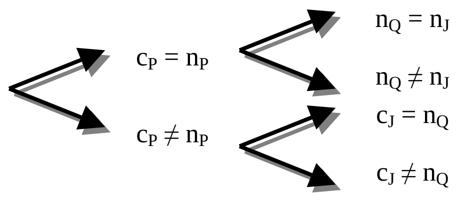


Figure 1 The data model of our example database.

Example 1 Figure 1 shows an example of a data model that describes parents, children and toys, as well as how each of these relate to each other. We will be referring to this example throughout this paper. The data model shows that parents may have 0 or more children, children may have 0 or more toys, and parents may have bought 0 or more toys. Note that toys owned by particular child may not necessarily have been bought by their parents. They can be presents from other parents. Also note that children have one parent (for simplicity).

Discovering multiplicities Associations between tables are part of the *conceptual* data model of a database. In the *physical* data model, the relationship between records is materialised by attributes in the two tables, which are known as *keys*. Keys may be built up of one or more attributes, but without loss of generality we will assume that they consist of a single attribute. The multiplicity of associations between tables is usually determined during the modelling phase of the database. However, the multiplicity can also be *discovered* from an existing database. Such a process is similar to the discovery of foreign key relations, described in [12]. Given the number of records (*n*) and the cardinality (*c*) of the key attribute of *P*, *Q* and the join *J* of *P* and *Q* respectively, we can use the following decision tree to determine the values of *Multiple*(*A*, *P*) and *Zero*(*A*, *P*) (analogous for *Q*).



		Zero(A,P)	*Multiple(A,P)*
$c_P = n_P$	$n_Q = n_J$	F	F
	$n_Q \neq n_J$	T	F
$c_P \neq n_P$	$c_J = n_Q$	F	T
	$c_J \neq n_Q$	T	T

Objects and patterns Even though the data model consists of multiple tables, there is still only a single kind of objects that is central to the analysis. You can choose the kind of objects you want to analyse, by selecting one of the tables as the *target table*. Each record in the target table, which we will refer to as t_0, will now corresponds to a single object in the database. Any information pertaining to the object which is stored in other tables can be looked up by following the associations in the data model. If the data mining algorithm requires a particular feature of an object to be used as a dependent attribute for classification or regression, we can define a particular *target attribute* within the target table.

The purpose of multi-relational data mining will be to discover interesting sets of objects in a relational database. We will refer to descriptions of potentially interesting sets of objects as *multi-relational patterns*, or simply *patterns* when clear from the context. To express a pattern one can use SQL or first order logic expressions. Instead, we will be using the graphical language of selection graphs, defined in the next section, which can be translated to either language. One can use multi-relational patterns when looking for interesting subgroups [21] or frequent patterns [17]. In the context of prediction, one can use multi-relational patterns as leaves in a decision tree, or as the left-hand side of predictive rules.

Definition 3 The *support* of a multi-relational pattern *P* in a database *D* is number of objects in *D* that are covered by *P*, which is equal to the number of selected records in the target table.

Example 2 In figure 1, the highlighting of the attribute *Car* in the *Parent* table indicates that *Parent* is the target table and *Car* is the attribute of interest, the target attribute. This means that our primary interest is in the parents, and that we will be considering different sets of parents in order to come up with good indicators for classifying parents as car owners or not.

Framework

In order to describe the set of conditions related to a multi-relational pattern, we introduce the concept of *selection graphs*:

Definition 4 A *selection graph G* is a pair (*N*, *E*), where *N* is a set of pairs (*t*, *C*), *t* is a table in the data model and *C* is a, possibly empty, set of conditions on attributes in *t* of type *t.a operator c*; the *operator* is one of the usual selection operators, =, >, etc. *E* is a set of triples (*p*, *q*, *a*) called *selection edges*, where *p* and *q* are selection nodes and *a* is an association between *p.t* and *q.t* in the data model. The selection graph contains at least one node n_0 that corresponds to the target table t_0. The objects in the target table that are covered by a selection graph *G* are those records in t_0:

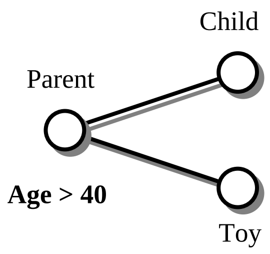
- that satisfy the conditions on t_0
- for which, recursively, there exist tuples in the other tables in the graph that are linked via the indicated associations and satisfy the conditions defined for that table.

Selection graphs represent selections of objects. The selection node *n* represents a selection of records in the corresponding table *n.t* which is determined by the set of conditions *n.C* and the relationship with records in other tables characterised by selection edges connected to *n*. Selection graphs are more intuitive than expressions in SQL or Prolog, because they reflect the structure of the data model, and refinements to existing graphs may be defined in terms of additions of edges and nodes.

A selection graph can be translated to SQL or Prolog in a straightforward manner. The following algorithm shows the translation to SQL. It will produce a list of tables *table_list*, a list of join conditions *join_list*, and a list of conditions *condition_list*, and combine these to produce a SQL-statement. A similar translation to Prolog can be made. The fact that we state `select distinct` rather than `select` is caused by the fact that a record in t_0 may be covered by the selection graph in multiple ways. Since the target table represents our objects, counting them once suffices.

```
table_list = ‘‘
condition_list = ‘‘
join_list = ‘‘
for each node i in selection graph S do
        table_list.add(i.table_name + ‘ T‘ + i)
        for each condition c in i do
                condition_list.add( ‘T‘ + i + ‘.‘ + c)
for each edge e in S do
    join_list.add(
        e.left_node + ‘.‘ +
        e.left_attribute + ‘ = ‘ +
        e.right_node + ‘.‘ +
        e.right_attribute)
return ‘select distinct ‘ +
    t0 + ‘.‘ + t0.primary_key +
    ‘ from ‘ + table_list +
    ‘ where ‘ + join_list +
    ‘ and ‘ + condition_list
```

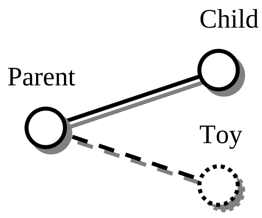
Example 3 The following selection graph, and its corresponding SQL statement, represents the set of parents older than 40, who have at least one child, and bought one toy (not necessarily for one of their own children):



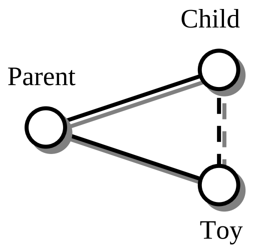
```
select distinct Parent.Name
from Parent T0, Child T1, Toy T2
where T0.Name = T1.ParentName and T0.Name = T2.BuyerName
and T0.Age > 40
```

Refinement. The multi-relational data mining framework proposed in this paper is based on the idea of a level-wise search for patterns [17], which is basically a top-down search. During the search, patterns will be considered and those that are promising will be refined. For a given selection graph *G* we define three refinement operations:

- **Add condition**. This refinement will simply add a condition to a selection node in *G* without actually changing the structure of *G*.
- **Add edge and node**. This refinement will instantiate an association in the data model as an edge together with its corresponding table and add these to *G*.



- **Add edge.** This refinement will add an edge between two nodes in *G*. This refinement is only valid when an association exists between the two corresponding tables in the data model.



This set of refinements is the main source of efficiency for our multi-relational data mining framework. By only allowing the addition of edges when consistent with the data model, unnecessary and invalid patterns will be pruned from the search space. The three refinements exploit the existence of associations between tables for optimisation. Below we will show how not only the existence, but also the multiplicity of an association can be used to achieve further efficiency.

Refinement & Multiplicity Which refinements are possible and the way the search algorithm uses these refinements is governed by the multiplicity of the available associations. Knowledge about the nature of associations between tables can therefore be used to guide and optimise the search. Note that once information about multiplicity is available, these decisions can be made in constant time. We will use this knowledge in three possible ways:

- **Look-ahead**. For some refinements to a multi-relational pattern, the set of objects that support a

pattern is not actually changed. Therefore, it may be desirable to use a look-ahead in the form of extra refinements [2]. Given the multiplicity of an association involved in a refinement, we can directly decide whether this refinement changes anything to the support of the pattern, and therefore whether the look-ahead is necessary. For example, if we know from the data model that all parents are required to have at least one child, we know that refining the pattern 'all parents' to 'all parents who have a child' will have no effect on the support of his pattern. The use of look-ahead when refining *G* from table *P* using an association *A* from *P* to *Q* is necessary when the following holds: not (*Zero*(*A*, *Q*)).
- **Multiple instantiations of associations**. In some cases it is desirable to have multiple instantiations of a particular association in the selection graph. For example, it may be interesting to consider parents with several children with various characteristics. Refining the pattern 'parents with a son' with the condition of having a daughter only make sense if parents are allowed to have multiple children. Multiple instantiations from an instance of table *P* to an instance of table *Q* are allowed when the following holds: *Multiple*(*A*, *Q*).
- **Mutual exclusion**. Some algorithms refine a single pattern into a set of derived patterns on the basis of the value of a single nominal attribute. The subsets belonging to these patterns do not necessarily form a partitioning of the original subset, in the sense that they may be overlapping. However, some algorithms, notably those for inducing decision trees, do require these subsets to be mutually exclusive. A partitioning of the original subset, by refining *G* from table *P* using an association *A* from *P* to *Q*, is produced when the following holds: not(*Multiple*(*A*, *Q*)).

Architecture

We will be using a client/server architecture to support the multi-relational data mining framework proposed in the previous section. The server will be responsible for storing and managing all the data, as well as computing the primitives, which will be explained in more detail later on in this section. The client on the other hand, will perform the actual search algorithm, which involves generating new candidates and sending primitive calls to the server in order to test these candidates. This way, a clear separation between handling massive datasets and performing intelligent search is achieved. Each operation can now be optimised separately.

A lot of work has been done on implementing efficient client/server architectures for mining attribute-value data [5, 7, 8, 9, 10, 11, 13]. Most of this work is centred around expressing primitives in SQL in order to use a conventional RDBMS, or extending the SQL language to support potential specific needs the data mining algorithm may have. An alternative approach is to use a dedicated data mining server which is optimised for the specific operations which are common in data mining. Such a data mining server will only support the loading of data (for example from a conventional RDBMS), and a small set of primitives for examining the data. All other operations that are common in conventional databases, such as transaction processing, locking, roll-back and ad hoc querying, are left out for optimal performance. In previous projects, we have worked on such a data mining architecture for attribute-value based data mining, according to the following design principles [8, 9, 13]:
- *Compute as much as possible during pre-processing.* Because datasets for data mining tend to be stable, one can benefit multiple times from prepossessing once. Specifically, the data can be coded such that it can be stored and indexed. Rather than using the original data, one can use the codes for indexing in cross tables etc., which is efficient for both storage and processing.
- *Exploit column oriented nature of KDD algorithms.* Most data mining algorithms use column oriented operations, rather than record oriented operations, which are more common in transaction processing. Therefore column oriented data storage and paging techniques will be attractive.
- *Exploit structure of search space.* Refinements of previously examined patterns will always be supported by a subset of the original set of supporting objects. Therefore, storing intermediate results for later use will reduce the amount of work for repeated primitives drastically.
- *Exploit inherent parallelism in algorithms*. Data mining algorithms are ideal candidates for parallelisation, due to the large set of extremely similar operations that have to be performed.

Such parallelism can be on a high level by partitioning the search space of candidate patterns over the available processors, or low-level by partitioning the data and performing a single primitive in parallel.

- *Optimise to low-level details*. This may involve disk access, effective paging and optimisation of the inner loops.

These design principles apply even more to the multi-relational data mining architecture. Specifically the first principle will be of great benefit. During pre-processing, all associations will be materialised, such that for a given record in one table, the list of corresponding records in the other table may be found in constant time. This means that partial joins are readily available, in contrast to conventional RDBMSs, where each query over multiple tables involves re-computing a join. Materialising all associations effectively means trading space for time. Materialising an association effectively means creating a lean indexing structure, rather than building the whole join between the two related tables. Only associations between pairs of tables will be materialised, so there is no combinatorial explosion.

Primitives The following multi-relational data mining primitives will be used in order to get the necessary counts from the database. These primitives are similar to those used in conventional data mining [7, 8, 9, 11], with extra facilities to cope with data in multiple tables. Note that although these primitives are represented using SQL, for better understanding, the actual architecture will provide optimised function calls for each of these primitives. All primitives produce counts in terms of the objects of interest, i.e. in terms of records in the target table:

CountSelection:

```
select count(distinct $t_0$.primary_key)
from table_list
where join_list and condition_list;
```

The CountSelection primitive is the basic primitive to compute the support of a given multi-relational pattern.

MultiRelationalHistogram:

```
select $t_0$.target, count(distinct $t_0$.primary_key)
from table_list
where join_list and condition_list
group by $t_0$.target;
```

The MultiRelationalHistogram primitive will produce the distribution of values for the target attribute within the set of objects belonging to the multi-relational pattern. It can be used to compute a range of interestingness-measures for this pattern. Note that the sum of the counts in the resulting multi-relational histogram is equal to the result of the CountSelection primitive.

MultiRelationalCrossTable:

```
select $t_0$.target, $t_i$.$c_j$, count(distinct $t_0$.primary_key)
from table_list
where join_list and condition_list
group by $t_0$.target, $t_i$.$c_j$;
```

The MultiRelationalCrossTable primitive will produce the distribution of pairs of values for the target attribute and an arbitrary nominal attribute $t_i.c_j$ in any of the tables. The resulting multi-relational cross table can be used to compute a range of statistical measures to describe the dependency between these two attributes. Note that the sum of these counts can exceed the support of the given multi-relational pattern, if the attribute $t_i.c_j$ is in a table which is not the target table. This is because multiple records, with different values for the selected attribute, may correspond to a single record in the target table, causing this record to contribute to multiple counts in the cross table.

MultiRelationalAggregateTable:

```
select $t_0$.target, m, count(*)
from
```

```
            (select t0.target, t0.primary_key, min(ti.cj) m
            from table_list
            where join_list and condition_list
            group by t0.target, t0.primary_key)
      group by t0.target, m;
```
The MultiRelationalAggregateTable primitive can be used to compute the dependency between a numeric attribute and the target attribute. We assume that each record in the target table has multiple associated records in table t_i. For each of these sets of records in table t_i we can compute the minimum for the attribute of interest $t_i.c_j$. The primitive produces a list of counts for pairs of values for the target attribute and each occurring minimum. The occurring minimums will be used to produce list of candidate tests on the attribute $t_i.c_j$. Here we use the fact that testing whether a set of values contains a value which is less than some threshold, is equal to testing whether the minimum of the values is less than the threshold. An analogous call for the maximum exists.

The MultiRelationalCrossTable and MultiRelationalAggregateTable primitives are used to evaluate a set of *add condition* refinements. A single call of the MultiRelationalCrossTable primitive for a nominal attribute X will supply the support of the set of patterns which are the refinements of the current pattern with the following condition: $X = x$, for every value x occurring in X. Similarly, a single call of the MultiRelationalAggregateTable primitive for a numeric attribute X will supply the support of the set of patterns which are the refinements of the current pattern with the following condition: $X < x$, for every interesting threshold x occurring in X, as explained above. The support of refinements with conditions of the type $X > x$ can be produced by the MultiRelationalAggregateTable primitive which uses `max`.

Example 4 The support of the pattern introduced in example 3 can be computed using the CountSelect primitive. In order to establish what portion of these parents do and do not own a car, one can use a single MultiRelationalHistogram call for the attribute *Car*. Using the MultiRelationalAggregateTable primitive, one can examine the effects the age of a child has on the ownership of a car. The primitive will produce a list of minimum ages within households and how frequent each minimum is, which can then be used to establish the interestingness of various thresholds on the age of children.

Instance

As an example of how a specific multi-relational data mining algorithm can be incorporated in the proposed framework, we consider the Warmr algorithm, which can be used to search for association rules over multiple relations [4]. The Warmr algorithm itself searches for patterns for which the support is above a given threshold. The resulting set of frequent patterns can be used to produce rules, just as in the single table counterpart Apriori.

For instance, during its search for interesting patterns Warmr might find the patterns P_1 = 'all people having a car ', P_2 = 'all people having a son of age at least 18' and P_3 = 'all people having a car and a son of age at least 18' to be of interest, because they have a sufficiently high support. If, moreover, P_3 (the intersection of P_1 and P_2) is unexpectedly large with respect to the support of P_2, this provides support for the claim that most people who have a son of at least 18 also have a car. This rule is then called a frequent multi-relational rule. Frequent multi-relational rules can be seen as a generalisation of association rules over a single relation.

The algorithm can be described in our framework as follows:

```
$F_1 = \{t_0\}$
$I_1 = \emptyset$
d = 2
while $F_{d-1} \neq \emptyset$
    for each g in $F_{d-1}$
        for each generic refinement R of g
            if R(g) is a candidate
                C = primitive(g, R)
                for each refinement r in R
                    if $C[r] \geq$ minsup
                        add r(g) to $F_d$
                    else
                        add r(g) to $I_d$
    d = d + 1
return $\cup_i F_i$
```

The algorithm is basically a level-wise search for patterns. At each level d, frequent patterns from the previous level (F_{d-1}) are used to produce candidates (as in Apriori). If such a candidate has not been considered yet, it will be validated with the database. Note how several potential add-condition refinements are tested by a single primitive call, using a single scan of the data. The type of primitive depends on the type of the attribute that is considered in the refinement. Each actual refinement will be added to either the set of frequent patterns F_d or the set of infrequent patterns I_d, depending on the support.

Conclusion

Traditional data mining algorithms work with data stored in attribute-value format, i.e. in a single table. This limits the class of objects that can be represented, and thus the type of knowledge that can be discovered. In this paper we have introduced a multi-relational data mining framework upon which a range of algorithms can be based that work with data which is stored in relational databases multiple tables. Such databases allow the representation of more complex and structured objects. The framework respects the underlying relation model, and only considers patterns which adhere to this model. Thus, the potentially huge search space of patterns is greatly reduced, which makes scaling up to large databases feasible.

Inspired by the language bias of most common ILP algorithms, we have introduced selection graphs to describe the patterns. As each node in a graph denotes an existential quantor, there is a strong focus on the *existence* of substructures as part of the objects. Rather than just testing the existence of substructures one could also consider richer languages of patterns that include *absence* of substructures and expressions over groups of substructures. We intend to examine extensions of the framework in the following three areas. Each of these concepts will be easily supportable by a dedicated data mining server:

- **Absence** Patterns may include conditions on the absence of related records: ‘all parents who do not have a child’.
- **Universal quantor** Patterns may include conditions over all related records: ‘all parents who have children that are adults’.
- **Aggregates** Patterns may include conditions over characteristics of groups of related records: ‘all parents that have three children’.

References

[1] Blockeel, H., De Raedt, L. *Relational knowledge discovery in databases*, Proceedings of the Sixth International Workshop on Inductive Logic Programming, Volume 314 of Lecture Notes in Artificial Intelligence, Springer Verlag, 1996

[2] Blockeel, H., De Raedt, L. *Top-down induction of first order logical decision trees*, Artificial Intelligence 101 (1-2), 1998

[3] Date, C.J. *An Introduction to Database Systems*, Volume I, The Systems Programming Series, Addison-Wesley, 1986

[4] Dehaspe, L., and De Raedt, L. *Mining association rules in multiple relations*, Proceedings of the Seventh International Workshop on Inductive Logic Programming, Volume 1297 of Lecture Notes in Artificial Intelligence, Springer Verlag, 1997

[5] Dewhurst, N., Lavington, S. *Knowledge Discovery from Client/Server Databases*, Proceedings PKDD ’98, 1998

[6] Dzeroski, S. *Inductive Logic Programming and Knowledge Discovery in Databases*, Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996

[7] Freitas, A.A., Lavington, S.H. *Using SQL-primitives and parallel DB servers to speed up knowledge discovery in large relational databases*, Proceedings EMCSR '96, 1996.

[8] George, F., Knobbe, A.J. *A Parallel Data Mining Architecture for Massive Data Sets*, To be published, 1999

[9] Hahn, M. *Info Charger Data Sheet*, http://www.tektonic.de/icdatash.htm, 1997

[10] Holsheimer, M., Kersten, M., Mannila, H., Toivonen, H. *A Perspective on Databases and Data Mining*, Proceedings KDD '95, 1995

[11] John, G.H., Lent, B. *SIPping from the Data Firehose*, Proceedings KDD '97, 1997

[12] Knobbe A.J., Adriaans, P.W. *Discovering Foreign Key Relations in Relational Databases*, Proceedings EMCSR '96, 1996

[13] Knobbe, A.J. *Towards Scalable Industrial Implementations of ILP*, ILPNet2 Seminar on ILP & KDD, Caminha, Portugal, 1998

[14] Lavrac, N., Dzeroski, S., and Grobelnik, M. *Learning nonrecursive definitions of relations with LINUS*, Proceedings Fifth European Working Session on Learning, Springer, Berlin, 1991

[15] Lavrac, N., Dzeroski, S. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994

[16] Lindner, G., Morik, K. *Coupling a relational learning algorithm with a database system*, Workshop Notes of the MLNet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, 1995

[17] Mannila, H., Toivonen, H. *On an algorithm for finding all interesting sentences*, Proceedings EMCSR '96, 1996

[18] Martin, L., Moal, F., Vrain, C. *A Relational Data Mining Tool Based on Genetic Programming*, Proceedings PKDD ’98, Nantes, France, 1998

[19] Ribeiro, J.S., Kaufman, K.A., and Kerschberg, L. *Knowledge discovery from multiple databases,* Proceedings of KDD'95, 1995

[20] Ullman, I.D. *Principles of Databases and Knowledge-Based Systems,* Volume I, Computer Science Press, 1988

[21] Wrobel, S. *An algorithm for multi-relational discovery of subgroups,* Proceedings PKDD '97, 1997

[22] Yao, I., Lin, H. *Searching Multiple Databases for Interesting Complexes,* Proceedings PAKDD '97, 1997