Propositionalisation and Aggregates

Arno J. Knobbe[1,2], Marc de Haas[3], Arno Siebes[2]

[1]Kiminkii, P.O. box 171, NL-3990 DD Houten, The Netherlands, a.knobbe@kiminkii.com
[2]Utrecht University, P.O. box 80 089, NL-3508 TB Utrecht, The Netherlands
siebes@cs.uu.nl
[3]Perot Systems Nederland B.V., P.O. box 2729, NL-3800 GG Amersfoort, The Netherlands
marc.dehaas@ps.net

Abstract The fact that data is scattered over many tables causes many problems in the practice of data mining. To deal with this problem, one either constructs a single table by hand, or one uses a Multi-Relational Data Mining algorithm. In this paper, we propose a different approach in which the single table is constructed automatically using aggregate functions, which repeatedly summarise information from different tables over associations in the datamodel. Following the construction of the single table, we apply traditional data mining algorithms. Next to an in-depth discussion of our approach, the paper presents results of experiments on three well-known data sets.

1 Introduction

An important practical problem in data mining is that we often want to find models and patterns over data that resides in multiple tables. This is solved by either constructing a single table by hand (deriving attributes from the other tables) or by using a Multi-Relational Data Mining or ILP approach. In this paper we propose another approach, viz., automatic construction of the single mining table using aggregates.

The motivation for the use of aggregates stems from the observation that the difficult case in constructing a single table is when there are one-to-many relationships between tables. The traditional way to summarise such relationships in Statistics and OLAP is through aggregates that are based on histograms, such as *count*, *sum*, *min*, *max*, and *avg*. We limit ourselves to these aggregates, but note that they can be applied recursively over a collection of relationships.

The idea of propositionalisation (the construction of one table) is not new. Several relatively successful algorithms have been proposed in the context of Inductive Logic Programming (ILP) [6, 12, 7, 1, 2]. A common aspect of these algorithms is that the derived table consists solely of binary features, each corresponding to a (promising) clause discovered by an ILP-algorithm. Especially for numerical attributes, our approach leads to a markedly different search space.

We illustrate our approach on three well-known data sets. The aim of these experiments is twofold. Firstly, to demonstrate the accuracy in a range of domains. Secondly, to illustrate the radically different way our approach models structured data, compared to ILP or MRDM approaches.

The paper is organised as follows. First we discuss propositionalisation and aggregates in more detail. In particular we introduce the notion of depth, to illustrate the complexity of the search space. Next we introduce the RollUp algorithm that constructs the single table. Then we present the results of our experiments and the paper ends with a discussion and conclusions.

2 Propositionalisation

In this section we describe the basic concepts involved in propositionalisation, and provide some definitions. In this paper, we define propositionalisation as the process of transforming a multi-relational dataset, containing structured examples, into a propositional dataset with derived attribute-value features, describing the structural properties of the examples. The process can thus be thought of as summarising data stored in multiple tables in a single table (*the target table*) containing one record per example. The aim of this process, of course, is to pre-process multi-relational data for subsequent analysis by attribute-value learners.

We will be using this definition in the broadest sense. We will make no assumptions about the datatype of the derived attribute (binary, nominal, numeric, etc.) nor do we specify what language will be used to specify the propositional features. Traditionally, propositionalisation has been approached from an ILP standpoint with only binary features, expressed in first-order logic (FOL)[6, 7, 1, 2]. To our knowledge, the use of other aggregates than existence has been limited. One example is given in [4], which describes a propositionalisation-step where numeric attributes were defined for counts of different substructures. [5] also mentions aggregates as a means of establishing probabilistic relationships between objects in two tables. It is our aim to analyse the applicability of a broader range of aggregates.

With a growing availability of algorithms from the fields of ILP and Multi-Relational Data Mining (MRDM), one might wonder why such a cumbersome pre-processing step is desirable in the first place, instead of applying one of these algorithms to the multi-relational data directly. The following is a (possibly incomplete) list of reasons:

- Pragmatic choice for specific propositional techniques. People may wish to apply their favourite attribute-value learner, or only have access to commercial of-the-shelf Data Mining tools. Good examples can be found in the contributions to the financial dataset challenge at PKDD conferences [14].
- Superiority of propositional algorithms with respect to certain Machine Learning parameters. Although extra facilities are quickly being added to existing ILP engines, propositional algorithms still have a head-start where it concerns handling of numeric values, regression, distance measures, cumulativity etc.
- Greater speed of propositional algorithms. This advantage of course only holds if the preceding work for propositionalisation was limited, or performed only once and then reused during multiple attribute-value learning sessions.
- Advantages related to multiple consecutive learning steps. Because we are applying two learning steps, we are effectively combining two search strategies. The first step essentially transforms a multi-relational search space into a propositional one. The second step then uses these complex patterns to search

deeper than either step could achieve when applied in isolation. This issue is investigated in more detail in the remainder of this section.

The term propositionalisation leads to some confusion because, although it pertains to the initial step of flattening a multi-relational database, it is often used to indicate the whole approach, including the subsequent propositional learning step. Because we are mostly interested in the two steps in unison, and for the sake of discussion, we introduce the following generic algorithm. The name is taken from Czech, and indicates a two-step dance.

```
Polka (DB D; DM M; int r, p)
    P := MRDM (D, M, r);
    R := PDM (P, p);
```

The algorithm takes a database D and datamodel M (acting as declarative bias), and first applies a Multi-Relational Data Mining algorithm MRDM. The resulting propositional features P are then fed to a propositional Data Mining algorithm PDM, producing result R. We use the integers r and p very informally to identify the extent of the multi-relational and propositional search, respectively. Note that the propositionalisation step is independent of subsequent use in propositional learning.

In order to characterise more formally the extent of the search, we introduce three measures that are functions of the patterns that are considered. The values of the measures for the most complex patterns in the search space are then measures for the extent of the search algorithm. We can thus characterise both individual patterns, as well as algorithms. The definition is based on the graphical pattern language of Selection Graphs, introduced in [8], but can be re-written in terms of other languages such as FOL, relational algebra or SQL. We first repeat our basic definition of Selection Graphs.

definition 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 (the root node) that corresponds to the target table t_0.

Now assume *G* is a Selection Graph.

definition variable-depth: d_v (*G*) equals the length of the longest path in *G*.

definition clause-depth: d_c (*G*) equals the sum of the number of non-root nodes, edges and conditions in *G*.

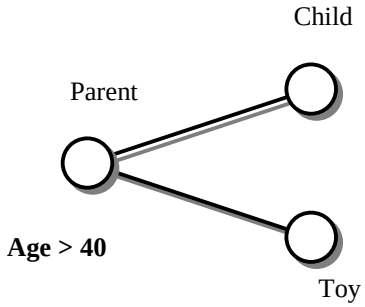
definition variable-width: w_v (*G*) equals the largest sum of the number of conditions and children per node, not including the root-node.

The intuition of these definitions is as follows. An algorithm searches *variable-deep*, if pieces of discovered substructure are refined by adding more substructure,

resulting in chains of variables (edges in Selection Graphs). With each new variable, information from a new table is involved. An algorithm searches *clause-deep*, if it considers very specific patterns, regardless of the number of tables involved. Even propositional algorithms may produce clause-deep patterns that contain many conditions at the root-node and no other nodes. Rather than long chains of variables, *variable-wide* algorithms are concerned with the frequent reuse of a single variable. If information from a new table is included, it will be further refined by extra restrictions, either through conditions on this information, or through further substructure.

example The following Selection Graph, which refers to a 3-table database introduced in [8], identifies parents above 40 who have a child and bought a toy. The measures produce the following complexity characteristics:

$$d_v(G)=1,\ d_c(G)=5,\ w_v(G)=0$$



The complexity measures can now be used to relate the search depth of Polka to the propositional and multi-relational algorithm it is made up of.

lemma 1 d_v (Polka) = d_v (MRDM)
lemma 2 d_c (Polka) = d_c (MRDM) · d_c (PDM)
lemma 3 w_v (Polka) = w_v (MRDM)

Not surprisingly, the complexity of Polka depends largely on the complexity of the actual propositionalisation step. However, lemma 2 demonstrates that Polka considers very clause-deep patterns, in fact deeper than a multi-relational algorithm would consider in isolation. This is due to the combining of search spaces mentioned earlier. Later on we will examine the search restrictions that the use of aggregates have on the propositionalisation step and thus on Polka.

3 Aggregates

In the previous section we observed that an essential element of propositionalisation is the ability to summarise information distributed over several tables in the target table. We require functions that can reduce pieces of substructure to a single value, which describes some aspects of this substructure. Such functions are called aggregates. Having a set of well-chosen aggregates will allow us to describe the essence of the structural information over a wide variety of structures.

We define an aggregate as a function that takes as input a set of records in a database, related through the associations in the data model, and produces a single value as output. We will be using aggregates to project information stored in several tables on one of these tables, essentially adding virtual attributes to this table. In the case where the information is projected on the target table, and structural information

belonging to an example is summarised as a new feature of that example, aggregates can be thought of as a form of feature construction.

Our broad definition includes aggregates of a great variety of complexity. An important aspect of the complexity of an aggregate is the number of (associations between) tables it involves. As each aggregate essentially considers a subset of the data model, we can use our 3 previously defined complexity-measures for data models to characterise aggregates. Specifically variable-depth is useful to classify aggregates. An aggregate of variable-depth 0 involves just one table, and is hence a case of propositional feature construction. In their basic usage, aggregates found in SQL (count, min, sum, etc.) have a variable-depth of 1, whereas variable-deeper aggregates represent some form of Multi-Relational pattern (benzene-rings in molecules, etc.). Using this classification of variable-depth we give some examples to illustrate the range of possibilities.

$d_v(A) = 0$:
- Propositions (adult == (age $\geq$ 18))
- Arithmetic functions (area == width·length)

$d_v(A) = 1$:
- Count, count with condition
- Count distinct
- Min, max, sum, avg
- Exists, exists with condition
- Select record (eldest son, first contract)
- Predominant value

$d_v(A) > 1$:
- Exists substructure
- Count substructure
- Conjunction of aggregates (maximum count of children)

Clearly the list of possible classes of aggregates is long, and the number of instances is infinite. In order to arrive at a practical and manageable solution for propositionalisation we will have to drastically limit the range of classes and instances. Apart from deterministic and heuristic rules to select good candidates, pragmatic limitations to a small set of aggregate classes are unavoidable. In this paper we have chosen to restrict ourselves to the classes available in SQL, and combinations thereof. The remainder of this paper further investigates the choice of instances.

4 Summarisation

We will be viewing the propositionalisation process as a series of steps in which information in one table is projected onto records in another table successively. Each association in the data model gives rise to one such step. The specifics of such a step, which we will refer to as summarisation, are the subject of this section.

Let us consider two tables *P* and *Q*, neither of which needs to be the target table, that are joined by an association *A*. By summarising over *A*, information can be added to *P* about the structural properties of *A*, as well as the data within *Q*. To summarise *Q*, a set of aggregates of variable-depth 1 are needed.

As was demonstrated before in [8], the multiplicity of association *A* influences the search space of multi-relational patterns involving *A*. The same is true for summarisation over *A* using aggregates. Our choice of aggregates depends on the multiplicity of *A*. In particular if we summarise *Q* over *A* only the multiplicity on the side of *Q* is relevant. This is because an association in general describes two relationships between the records in both tables, one for each direction. The following four options exist:

1 For every record in *P* there is but a single record in *Q*. This is basically a look-up over a foreign key relation and no aggregates are required. A simple join will add all non-key attributes of *Q* to *P*.

0..1 Similar to the 1 case, but now a look-up may fail because a record in *P* may not have a corresponding record in *Q*. An outer join is necessary, which fills in NULL values for missing records.

1..n For every record in *P*, there is at least one record in *Q*. Aggregates are required in order to capture the information in the group of records belonging to a single record in *P*.

0..n Similar to the 1..n case, but now the value of certain aggregates may be undefined due to empty groups. Special care will need to be taken to deal with the resulting NULL values.

Let us now consider the 1..n case in more detail. *A* imposes a grouping on the records in *Q*. For *m* records in *P* there will be *m* groups of records in *Q*. Because of the set-semantics of relational databases every group can be described by a collection of histograms or data-cubes. We can now view an aggregate instance as a function of one of these types of histograms. For example the *predominant* aggregate for an attribute *Q.a* simply returns the value corresponding to the highest count in the histogram of *Q.a*. Note that m groups will produce m histograms and thus m values for one aggregate instance, one for each record in *P*. The notion of functions of histograms helps us to define relevant aggregate classes.

count The *count* aggregate is the most obvious aggregate through its direct relation to histograms. The most basic instance without conditions simply returns the single value in the 0-dimensional histogram. Adding a single condition requires a 1-dimensional histogram of the attribute involved in the condition. For example the number of sons in a family can be computed from a histogram of gender of that family. An attribute with a cardinalty *c* will produce *c* aggregate instances of *count* with one condition. It is clear that the number of instances will explode if we allow even more conditions. As our final propositional dataset will then become impractically large we will have to restrict the number of instances. We will only consider counts with no condition and counts with one condition on nominal attributes. This implies that for the *count* aggregate $w_v \leq 1$.

There is some overlap in the patterns that can be expressed by using the *count* aggregate and those expressed in FOL. Testing for a count greater than zero obviously corresponds to existence. Testing for a count greater than some threshold *t* however,

requires a clause-depth of $O(t^2)$ in FOL. With the less-than operator things become even worse for FOL representations as it requires the use of negation in a way that the language bias of many ILP algorithms does not cater for. The use of the *count* aggregate is clearly more powerful in these respects.

min and max The two obvious aggregates for numeric attributes, *min* and *max*, exhibit similar behaviour. Again there is a trivial way of computing *min* and *max* from the histogram; the smallest and largest value for which there is a non-zero count, respectively. The *min* and *max* aggregates support another type of constraint commonly used in FOL-based algorithms, *existence* with a numeric constraint. The following proposition describes the correspondence between the minimum and maximum of a group of numbers, and the occurrence of particular values in the group.

proposition Let B be a bag of real numbers, and t some real, then

$$\max(B) > t \quad \text{iff} \quad \exists\, v \in B : v > t,$$
$$\min(B) < t \quad \text{iff} \quad \exists\, v \in B : v < t.$$

This simply states that testing whether the maximum is greater than some threshold is equivalent to testing whether any value is greater than t. Analogous for *min*. It is important to note the combination of *max* and >, and *min* and < respectively. If *max* were to be used in combination with < or = then the FOL equivalent would again require the use of negation. Such use of the *min* and *max* aggregate gives us a natural means of introducing the universal quantor ∀: all values are required to be above the minimum, or below the maximum. Another advantage of the *min* and *max* aggregate is that they each replace a set of binary *existence* aggregate instances (one for each threshold), making the propositional representation a lot more compact.

In short we can conclude that on the level of summarisation ($d_v = 1$) aggregates can express many of the concepts used in FOL. They can even express concepts that are hard or impossible to express in FOL. The most important limitation of our choice of aggregate instances is the number of attributes involved: $w_v \leq 1$. This restriction prevents the use of combinations of attributes which cause a combinatorial explosion of features [10].

5 The RollUp Algorithm

With the basic operations provided in the previous sections we can now define a basic propositionalisation algorithm. The algorithm will traverse the data model graph and repeatedly use the summarisation operation to project data from one table onto another, until all information has been aggregated at the target table. Although this repeated summarisation can be done in several ways, we will describe a basic algorithm, called RollUp.

The RollUp algorithm performs a depth-first search (DFS) through the data model, up to a specified depth. Whenever the recursive algorithm reaches its maximum depth or a leaf in the graph, it will "roll up" the relevant table by summarising it on the parent in the DFS tree. Internal nodes in the tree will again be summarised after all its children have been summarised. This means that attributes considered deep in the tree may be aggregated multiple times. The process continues until all tables are

summarised on the target table. In combination with a propositional learner we have an instance of Polka. The following pseudo code describes RollUp more formally:

```
RollUp (Table T, Datamodel M, int d)
    V := T;
    if d > 0
        for all associations A from T in M
            W := RollUp(T.getTable(A), M, d-1);
            S := Summarise(W, A);
            V.add(S);
    return V;
```

The effect of RollUp is that each attribute appearing in a table other than the target table will appear several times in aggregated form in the resulting view. This multiple occurrence happens for two reasons. The first reason is that tables may occur multiple times in the DFS tree because they can be reached through multiple paths in the datamodel. Each path will produce a different aggregation of the available attributes. The second reason is related to the choices of aggregate class at each summarisation along a path in the datamodel. This choice, and the fact that aggregates may be combined in longer paths produces multiple occurrences of an attribute per path.

The variable-depth of the deepest feature is equal to the parameter *d*. Each feature corresponds to at most one attribute aggregated along a path of depth d_v. The clause-depth is therefore a linear function of the variable-depth. As each feature involves at most one attribute, and is aggregated along a path with no branches, the variable-width will always be either 0 or 1. This produces the following characteristics for RollUp. Use lemmas 1 to 3 to characterise Polka instantiated with RollUp.

lemma 4 d_v (RollUp) = *d*
lemma 5 d_c (RollUp) = $2 \cdot d_v$ (RollUp) + 1
lemma 6 w_v (RollUp) = 1

6 Experiments

In order to acquire empirical knowledge about the effectiveness of our approach, we have tested RollUp on three well-known multi-relational datasets. These datasets were chosen because they show a variety of datamodels that occur frequently in many multi-relational problems. They are Musk [3], Mutagenesis [11], and Financial [14].

Each dataset was loaded in the RDBMS Oracle. The data was modelled in UML using the multi-relational modelling tool Tolkien (see [9]) and subsequently translated to CDBL. Based on this declarative bias, the RollUp module produced one database view for each dataset, containing the propositionalised data. This was then taken as input for the common Machine Learning procedure C5.0.

For quantitative comparison with other techniques, we have computed the average accuracy by leave-one-out cross-validation for Musk and Mutagenesis, and by 10-fold cross-validation for Financial.

6.1 Musk

The Musk database [3] describes molecules occurring in different conformations. Each molecule is either *musk* or *non-musk* and one of the conformations determines this property. Such a problem is known as a multiple-instance problem, and will be modelled by two tables **molecule** and **conformation**, joined by a one-to-many association. **Confirmation** contains a molecule identifier plus 166 continuous features. **Molecule** just contains the identifier and the class. We have analysed two datasets, MuskSmall, containing 92 molecules and 476 confirmations, and MuskLarge, containing 102 molecules and 6598 confirmations. The resulting table contains a total of 674 features.

Table 1 shows the results of RollUp compared to other, previously published results. The performance of RollUp is comparable to Tilde, but below that of special-purpose algorithms.

Algorithm	MuskSmall	MuskLarge
Iterated-discrim APR	92.4%	89.2%
GFS elim kde APR	91.3%	80.4%
RollUp	**89.1%**	**77.5%**
Tilde	87.0%	79.4%
Back-propagation	75.0%	67.7%
C4.5	68.5%	58.8%

Table 1 Results on Musk

6.2 Mutagenesis

Similar to the Musk database, the Mutagenesis database describes molecules falling in two classes, *mutagenic* and *non-mutagenic*. However, this time structural information about the atoms and bonds that make up the compound are provided. As chemical compounds are essentially annotated graphs, this database is a good test-case for how well our approach deals with graph-data. The dataset we have analysed is known as the 'regression-friendly' dataset, and consists of 188 molecules. The database consists of 26 tables, of which three tables directly describe the graphical structure of the molecule (**molecule**, **atom** and **bond**). The remaining 23 tables describe the occurrence of predefined functional groups, such as benzene rings.

Four different experiments will be performed, using different settings, or so-called *backgrounds*. They will be referred to as experiment B1 to B4:

- B1: the atoms in the molecule are given, as well as the bonds between them; the type of each bond is given as well as the element and type of each atom.
- B2: as B1, but continuous values about the charge of atoms are added.
- B3: as B2, but two continuous values describing each molecule are added.
- B4: as B3, but knowledge about functional groups is added.

The largest resulting table, for B4, contains 309 constructed features.

Table 2 shows the results of RollUp compared to other, previously published results. Clearly RollUp outperforms the other methods on all backgrounds, except B4.

Most surprisingly, RollUp already performs well on B1, whereas the ILP methods seem to benefit from the propositional information provided in B3.

	Progol	FOIL	Tilde	**RollUp**
B1	76%	61%	75%	**86%**
B2	81%	61%	79%	**85%**
B3	83%	83%	85%	**89%**
B4	88%	82%	86%	**84%**

Table 2 Results on Mutagenesis

example The following tree of the B3 experiment illustrates the use of aggregates for structural descriptions.

```
CNT_BOND =< 26
    PREDOMINANT_TYPE_ATOM [21 27] -> F
    PREDOMINANT_TYPE_ATOM 22 -> F
    PREDOMINANT_TYPE_ATOM 3
        MAX_CHARGE_ATOM =< 0.0
            PREDOMINANT_TYPE_BOND 7 -> F
            PREDOMINANT_TYPE_BOND 1 -> T
        MAX_CHARGE_ATOM  > 0.0 -> F
CNT_BOND  > 26
    LUMO =< -1.102
        LOGP =< 6.26 -> T
        LOGP  > 6.26 -> F
    LUMO  > -1.102 -> F
```

6.3 Financial

Our third database is taken from the Discovery Challenge organised at PKDD '99 and PKDD 2000 [14]. The database is based on data from a Czech bank. It describes the operations of 5369 clients holding 4500 accounts. The data is stored in 8 tables, 4 of which describe the usage of products, such as credit cards and loans. Three tables describe client and account information, and the remaining table contains demographic information about 77 Czech districts. We have chosen the **account** table as our target table. Although we thus have 4500 examples, the dataset contains a total of 1079680 records. Our aim was to determine the loan-quality of an account, that is the existence of a loan with status 'finished good loan' or 'running good loan'. The resulting table contains a total of 148 features.

A near perfect score of 99.9% was achieved on the Financial dataset. Due to the great variety of problem definitions described in the literature, quantitative comparisons with previous results are impossible. Similar (descriptive) analyses of loan-quality however never produced the pattern responsible for RollUp's performance. The aggregation approach proved particularly successful on the large **transaction** table (1056320 records). This table has sometimes been left out of other experiments due to scalability problems.

7 Discussion

The experimental results in the previous section demonstrate that our approach is at least competitive with existing multi-relational techniques, such as Progol and Tilde. Our approach has two major differences with these techniques, which may be the source of the good performance: the use of aggregates and the use of propositionalisation. Let us consider the contribution of each of these in turn.

Aggregates There is an essential difference in the way a group of records is characterised by FOL and by aggregates. FOL characterisation are based on the occurrence of one or more records in the group with certain properties. Aggregates on the other hand typically describe the group as a whole; each record has some influence on the value of the aggregate. The result of this difference is that FOL and aggregates provide two unique feature-spaces to the learning procedure. Each feature-space has its advantages and disadvantages, and may be more or less suitable for the problem at hand.

Although the feature-spaces produced by FOL and aggregates have entirely different characteristics, there is still some overlap. As was shown in section 4, some aggregates are very similar in behaviour to FOL expressions. The common features in the two spaces typically

- select one or a few records in a group (*min* and <, *max* and >, *count* > 0 for some condition).
- involve a single attribute: $w_v \leq 1$
- have a relatively low variable-depth.

If these properties hold, aggregate-based learning procedures will generally perform better, as they can dispose of the common selective aggregates, as well as the complete aggregates such as *sum* and *avg*.

Datamodels with a low variable depth are quite common in database design, and are called star-shaped ($d_v = 1$) or snowflake schemata ($d_v > 1$). The Musk dataset is the most simple example of a star-shaped model. The datamodel of Mutagenesis consists for a large part of a star-shaped model, and Financial is essentially a snowflake schema. Many real-world datasets described in the literature as ILP applications essentially have such a manageable structure. Moreover, results on these datasets frequently exhibit the extra condition of $w_v \leq 1$. Some illustrative examples are given in [4, 13].

Propositionalisation According to lemma 2, Polka has the ability to discovery patterns that have a bigger clause-depth than either of its steps has. This is demonstrated by the experiments with our particular instance of Polka. RollUp produces variable-deep and thus clause-deep features. These clause-deep features are combined in the decision tree. Some leafs represent very clause-deep patterns, even though their support is still sufficient. This is an advantage of Polka (propositionalisation + propositional learning) over multi-relational algorithms in general.

Next to advantages related to expressivity, there are more practical reasons for using Polka. Once the propositionalisation-stage is finished, a large part of the computationally expensive work is done, and the derived view can be analysed

multiple times. This not only provides a greater efficiency, but gives the analyst more flexibility in choosing the right modelling technique from a large range of well-developed commercially available set of tools. The analyst can vary the style of analysis (trees, rules, neural, instance-based) as well as the paradigm (classification, regression).

8 Conclusion

We have presented a method that uses aggregates to propositionalise a multi-relational database, such that the resulting view can be analysed by existing propositional methods. The method uses information from the datamodel to guide a process of repeated summarisation of tables on other tables. The method has shown good performance on three well-known datasets, both in terms of accuracy as well as in terms of speed and flexibility.

The experimental findings are supported by theoretical results, which indicate the strength of this approach on so-called star-shaped or snowflake datamodels. We have also given evidence for why propositionalisation approaches in general may outperform ILP or MRDM systems, as was suggested before in the literature [4, 12].

9 References

1. Alphonse, É., Rouveirol, C. *Selective Propositionalization for Relational Learning*, In Proceedings of PKDD '99, 1999
2. Dehaspe, L., Toivonen, H., *Discovery of frequent Datalog patterns*, Data Mining and Knowledge Discovery 3(1), 1999
3. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T., *Solving the multiple-instance problem with axis-parallel rectangles*, Artificial Intelligence, 89(1-2):31-71, 1997
4. Džeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., Van Laer, W., *Experiments in Predicting Biodegradability*, In Proceedings of ILP '99, 1999
5. Friedman, N., Getoor, L., Koller, D., Pfeffer, A., *Learning Probabilistic Relational Models*, In Proceedings of IJCAI '99, 1999
6. Kramer, S., *Relational Learning vs. Propositionalization*, Ph.D thesis, 1999
7. Kramer, S., Pfahringer, B., Helma, C., *Stochastic Propositionalization of non-determinate background knowledge*, In Proceedings of ILP '98, 1998
8. Knobbe, A.J., Blockeel, H., Siebes, A., Van der Wallen, D.M.G. *Multi-Relational Data Mining*, In Proceedings of Benelearn '99, 1999
9. Knobbe, A.J., Siebes, A., Blockeel, H., Van der Wallen, D.M.G., *Multi-Relational Data Mining, using UML for ILP*, In Proceedings of PKDD 2000, 2000
10. Lavrač, N., Džeroski, S., Grobelnik, M., *Learning nonrecursive definitions of relations with LINUS*, In Proceedings of EWSL'91, 1991
11. Srinivasan, A., King, R.D., *Feature construction with ILP: a study of quantitative predictions of biological activity by structural attributes*, In Proceedings of ILP '96, 1996
12. Srinivasan, A., King, R.D., Bristol, D.W., *An Assessment of ILP-Assisted Models for Toxicology and the PTE-3 Experiment*, In Proceedings of ILP '99, 1999
13. Todorovski, L., Džeroski, S., *Experiments in Meta-level Learning with ILP*, In Proceedings of PKDD '99, 1999
14. Workshop notes on Discovery Challenge PKDD '99, 1999