Comparative Evaluation of Approaches to Propositionalization

Mark-A. Krogel[1], Simon Rawles[2], Filip Železný[3,4], Peter A. Flach[2], Nada Lavrač[5], and Stefan Wrobel[6,7]

[1] Otto-von-Guericke-Universität, Magdeburg, Germany
mark.krogel@iws.cs.uni-magdeburg.de
[2] University of Bristol, Bristol, UK
peter.flach@bristol.ac.uk
simon.rawles@bristol.ac.uk
[3] Czech Technical University, Prague, Czech Republic
zelezny@fel.cvut.cz
[4] University of Wisconsin, Madison, USA
zelezny@biostat.wisc.edu
[5] Institute Jožef Stefan, Ljubljana, Slovenia
nada.lavrac@ijs.si
[6] Fraunhofer AiS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany
stefan.wrobel@ais.fraunhofer.de
[7] Universität Bonn, Informatik III, Römerstr. 164, 53117 Bonn, Germany
stefan.wrobel@cs.uni-bonn.de

Abstract. Propositionalization has already been shown to be a promising approach for robustly and effectively handling relational data sets for knowledge discovery. In this paper, we compare up-to-date methods for propositionalization from two main groups: logic-oriented and database-oriented techniques. Experiments using several learning tasks — both ILP benchmarks and tasks from recent international data mining competitions — show that both groups have their specific advantages. While logic-oriented methods can handle complex background knowledge and provide expressive first-order models, database-oriented methods can be more efficient especially on larger data sets. Obtained accuracies vary such that a combination of the features produced by both groups seems a further valuable venture.

1 Introduction

Following the initial success of the system LINUS [13], approaches to multi-relational learning based on propositionalization have gained significant new interest in the last few years. In a multi-relational learner based on propositionalization, instead of searching the first-order hypothesis space directly, one uses a transformation module to compute a large number of propositional features and then uses a propositional learner. While less powerful in principle than systems that directly search the full first-order hypothesis space, it has turned out that in practice, in many cases it is sufficient to search a fixed subspace that can be defined by feature transformations. In addition, basing learning on such transformations offers a potential for enhanced efficiency which is becoming more and more important for large applications in data mining. Lastly, transforming multi-relational problems into a single table format allows one to directly use all propositional learning systems, thus making a wider choice of algorithms available.

In the past three years, quite a number of different propositionalization learners have been proposed [1, 10, 9, 12, 14, 17]. While all such learning systems explicitly or implicitly assume that an individual-centered representation is used, the available systems differ in their details. Some of them constrain themselves to features that can be defined in pure logic (existential features), while others, inspired by the database area, include features based on e.g. aggregation. Unfortunately, in the existing literature, only individual empirical evaluations of each system are available, so it is difficult to clearly see what the advantages and disadvantages of each system are, and on which type of application each one is particularly strong.

In this paper, we therefore present the first comparative evaluation of three different multi-relational learning systems based on propositionalization. In particular, we have chosen to compare the systems RSD [17], a subgroup discovery system of which we are interested in its feature construction part, SINUS, the successor of LINUS and DINUS [13], and RELAGGS [12], a database-inspired system which adds non-existential features. We give details on each system, and then, in the main part of the paper, provide an extensive empirical evaluation on six popular multi-relational problems. As far as possible, we have taken great care to ensure that all systems use identical background knowledge and declarations to maximize the strength of the empirical results. Our evaluation shows interesting differences between the involved systems, indicating that each has its own strengths and weaknesses and that neither is universally the best. In our discussion, we analyze this outcome and point out which directions of future research appear most promising.

The paper is structured as follows. In the following section (Sect. 2), we first recall the basics of propositionalization as used for multi-relational learning. In the subsequent sections, we then discuss each of the three chosen systems individually, first RSD (Sect. 3.1), then SINUS (Sect. 3.2), and finally RELAGGS (Sect. 4). Section 5 is the main part of the paper, and presents an empirical evaluation of the approaches. We give details on the domains that were used, explain how the domains were handled by each learning system, and present a detailed comparison of running times and classification accuracies. The results show noticeable differences between the systems, and we discuss the possible reasons for their respective behavior. We finish with a summary and conclusion in Sect. 6, pointing out some areas of further work.

2 Propositionalization

Following [11], we understand propositionalization as a transformation of relational learning problems into attribute-value representations amenable for conventional data mining systems such as C4.5 [21], which can be seen as propositional learners. Attributes are often called features and form the basis for columns in single table representations of data. Single-table representations and models that can be learned from them have a strong relationship to propositional logic and its expressive power [7], hence the name for the approaches discussed here. As further pointed out there, propositionalization can mostly be applied in domains with a clear notion of individual with learning occurring on the level of individuals only.

We focus in this paper on the same kind of learning tasks as [11]:

Given some evidence E (examples, given extensionally either as a set of ground facts or tuples representing a predicate/relation whose intensional definition is to be learned),
and an initial theory B (background knowledge, given either extensionally as a set of ground facts, relational tuples or sets of clauses over the set of background predicates/relations)

Find a theory H (hypothesis, in the form of a set of logical clauses) that together with B *explains* some properties of E.

Usually, hypotheses have to obey certain constraints in order to arrive at hypothesis spaces that can be handled efficiently. These restrictions can introduce different kinds of bias.

During propositionalization, features are constructed from relational background knowledge and structural properties of individuals. Results can then serve as input to different propositional learners, e.g. as preferred by the user.

Propositionalizations can be either complete or partial (heuristic). In the former case, no information is lost in the process; in the latter, information is lost and the representation change is incomplete: the goal is to automatically generate a small but relevant set of structural features. Further, general-purpose approaches to propositionalization can be distinguished from special-purpose approaches that could be domain-dependent or applicable to a limited problem class only.

In this paper, we focus on general-purpose approaches for partial propositionalization. In partial propositionalization, one is looking for a set of features, where each feature is defined in terms of a corresponding program clause. If the number of features is m, then a propositionalization of the relational learning problem is a set of clauses:

$$
\begin{aligned}
&f_1(X) : -Lit_{1,1}, \ldots, Lit_{1,n_1}.\\
&f_2(X) : -Lit_{2,1}, \ldots, Lit_{2,n_2}.\\
&\ldots\\
&f_m(X) : -Lit_{m,1}, \ldots, Lit_{m,n_m}.
\end{aligned}
$$

where each clause defines a feature f_i. Clause body $Lit_{i,1}, ..., Lit_{i,n}$ is said to be the definition of feature f_i; these literals are derived from the relational background knowledge. In clause head $f_i(X)$, argument X refers to an individual. If such a clause is called for a particular individual (i.e., if X is bound to some example identifier) and this call succeeds at least once, the corresponding Boolean feature is defined to be "true" for the given example; otherwise, it is defined to be "false".

It is pointed out in [11] that features can also be non-Boolean requiring a second variable in the head of the clause defining the feature to return the value of the feature. The usual application of features of this kind would be in situations where the second variable would have a unique binding. However, variants of those features can also be constructed for non-determinate domains, e.g. using aggregation as described below.

3 Logic-Oriented Approaches

The next two presented systems, RSD and SINUS, tackle the propositionalization task by constructing first-order logic features, assuming – as mentioned earlier – there is a clear notion of a distinguishable individual. In this approach to first-order feature construction, based on [8, 11, 14], local variables referring to parts of individuals are introduced by the so-called *structural predicates*. The only place where non-determinacy can occur in individual-centered representations is in structural predicates. Structural predicates introduce new variables. In the proposed language bias for first-order feature construction, a first-order feature is composed of one or more structural predicates introducing a new variable, and of *utility predicates* as in LINUS [13] (called *properties* in [8]) that 'consume' all new variables by assigning properties to individuals or their parts, represented by variables introduced so far. Utility predicates do not introduce new variables.

Although the two systems presented below are based on a common understanding of the notion of a first-order feature, they vary in several aspects. We first overview their basic principles separately and then compare the approaches.

3.1 RSD

RSD has been originally designed as a system for relational subgroup discovery [17]. Here we are concerned only with its auxiliary component providing means of first-order feature construction. The RSD implementation in the Yap Prolog is publicly available from `http://labe.felk.cvut.cz/~zelezny/rsd`, and accompanied by a comprehensive user's manual.

To propositionalize data, RSD conducts the following three steps.

1. Identify all first-order literal conjunctions which form a legal feature definition, and at the same time comply to user-defined constraints (mode-language). Such features do not contain any constants and the task can be completed independently of the input data.

2. Extend the feature set by variable instantiations. Certain features are copied several times with some variables substituted by constants detected by inspecting the input data. During this process, some irrelevant features are detected and eliminated.
3. Generate a propositionalized representation of the input data using the generated feature set, i.e., a relational table consisting of binary attributes corresponding to the truth values of features with respect to instances of data.

Syntactical construction of features. RSD accepts declarations very similar to those used by the systems Aleph [22] and Progol [19], including variable types, modes, setting a *recall* parameter etc., used to syntactically constrain the set of possible features. Let us illustrate the language bias declarations by an example on the well-known East-West Trains data domain [18].

– A structural predicate declaration in the East-West trains domain can be defined as follows:

  ```
  :-modeb(1, hasCar(+train, -car)).
  ```

  where the recall number 1 determines that a feature can address at most one car of a given train. Input variables are labelled by the `+` sign, and output variables by the `-` sign.
– Property predicates are those with no output variables.
– A head predicate declaration always contains exactly one variable of the input mode, e.g., `:-modeh(1, train(+train))`.

Additional settings can also be specified, or they acquire a default value. These are the maximum length of a feature (number of contained literals), maximum *variable depth* [19] and maximum number of occurrences of a given predicate symbol.

RSD produces an exhaustive set of features satisfying the mode and setting declarations. No feature produced by RSD can be decomposed into a conjunction of two features. For example, the feature set based on the following declaration

```
:-modeh(1, train(+train)).
:-modeb(2, hasCar(+train, -car)).
:-modeb(1, long(+car)).
:-modeb(1, notSame(+car, +car)).
```

will contain a feature

$$\mathtt{f(A)} : -\mathtt{hasCar(A,B)}, \mathtt{hasCar(A,C)}, \mathtt{long(B)}, \mathtt{long(C)}, \mathtt{notSame(B,C)}. \quad (1)$$

but it will not contain a feature with a body

$$\mathtt{hasCar(A,B)}, \mathtt{hasCar(A,C)}, \mathtt{long(B)}, \mathtt{long(C)} \quad (2)$$

as such an expression would clearly be decomposable into two separate features.

In the search for legal feature definitions (corresponding to the exploration of a subsumption tree), several pruning rules are used in RSD, that often drastically decrease the run times needed to achieve the feature set. For example, a simple calculation is employed to make sure that structural predicates are no longer added to a partially constructed feature definition, when there would not remain enough places (within the maximum feature length) to hold property literals consuming all output variables. A detailed treatment of the pruning principles is out the scope of this paper and will be reported elsewhere.

Extraction of constants and filtering features. The user can utilize the reserved property predicate `instantiate/1` to specify a type of variable that should be substituted with a constant during feature construction.[8] For example, consider that the result of the first step is the following feature:

$$\mathtt{f1(A) :- hasCar(A, B), hasLoad(B, C), shape(C, D), instantiate(D).} \qquad (3)$$

In the second step, after consulting the input data, `f1` will be substituted by a set of features, in each of which the `instantiate/1` literal is removed and the `D` variable is substituted by a constant, making the body of `f1` provable in the data. Provided they contain a train with a rectangle load, the following feature will appear among those created out of `f1`:

$$\mathtt{f11(A) :- hasCar(A, B), hasLoad(B, C), shape(C, rectangle).} \qquad (4)$$

A similar principle applies for features with multiple occurrences of `instantiate/1` literals. Arguments of these literals within a feature form a set of variables ϑ; only those (complete) instantiations of ϑ making the feature's body provable on the input database will be considered. Upon the user's request, the system also repeats this feature expansion process considering the negated version of each constructible feature.[9] However, not all of such features will appear in the resulting set. For the sake of efficiency, we do not perform feature filtering by a separate post-processing procedure, but rather discard certain features already during the feature construction process described above. We keep a currently developed feature f if and only if simultaneously (a) no feature has so far been generated that covers (is satisfied for) the same set of instances in the input data as f, (b) f does not cover all instances, and finally: (c) either, the fraction of instances covered by f is larger than a user-specified threshold, or the threshold coverage is reached by $\neg f$.

Creating a single-relational representation. When an appropriate set of features has been generated, RSD can use it to produce a single relational table representing the original data. Currently, the following data formats are supported: a comma-separated text file, a WEKA input file, a CN2 input file, and a file acceptable by the RSD subgroup discovery component [17].

[8] This is similar to using the `#` mode in a Progol or Aleph declaration.

[9] Note also that negations on *individual* literals can be applied via appropriate declarations in Step 1 of the feature construction.

3.2 SINUS

What follows is an overview of the SINUS approach. More detailed information about the particulars of implementation and its wide set of options can be read at the SINUS website at `http://www.cs.bris.ac.uk/home/rawles/sinus`.

LINUS. SINUS was first implemented as an intended extension to the original LINUS transformational ILP learner [13]. Work had been done in incorporating feature generation mechanisms into LINUS for structured domains and SINUS was implemented from a desire to incorporate this into a modular, transformational ILP system which integrated its propositional learner, including translating induced models back into Prolog form.

The original LINUS system had little support for the generation of features as they are discussed here. Transformation was performed by considering only possible applications of background predicates on the arguments of the target relation, taking into account the types of arguments. The clauses it could learn were constrained. The development of DINUS ('determinate LINUS') [13] relaxed the bias so that non-constrained clauses could be constructed given that the clauses involved were determinate. DINUS was also extendable to learn recursive clauses. However, not all real-world structured domains have the determinacy property, and for learning in these kinds of domains, feature generation of the sort discussed here is necessary.

SINUS. SINUS 1.0.3 is implemented in SICStus Prolog and provides an environment for transformational ILP experimentation, taking ground facts and transforming them to standalone Prolog models. The system works by performing a series of distinct and sequential steps. These steps form the functional decomposition of the system into modules, which enable a 'plug-in' approach to experimentation — the user can elect to use a number of alternative approaches for each step. For the sake of this comparison, we focus on the propositionalization step, taking into account the nature of the declarations processed before it.

- *Processing the input declarations.* SINUS takes in a set of declarations for each predicate involved in the facts and the background knowledge.
- *Constructing the types.* SINUS constructs a set of values for each type from the predicate declarations.
- *Feature generation.* The first-order features to be used as attributes in the input to the propositional learner are recursively generated.
- *Feature reduction.* The set of features generated are reduced. For example, irrelevant features may be removed, or a feature quality measure applied.
- *Propositionalization.* The propositional table of data is prepared internally.
- *File output and invocation of the propositional learner.* The necessary files are output ready for the learner to use and the user's chosen learner is invoked from inside SINUS. At present the CN2 [5] and CN2-SD (subgroup

discovery) [15] learners are supported, as well as Ripper [6]. The Weka ARFF format may also be used.
– *Transformation and output of rules.* The models induced by the propositional learner are translated back into Prolog form.

Predicate declaration and bias. SINUS uses flattened Prolog clauses together with a definition of that data. This definition takes the form of an adapted PRD file (as in the first-order Bayesian classifier 1BC [8]), which gives information about each predicate used in the facts and background information. Each predicate is listed in separate sections describing individual, structural and property predicates. From the type information, SINUS constructs the range of possible values for each type. It is therefore not necessary to specify the possible values for each type. Although this means values may appear in test data which did not appear in training data, it makes declarations much more simple and allows the easy incorporation of intensional background knowledge.

Example of a domain definition in SINUS. Revisiting the trains example, we could define the domain as follows:

```
--INDIVIDUAL
train 1 train cwa
--STRUCTURAL
train2car 2 1:train *:#car * cwa
car2load 2 1:car 1:#load * cwa
--PROPERTIES
cshape 2 car #shape * cwa
clength 2 car #length * cwa
cwall 2 car #wall * cwa
croof 2 car #roof * cwa
cwheels 2 car #wheels * cwa
lshape 2 load #shapel * cwa
lnumber 2 load #numberl * cwa
```

For each predicate, the name and number of arguments is given. Following that appears a list of the types of each argument in turn.[10] Types are defined with symbols describing their status. The `#` symbol denotes an output argument, and its absence indicates an input argument. In the structural predicates, the `1:` and `*:` prefixes allow the user to define the cardinality of the relationships. The example states that while a train has many cars, a car only has one load.

SINUS constructs features left-to-right, starting with a single literal describing the individual. For each new literal, SINUS considers the application of a structural or property predicate given the current bindings of the variables. In the case of structural predicates SINUS introduces new variable(s) for all possible type matches. In the case of property predicates SINUS substitutes all possible

[10] The remaining `* cwa` was originally for compatibility with PRD files.

constants belonging to a type of the output argument to form the new candidate literals.

The user can constrain the following factors of generated features: the maximum number of literals ($MaxL$ parameter), the maximum number of variables ($MaxV$ parameter) and the maximum number of distinct values a type can take ($MaxT$ parameter).

The character of the feature set produced by SINUS depends principally on the choice of whether and how to *reuse variables*, i.e. whether to use those variables which have already been consumed during construction of a new literal. We consider three possible cases separately.

No reuse of variables. When predicates are not allowed to reuse variables, 27 features are produced. Some features which differ from previous ones in constants only have been omitted for brevity. The full feature set contains one feature for each constant, such as[11]

```
f_aaaa(A) :- train(A),hasCar(A,B),shape(B,bucket).
f_aaaq(A) :- train(A),hasCar(A,B),hasLoad(B,C),lshape(C,circle).
f_aaax(A) :- train(A),hasCar(A,B),hasLoad(B,C),lnumber(C,0).
```

This feature set is compact but represents only simple first-order features — those which test for the existence of an object with a given property somewhere in the complex object.

Reuse of variables. When all predicates are allowed to reuse variables, the feature set increases to 283. Examples of features generated with reuse enabled include

$$\texttt{f\_aaab(A)} : -\texttt{train(A)}, \texttt{hasCar(A,B)}, \texttt{shape(B,bucket)}, \texttt{shape(B,ellipse)}. \quad (5)$$

It can be seen that generation with unrestricted variable reuse in this way is not optimal. Firstly, equivalent literals may appear in different orders and form new clauses, and features which are clearly redundant may be produced. The application of the REDUCE algorithm [16] after feature generation eliminates both these problems. Secondly, the size of the feature set increases rapidly with even slight relaxation of constraints.

Structural predicates only may reuse variables. We can allow only structural predicates to reuse variables. Using this setting, by adapting the declarations with a new property `notsame 2 car car * cwa`, taking two cars as input, we can introduce more complex features, such as the equivalent of feature `f` from Expression 1. The feature set has much fewer irrelevant and redundant features, but it can still contain decomposable features, since no explicit check is carried

[11] Note a minor difference between the feature notation formalisms in SINUS and RSD: the first listed feature, for instance, would be represented as `f1(A):-hasCar(A,B),shape(B,bucket)` in RSD.

out. However, with careful choice of the reuse option, the feature generation constraints and the background predicates, a practical and compact feature set can be generated.

3.3 Comparing RSD and SINUS

Both systems solve the propositionalization problem in a principally similar way: by first-order feature construction viewed as an exploration of the space of legal feature definitions, while the understanding of a feature is common to both systems.

Exhaustive feature generation for propositionalization is generally problematic in some domains due to the exponential increase in features size with a number of factors — number of predicates used, maximum number of literals and variables allowed, number of values possible for each type — and experience has shown there are sometimes difficulties with overfitting, as well as time and memory problems during runtime.

Both systems provide facilities to overcome this effect. SINUS allows the user to specify the bounds on the number of literals (feature length), variables and number of values taken by the types in a feature, as described above. RSD can constrain the feature length, the variable depth, number of occurrences of specified predicates and their recall number.

Both systems conduct a recursively implemented search to produce an exhaustive set of features which satisfy the user's declarations. However, unlike RSD, SINUS may produce redundant decomposable features; there is no explicit check.

Concerning the utilisation of constant values in features, SINUS collects all possible constants for each declared type from the database and proceeds to construct features using the collected constants. Unlike SINUS, RSD first generates a feature set with no constants, guided solely by the syntactical declarations. Specified variables are instantiated in a separate following step, in a way that satisfies constraints related to feature coverage on data.

Both systems are also able to generate the propositionalized form of data in a format acceptable by propositional learners including CN2 and those present in the system WEKA. SINUS furthermore provides means to translate the outputs of the propositional learner fed with the generated features back into a predicate form. To interpret such an output of a learner using RSD's features, one has to look up the meaning of each used feature in the feature definition file.

Summary. RSD puts more stress on the pre-processing stage, in that it allows a fine language declaration (such as by setting bounds on the recall of specific predicates, variable-depth etc.), verifies the undecomposability of features and offers efficiency-oriented improvements (pruning techniques in the feature search, coverage-based feature filtering). On the other hand, SINUS provides added value in the post-processing and interpretation of results obtained from a learner using the generated features, in that it is able to translate the resulting hypotheses back into a predicate form.

4 Database-Oriented Approaches

In [12], we presented a framework for approaches to propositionalization and an extension thereof by including the application of aggregation functions, which are widely used in the database area. Our approach is based on ideas from MIDOS [25], and it is called RELAGGS, which stands for *rel*ational *agg*regation*s*. It is very similar to an approach called Polka developed independently by a different research group [9]. A difference between the two approaches concerns efficiency of the implementation, which was higher for Polka. Indeed, we were inspired by Polka to develop new ideas for RELAGGS. Here, we present this new variant of our approach, implemented with Java and MySQL, with an illustrative example at the end of this section.

Besides the focus on aggregation functions, we concentrate on the exploitation of relational database schema information, especially foreign key relationships as a basis for a declarative bias during propositionalization, as well as the usage of optimization techniques as usually applied for relational databases such as indexes. These points led us to the heading for this section and do not constitute differences in principle to the logic-oriented approaches as presented above. Rather, predicate logic can be seen as fundamental to relational databases and their query languages.

In the following, we prefer to use database terminology, where a relation (table) as a collection of tuples largely corresponds to ground facts of a logical predicate, and an attribute (column) of a relation to an argument of a predicate, cf. also [14].

A relational database can be depicted as a graph with its relations as nodes and foreign key relationships as edges, conventionally by arrows pointing from the foreign key attribute in the dependent table to the corresponding primary key attribute in the independent table, cf. the examples in Fig. 1 below.

The main idea of our approach is that it is possible to summarize non-target relations with respect to the individuals dealt with, or in other words, per example from the target relation. In order to relate non-target relation tuples to the individuals, we propagate the identifiers of the individuals to the non-target tables via foreign key relationships. This can be accomplished by comparatively inexpensive joins that use indexes on primary and foreign key attributes.

In the current variant of RELAGGS, these joins – as views on the database – are materialized in order to allow for fast aggregation. Aggregation functions are applied to single columns as in [12], and to pairs of columns of single tables. The application of the functions depends on the type of attributes. For numeric attributes, *average*, *minimum*, *maximum*, and *sum* are computed as in [12], moreover standard deviations, ranges, and quartiles. For nominal attributes, the different possible values are counted, as in [12]. Here, the user can exclude nominal attributes with high numbers of possible values with the help of the parameter *cardinality*. Besides numeric and nominal attributes, we now also treat identifier attributes as ordinary numeric or nominal attributes, and date attributes as decomposable nominal attributes, e.g. for counting occurrences of

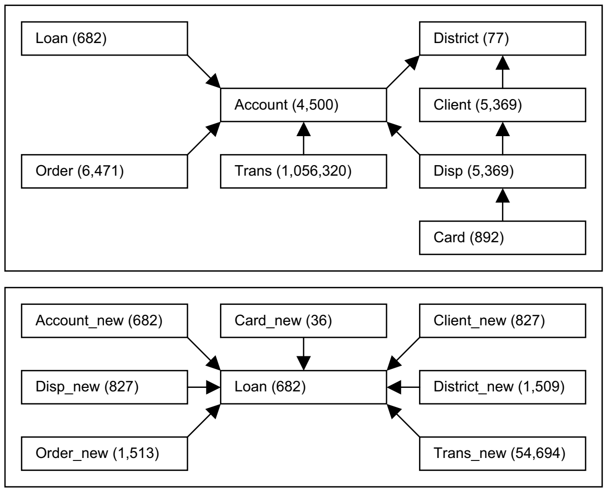

Fig. 1. Top: The PKDD 1999/2000 challenges financial data set: Relations as rectangles with relation names and tuple numbers in parentheses, arrows indicate foreign-key relationships [2]. **Bottom:** Relations after identifier propagation

a specific year. Using all these aggregation functions, most features constructed here are not Boolean as usual in logic-oriented approaches, but numeric.

Note that the RELAGGS approach can be seen as corresponding to the application of appropriate utility functions in a logic-oriented setting as pointed to in [14].

Example 1 (A PKDD data set).

Figure 1 (top) depicts parts of a relational database schema provided for the PKDD 1999/2000 challenges [2]. This data set is also used for our experiments reported on below in this paper, with table *Loan* as target relation containing the target attribute *Status*.

All relations have a single-attribute primary key of type integer with a name built from the relation name, such as *Loan_id*. Foreign key attributes are named as their primary key counterparts. Single-attribute integer keys are common and correspond to general recommendations for efficient relational database design. Here, this allows for fast propagation of example identifiers, e.g. by a statement such as *select Loan.Loan_id, Trans.* from Loan, Trans where Loan.Account_id = Trans.Account_id;* using indexes on the *Account_id* attributes. The result of this query forms the relation *Trans_new*.

Figure 1 (bottom) depicts the database following the introduction of additional foreign key attributes for propagated example identifiers in the non-target relations.

Note that relation *Trans_new*, which contains information about transactions on accounts, has become much smaller than the original relation *Trans*, mainly because there are loans for a minority of accounts only. This holds in a similar way for most other relations in this example. However, relation *District_new* has grown compared to *District*, now being the sum of accounts' district and clients' district information.

The new relations can be summarized with aggregation functions in *group by Loan_id* statements that are especially efficient here because no further joins have to be executed after identifier propagation. Finally, results of summarization such as values for a feature *min(Trans_new.Balance)* are concatenated to the central table's *Loan* tuples to form the result of propositionalization.

5 Empirical Evaluation

5.1 Learning Tasks

We chose to focus on binary classification tasks for a series of experiments to evaluate the different approaches to propositionalization described above, although the approaches can also support solutions of multi-class problems, regression problems, and even other types of learning tasks such as subgroup discovery.

As an example of the series of Trains data sets and problems as first instantiated by the East-West challenge [18], we chose a 20 trains problem, already used as an illustrating example earlier in this paper. For these trains, information is given about their cars and the loads of these cars. The learning task is to discover (low-complexity) models that classify trains as eastbound or westbound.

In the chess endgame domain White King and Rook versus Black King, taken from [20] , the target relation $illegal(A, B, C, D, E, F)$ states whether a position where the White King is at file and rank (A, B), the White Rook at (C, D) and the Black King at (E, F) is an illegal White-to-move position. For example, $illegal(g, 6, c, 7, c, 8)$ is a positive example, i.e., an illegal position. Two background predicates are available: `lt/2` expressing the "less than" relation on a pair of ranks (files), and `adj/2` denoting the adjacency relation on such pairs. The data set consists of 1,000 instances.

For the Mutagenesis problem, [23] presents a variant of the original data named NS+S2 (also known as B4) that contains information about chemical concepts relevant to a special kind of drugs, the drugs' atoms and the bonds between those atoms. The Mutagenesis learning task is to predict whether a drug is mutagenic or not. The separation of data into "regression-friendly" (188 instances) and "regression-unfriendly" (42 instances) subsets as described by [23] is kept here. Our investigations concentrate on the first subset.

The PKDD Challenges in 1999 and 2000 offered a data set from a Czech bank [2]. The data set comprises of 8 relations that describe accounts, their transactions, orders, and loans, as well as customers including personal, credit card ownership, and socio-demographic data, cf. Fig. 1. A learning task was not explicitly given for the challenges. We compare problematic to non-problematic loans regardless if the loan projects are finished or not. We exclude information from the analysis dating after loan grantings in order to arrive at models with predictive power for decision support in loan granting processes. The data describes 682 loans.

The KDD Cup 2001 [4] tasks 2 and 3 asked for the prediction of gene function and gene localization, respectively. From these non-binary classification tasks, we extracted two binary tasks, viz. the prediction whether a gene codes for a protein that serves cell growth, cell division and DNA synthesis or not and the prediction whether the protein produced by the gene described would be allocated in the nucleus or not. We deal here with the 862 training examples provided for the Cup.

5.2 Procedure

The general scheme for experiments reported here is the following. As a starting point, we take identical preparations of the data sets in Prolog form. These are adapted for usage with the different propositionalization systems, e.g. SQL scripts with create table and insert statements are derived from Prolog ground facts in a straightforward manner. Then, propositionalization is carried out and the results are formated in a way accessible to the data mining environment WEKA [24]. Here, we use the J48 learner, which is basically a reimplementation of C4.5 [21]. We use default parameter settings of this learner, including a stratified 10-fold cross-validation scheme for evaluating the learning results.

The software used for the experiments as well as SQL scripts used in the RELAGGS application are available on request from the first author. Declaration and background knowledge files used with SINUS and RSD are available from the second and third author, respectively.

Both RSD and SINUS share the same basic first-order background knowledge in all domains, adapted in formal ways for compatibility purposes. The language constraint settings applicable in either system are in principle different and for each system they were set to values allowing to complete the feature generation in a time not longer than 30 minutes.

Varying the language constraints (for RSD also the minimum feature coverage constraint; for RELAGGS: parameter *cardinality*), feature sets of different sizes were obtained, each supplied for a separate learning experiment.

5.3 Results

Accuracies Figure 2 presents for all six learning problems the predictive accuracies obtained by the J48 learner supplied with propositional data based on feature sets of growing sizes, resulting from each of the respective propositionalization systems.

Running times The three tested systems are implemented in different languages and interpreters and operate on different hardware platforms. An exact comparison of efficiency was thus not possible. For each domain and system we report the approximate average (over feature sets of different sizes) running times. RSD ran under the Yap Prolog on a Celeron 800 MHz computer with 256

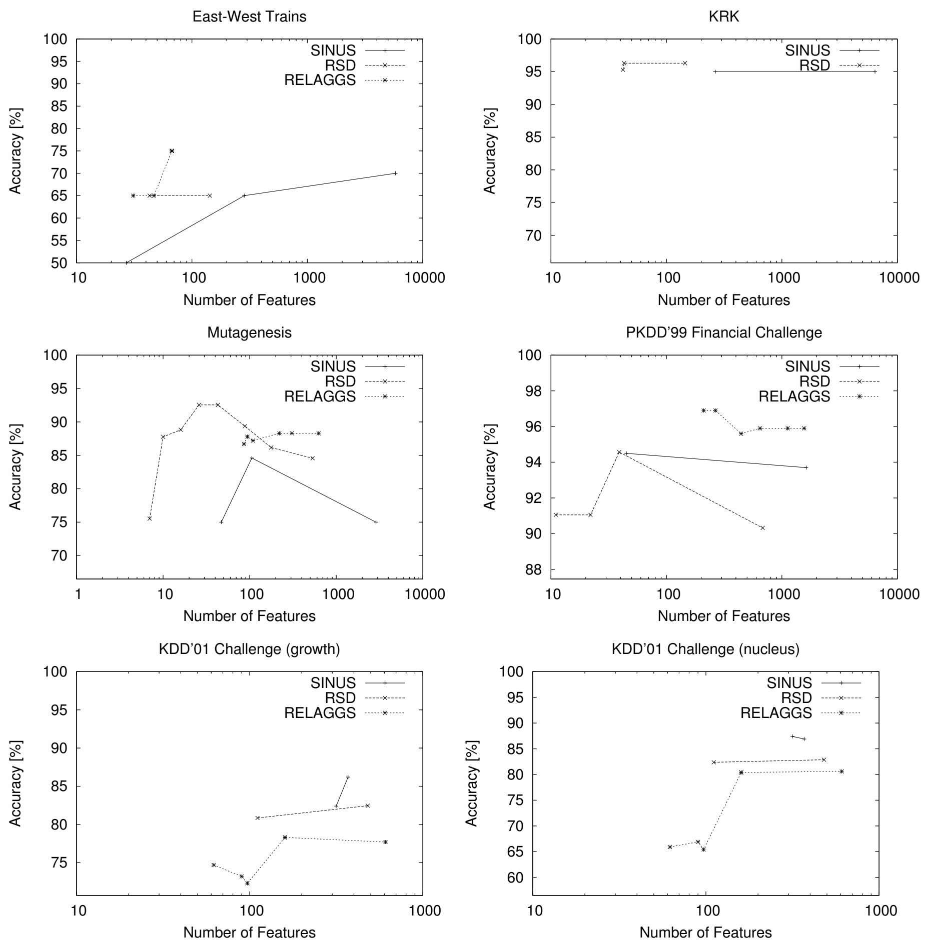

Fig. 2. Accuracies resulting from the J48 propositional learner supplied with propositionalized data based on feature sets of varying size obtained from three propositionalization systems. The bottom line of each diagram corresponds to the accuracy of the majority vote

MB of RAM. SINUS was running under SICStus Prolog[12] on a Sun Ultra 10 computer. For the Java implementation of RELAGGS, a PC platform was used with a 2.2 GHz processor and 512 MB main memory. Table 1 shows running times of the propositionalization systems on the learning tasks with best results in bold.

[12] It should be noted that SICStus Prolog is generally considered to be several times slower than Yap Prolog.

Table 1. Indicators of running times (different platforms, cf. text) and systems providing the feature set for the best-accuracy result in each domain

Problem	**Running Times**			**Best Accuracy**
	RSD	SINUS	RELAGGS	**Achieved with**
Trains	$<$ **1 sec**	2 to 10 min	$<$ **1 sec**	RELAGGS
King-Rook-King	$<$ **1 sec**	2 to 6 min	n.a.	RSD
Mutagenesis	5 min	6 to 15 min	**30 sec**	RSD
PKDD99-00 Loan.status	**5 sec**	2 to 30 min	30 sec	RELAGGS
KDD01 Gene.fctCellGrowth	3 min	30 min	**1 min**	SINUS
KDD01 Gene.locNucleus	3 min	30 min	**1 min**	SINUS

5.4 Discussion

The obtained results are not generally conclusive in favor of either of the tested systems. Interestingly, from the point of view of predictive accuracy, each of them provided the winning feature set in exactly two domains.

The strength of the aggregation approach implemented by RELAGGS manifested itself in the domain of East-West Trains (where *counting* of structural primitives seems to outperform the pure existential quantification used by the logic-based approaches) and, more importantly, in the PKDD'99 financial challenge rich with numeric data, evidently well-modelled by RELAGGS' features based on the computation of data statistics. On the other hand, this approach could not yield any reasonable results for the purely relational challenge of the King-Rook-King problem. Different performances of the two logic-based approaches, RSD and SINUS, are mainly due to their different ways of constraining the language bias. SINUS wins in both of the KDD'01 challenge versions, RSD wins in the KRK domain and Mutagenesis. While the gap on KRK seems insignificant, the result obtained on Mutagenesis with RSD's 25 features [13] is the best so far reported we are aware of with an accuracy of 92.6%.

From the point of view of running times, RELAGGS seems to be the most efficient system. It seems to be outperformed on the PKDD challenge by RSD, however, on this domain the features of both of the logic-based systems are very simple (ignoring the cumulative effects of numeric observations) and yield relatively poor accuracy results. Whether the apparent efficiency superiority of RSD with respect to SINUS is due to RSD's pruning mechanisms, or the implementation in the faster Yap Prolog, or a combined effect thereof has yet to be determined.

6 Future Work and Conclusion

In future work, we plan to complete the formal framework started in [12], which should also help to clarify relationships between the approaches. We intend to

[13] The longest have 5 literals in their bodies. Prior to irrelevant-feature filtering conducted by RSD, the feature set has more than 5,000 features.

compare our systems to other ILP approaches such as Progol [19] and Tilde [3]. Furthermore, extensions of the feature subset selection mechanisms in the different systems should be considered, as well as other propositional learners such as support vector machines.

Specifically, for RELAGGS, we will investigate a deeper integration with databases, also taking into account their dynamics. The highest future work priorities for SINUS are the implementation of a direct support for data relationships informing feature construction, incorporating a range of feature elimination mechanisms and enabling greater control over the bias used for feature construction. In RSD, we will try to devise a procedure to interpret the results of a propositional learner by a first-order theory by plugging the generated features into the obtained hypothesis.

As this paper has shown, each of the three considered systems has certain unique benefits. The common goal of all of the involved developers is to implement a wrapper that would integrate the advantages of each.

Acknowledgements

This work was partially supported by the DFG (German Science Foundation), projects FOR345/1-1TP6 and WR 40/2-1. Part of the work was done during a visit of the second author to the Czech Technical University in Prague, funded by an EC MIRACLE grant (contract ICA1-CT-2000-70002). The third author was supported by DARPA EELD Grant F30602-01-2-0571 and by the Czech Ministry of Education MSM 212300013 (Decision making and control for industrial production).

References

1. E. Alphonse and C. Rouveirol. Lazy propositionalisation for Relational Learning. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI)*, pages 256–260. IOS, 2000.
2. P. Berka. Guide to the Financial Data Set. In A. Siebes and P. Berka, editors, *PKDD2000 Discovery Challenge*, 2000.
3. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
4. J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, S. Morishita, D. Page, and J. Sese. KDD Cup 2001 Report. *SIGKDD Explorations*, 3(2):47–64, 2002.
5. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
6. W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML)*, pages 115–123. Morgan Kaufmann, 1995.
7. P. A. Flach. Knowledge representation for inductive learning. In A. Hunter and S. Parsons, editors, *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, LNAI 1638, pages 160–167. Springer, 1999.

8. P. A. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. A. Flach, editors, *Proceedings of the Ninth International Conference on Inductive Logic Programming (ILP)*, LNAI 1634, pages 92–103. Springer, 1999.
9. A. J. Knobbe, M. de Haas, and A. Siebes. Propositionalisation and Aggregates. In L. de Raedt and A. Siebes, editors, *Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Disovery (PKDD)*, LNAI 2168, pages 277–288. Springer, 2001.
10. S. Kramer and E. Frank. Bottom-up propositionalization. In *Work-in-Progress Track at the Tenth International Conference on Inductive Logic Programming (ILP)*, 2000.
11. S. Kramer, N. Lavrač, and P. A. Flach. Propositionalization Approaches to Relational Data Mining. In N. Lavrač and S. Džeroski, editors, *Relational Data Mining*, pages 262–291. Springer, 2001.
12. M.-A. Krogel and S. Wrobel. Transformation-Based Learning Using Multirelational Aggregation. In C. Rouveirol and M. Sebag, editors, *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP)*, LNAI 2157, pages 142–155. Springer, 2001.
13. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
14. N. Lavrač and P. A. Flach. An extended transformation approach to Inductive Logic Programming. *ACM Transactions on Computational Logic*, 2(4):458–494, 2001.
15. N. Lavrač, P. A. Flach, B. Kavšek, and L. Todorovski. Adapting classification rule induction to subgroup discovery. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, pages 266–273. IEEE, 2002.
16. N. Lavrač, D. Gamberger, and P. Turney. A relevancy filter for constructive induction. *IEEE Intelligent Systems*, 13(2):50–56, 1998.
17. N. Lavrač, F. Železný, and P. A. Flach. RSD: Relational subgroup discovery through first-order feature construction. In S. Matwin and C. Sammut, editors, *Proceedings of the Twelfth International Conference on Inductive Logic Programming (ILP)*, LNAI 2538, pages 149–165. Springer, 2002.
18. R. S. Michalski. Pattern Recognition as Rule-guided Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):349–361, 1980.
19. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
20. J. R. Quinlan. Learning logical definitions from relations. 5:239–266, 1990.
21. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
22. A. Srinivasan and R. D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton, editor, *Proceedings of the Sixth International Conference on Inductive Logic Programming (ILP)*, LNAI 1314, pages 89–104. Springer, 1996.
23. A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: a study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2):277–299, 1996.
24. I. H. Witten and E. Frank. *Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.
25. S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD)*, LNAI 1263, pages 78–87. Springer, 1997.