Transformation-Based Learning Using Multirelational Aggregation

Mark-A. Krogel and Stefan Wrobel

Otto-von-Guericke-Universität, Magdeburg, Germany
{krogel,wrobel}@iws.cs.uni-magdeburg.de

Abstract. Given the very widespread use of multirelational databases, ILP systems are increasingly being used on data originating from such warehouses. Unfortunately, even though not complex in structure, such business data often contain highly non-determinate components, making them difficult for ILP learners geared towards structurally complex tasks. In this paper, we build on popular transformation-based approaches to ILP and describe how they can naturally be extended with relational aggregation. We experimentall y show that this results in a multirelational learner that outperforms a structurally-oriented ILP system both in speed and accuracy on this class of problems.

1 Introduction

Relational databases and data warehouses are arguably the most widespread and commonly used technology for storing information in business, industry, and administration. The increasing popularity of data warehouses in particular has highlighted the fact that there is a large reservoir of application problems in the business area which would benefit from multirelational analysis techniques. Business databases, however, present somewhat different challenges than those found in the classical show case areas of ILP, such as molecular biology or language learning. Whereas the latter often involve highly complex structural elements, perhaps requiring deep nesting and recursion, business databases are usually structurally simple and restricted to a function-free representation. Their challenges are in two other directions. Firstly, it is quite normal that such databases are highly non-determinate — consider the case of a bank where a separate table stores thousands of transactions of a particular customer. Secondly, even though the number of involved relations may not be huge, the total size of these relations often will be, and scalability to perhaps millions of tuples is important.

Given these differing characteristics, an interesting question is whether it would not be beneficial to construct ILP learning systems that are optimised for these kinds of applications, just as many of today's state-of-the-art ILP systems are geared more towards structurally complex applications. In this paper, we show that indeed it is possible to construct a learning system that is well suited to such domains by building on approaches from the fields of ILP and the field of databases.

In particular, to ensure scalability, we have adopted a *transformation-based approach* where an ILP-problem is first transformed into a propositional problem, and then handled by a fast propositional learner. Whereas existing ILP learners based on transformation either handle only constrained or determinate clauses [12], such as LINUS [7] or DINUS [8], or use heuristically or bias-selected clauses as simple binary features [6,9], in our approach we fully treat non-determinate clauses by emploing the idea of *aggregation* from the area of databases, thus allowing non-determinate relationships to be represented in summary features of the propositional representation.

In an experimental evaluation on different learning problems arising from the multirelational data supplied by the ECML-1998 and the PKDD-1999/PKDD-2000 challenges, respectively, we show that indeed this approach outperforms both more restricted transformation-based learners, as well as a state-of-the-art ILP learning system geared more towards structurally complex domains. We compare the use of decision trees and support vector machines as propositional learners, and conclude that our approach reaches a good performance and fast runtimes with both of these.

The paper is organised as follows. In Section 2, we give an introduction to transformation-based approaches to ILP. In Section 3, we provide details of our own feature construction method, discuss its declarative bias language which is based on *foreign links*, and show how it incorporates the aggregation operator. In Section 4, we give a detailed experimental evaluation of our method, comparing it both to simpler transformation-based learners and to a state-of-the-art non-transformation-based ILP system. Section 5 provides references to related work, and Section 6 concludes and gives pointers to future work.

2 Transformation-Based Approaches

As usual in ILP, in this paper we assume that we are given a set of positive examples E^+, a set of negative examples E^-, and background knowledge B. Since we are dealing with data originating in relational databases, we will assume that E^+ is a set of ground p-atoms, i.e., atoms the predicate of which is the target predicate p (of arity a). Similarly, E^- is a set of ground negated p-atoms, and B is a set of ground atoms using different background knowledge predicates. The learning task can then be defined as follows.

– **Given:** E^+, E^-, B as described above, such that $E^+ \cup E^- \cup B \not\models \Box$ and $B \not\models E^+$
– **Find:** A hypothesis h from a set of allowed hypotheses H such that the error of h on future instances is minimised.

In ILP, h is usually a set of first-order clauses, and a new instance is classified as positive if and only if it is covered by this set of clauses. In a *transformation-based* approach to ILP, on the other hand, we assume we are given a transformation function τ which transforms the given E^+, E^-, and B into a single propositional table. One then uses a propositional learner on this table, producing a propositional hypothesis h which can then be used to classify future instances (which of course first need to be transformed by τ as well)[1].

In principle, designers of transformation-based ILP systems are not restricted to any particular form of τ functions. In practice, it is commonplace to base the transformation on an implicit first-order hypothesis space H, and use the literals and variable bindings of the clauses in H to define the transformation. For example, in the pioneering work on LINUS [7], a space of constrained clauses was used, whereas in its successor system DINUS [8], a space of determinate clauses [12] was used instead. As an alternative, if selected arbitrary clauses are used, one can apply existential transformations and use the clauses as binary features [6,9]. In order to better understand this framework, and to allow for an easier description of our own work, we will now describe this process of defining transformation functions in more detail.

2.1 Transformation Functions Based on Clauses

We will start by assuming that we are given a set $\mathcal{C}$ of clauses upon which feature generation is to be based. Note that $\mathcal{C}$ can be a systematically defined entire hypothesis space, but could also consist of a few selected clauses, so the following formalisation also covers the case of using individual clauses (perhaps learned by a non-transformation-based ILP learner) for feature generation as it is suggested in [9]. As a piece of notation, for a target predicate p of arity a, let

$$\top := p(X_1, ..., X_a) \tag{1}$$

denote the most general p-atom. Since we are considering a single predicate learning task, we can assume without loss of generality that all $C \in \mathcal{C}$ have $\top$ as head.

Let $bvars(C)$ denote the ordered set of body variables of C. For a clause C with

$$bvars(C) = \{Y_1, ..., Y_m\} \tag{2}$$

and a ground p-atom e, let

$$val(C, e) := \{(Y_1\sigma, ..., Y_m\sigma) \mid C\sigma \subseteq B \cup \{e\}\} \tag{3}$$

denote the different value combinations assumed by the body variables of C when matching the clause head against the example and the clause body against the background knowledge[2]. Note that for determinate clauses [12], $val(C, e)$ contains exactly one tuple.

[1] Depending on the transformation and the propositional learner that are used, in certain cases it is even possible to transform the propositional learning results back into an equivalent clausal theory [7,8].

[2] To simplify our notation, we are treating B as constant and do not mention it explicitly in our definitions.

We can now define a propositionalisation function φ as follows:

$$\varphi : C, e, T \mapsto (v_1, ..., v_{n_{\varphi,C}}) \ , \tag{4}$$

where $C \in \mathcal{C}$, e is a ground p-atom, and T is a set of tuples of width $\mid bvars(C) \mid$. In other words, φ produces the tuple of desired feature values for an example e with respect to the literals and variable bindings of the clause C. Sometimes, it will be useful to also have a function which generates not the individual feature *values*, but the list of *names* (and types) of the features that are the result of propositionalising based on C:

$$\Phi : C \mapsto Att_1, ..., Att_{n_{\varphi,C}} \ . \tag{5}$$

Note that since in a propositional table, all examples must have the same attributes, Φ and the width of φ must not depend on e. Also note that for both φ and Φ, we implicitly assume that the variables of each clause are typed, so φ and Φ can make use of this information when performing the propositionalisation.

Here are two simple examples of using clauses in their entirety as features. The first is the transformation used in [6,9] on selected (parts of) clauses to transform them into binary features.

Example 1 (Existential Features). This transformation simply records whether C is present in e:

$$\varphi_{\exists}(C, e, T) := \begin{cases} (1) & \text{if } \mid T \mid > 0 \ , \\ (0) & \text{otherwise.} \end{cases} \tag{6}$$

Example 2 (Counting Features). As a slight generalisation of the previous example, this function counts how often C can be matched against the example e and background knowledge B:

$$\varphi_{\#}(C, e, T) := (\mid T \mid) \ . \tag{7}$$

In order to define the complete row of features corresponding to a particular example, we simply concatenate the features generated with respect to each clause in $\mathcal{C}$ with the values of the variables in $\top$. For a p-atom $e = \top\sigma$, the propositionalisation with respect to $\mathcal{C}$ is defined as follows:

$$prop(\mathcal{C}, e) := (X_1\sigma, ..., X_a\sigma) \bigoplus_{C \in \mathcal{C}} \varphi(C, e, val(C, e)) \ , \tag{8}$$

where $\bigoplus$ denotes tuple concatenation.

Finally, the propositionalised table of examples is defined as the union of all example propositionalisations, adding in the class attribute[3]:

$$\tau(\mathcal{C}, E^+, E^-) := \{prop(\mathcal{C}, e) \oplus (1) \mid e \in E^+\} \cup \{prop(\mathcal{C}, e) \oplus (0) \mid \neg e \in E^-\} \ . \tag{9}$$

[3] Note that this definition can easily be adapted to the case where one of the arguments of $\top$ is the attribute to be predicted.

2.2 Local Functions and Redundancy

An important class of propositionalisation functions is the class of *local* propositionalisation functions which compute propositional features taking into account only one of the body variables at a time.

φ is *local* iff there is a function φ' such that

$$\varphi(C, e, T) = \bigoplus_{i=1..width(T)} \varphi'(\pi_{(i)}(T)) \ , \tag{10}$$

where $\pi_{(i)}$ denotes projection on the i-th column.

This class of propositionalisation functions is important because it easily allows the removal of redundant features whenever there are *functional dependencies* between a single predicate (or set of predicates) and another predicate.

If D is a set of atoms, L an atom, then $D \rhd L$ is a *functional dependency* iff for any σ such that

$$D\sigma \subseteq E \cup B \ , \tag{11}$$

there is *exactly* one θ such that

$$L\sigma\theta \in E \cup B \ . \tag{12}$$

Note that functional dependencies are closely related to the idea of *determinate literals* [12], except that for determinate literals, one often allows *at most* one substitution given the preceding literals, whereas a functional dependency requires that there be *exactly* one such substitution.

For local propositionalisation functions, we can drop all the features generated based on one clause if there is another clause which differs from it only in that it contains an additional functionally dependent literal. The reason for this is expressed in the following lemma.

Lemma 1. *Let C and C' two clauses from $\mathcal{C}$ such that*

$$C' = C \cup \{L\} \ . \tag{13}$$

If there is a functional dependency $D \rhd L$ such that

$$D \succeq C \tag{14}$$

(D θ-subsumes C), then for any local φ, *and any p-atom e,*

$$\varphi(C', e, val(C', e)) = \varphi(C, e, val(C, e)) \bigoplus_{z \in V_L} \varphi'(\pi_{(z)}(val(C', e)) \ , \tag{15}$$

where we assume that V_L are the variables of L not occurring in C.

Proof. Clearly, due to the functional dependency, for any variable binding tuple in $val(C, e)$ there will be exactly one completion resulting in a matching tuple in $val(C', e)$. This means that $val(C, e)$ and $val(C', e)$ are different, but since the transformation function is local, the extra columns in $val(C', e)$ do not influence the computation of the feature values on variables contained in both C and C', so the feature values computed for these variables with respect to C and C' will be identical.

This means, it suffices to consider C' when constructing $prop(\mathcal{C}, e)$ since the features constructed based on C will be redundant.

Note that this lemma can be generalised to cases where there is more than one additional functionally dependent literal, and to cases where φ also produces existential or counting features.

In our approach to be described in the next section, we assume that the functional dependencies to be used for redundancy removal are explicitly given by the user. However, it will of course also be possible to use one of the existing algorithms for functional dependency discovery to automate this step.

3 Propositionalisation by Automatic Aggregation

As pointed out in the introduction, the primary challenge in propositionalising ILP data is due to the non-determinacy of most applications. In the terminology introduced in the preceding section, this means that $val(C, e)$ can become quite a large set. This is especially true in business applications, where it is quite possible for example that a company maintains hundreds of transactions on record for a single customer. Previous approaches to propositionalisation in ILP that were restricted to determinate clauses thus cannot adequately handle such datasets.

Transformation function. In order to design our approach to transformation-based ILP learning, we have therefore borrowed the idea of *aggregation* that is commonplace in the database area [2] and often used in preprocessing for propositional learners. Aggregation is an operation that replaces a set of values by a suitable single value that summarises properties of the set. For numerical values, simple statistical descriptors such as average, maximum, and minimum can be used, for nominal values, we can use the mode (the most frequent value) or count the number of occurences of the different possible values.

More precisely, in the framework of the preceding section, we define a local propositionalisation function φ' as follows. Let C be a clause with $bvars(C) = \{Y_1, \ldots, Y_m\}$. For a numeric variable $Y_i \in bvars(C)$, let $T_i := \pi_{(i)} val(C, e)$. Then define

$$\varphi'(T_i) := (avg(T_i), max(T_i), min(T_i), sum(T_i)) \ , \tag{16}$$

where $avg(T_i)$, $max(T_i)$, $min(T_i)$, and $sum(T_i)$ compute the average, maximum, minimum, and sum of the values in T_i, respectively. For a nominal variable $Y_i \in bvars(C)$, let T_i as above. Then define

$$\varphi'(T_i) := \bigoplus_{v \in domain(Y_i)} (count(v, T_i)) \ , \tag{17}$$

where $domain(Y_i)$ is the ordered set of possible values for Y_i, and $count(v, T_i)$ is a function that provides the number of occurences of value v in T_i (again, $\bigoplus$ denotes tuple concatenation). In addition, we use the total size of the set $T := val(C, e)$ as a feature, resulting in the transformation function

$$\varphi(C, e, T) := (\mid T \mid) \bigoplus_{i=1..m} \varphi'(T_i) \ . \tag{18}$$

A function Φ was chosen to produce attribute names for the tuples resulting from propositionalisation. This function ensures unique attribute names by including the following information about the items used in the computation of the attribute values: a short name of the aggregate function applied, the name of the predicate from E or B concerned, the position/name of the argument, if applicable, and an identification of $C \in \mathcal{C}$.

Clause set. In order to decide which clause set $\mathcal{C}$ to use as the basis of our transformation, consider again the nature of business databases. Typically, they will exploit foreign key relationships to structure their data. We have therefore chosen to generate the set $\mathcal{C}$ on the basis of the *foreign link* bias language which was first introduced in MIDOS [14,15] and allows to easily model the structure of such databases. This bias is an ordered set of links $\mathcal{L}$, where each $l \in \mathcal{L}$ provides information about the argument positions of literals of two predicates where variables may be shared. As an additional level of control, our declarative bias language allows the specification of an upper limit on the number of literals with which a given literal may share variables. This limit effectively controls the branching factor in the tree of literals generated by the foreign links.

Redundancy removal. In order to exploit the potential offered by Lemma 1 for removing redundant features, we also allow the user to specify a set of functional dependencies $\mathcal{F}$.

The components discussed above result in an algorithm which is given in Table 1. Step 2 of the algorithm implements the clause construction process based on foreign links, removing redundant clauses (and thus the redundant features they would otherwise give rise to) in step 2b. Steps 3 to 5 implement the actual construction of the propositional table based on the transformation function φ defined above. Step 6 finally is a normalisation step which maps the value range of each numeric attribute to the interval $[-1, 1]$; this is used for propositional learners which benefit from normalised value ranges.

Example 3. The following examples are based on the financial dataset of the PKDD-1999/PKDD-2000 challenge that is utilised in the first half of the experiments. For illustrative purposes, the number of relations and examples is reduced here, and all but the key attributes (with primary keys always in the first argument positions) are invented.

Let the sets of positive and negative examples be
$E^+ = \{e\}$ with $e =$ `loan(1,1)`, $E^- = \emptyset$.

Let the set of background knowledge atoms be
$B = \{$ `account(1),`
`disposition(1,1,1,10,a), disposition(2,1,2,20,b),`
`client(1,1000), client(2,2000),`
`card(1,1,100), card(2,2,200), card(3,2,300)`$\}$.

Let the ordered set of foreign links (obeying the pattern:
`link(<rel1>:<pos1>, <rel2>:<pos2>`, with “rel” for relation, “pos” for argument position) be

Table 1. RELAGGS algorithm

```
1. **Accept** as input: $E$, $B$ ($n$ predicates), $\mathcal{L}$, $\mathcal{F}$
2. **Construct** $\mathcal{C}$:
   (a) **Generate** all clauses $C$ as ordered sets of literals $L$ subject to following restrictions:
       i. $\mid C \mid \leq n+1$
       ii. For $L, L' \in C : predicate(L) \neq predicate(L')$
       iii. First $L \in C : predicate(L) =$ target predicate
       iv. $L \in C$: most general, i.e. variables in argument positions
       v. $L, L' \in C$, $L$ non-first, $\exists L'$ before $L$ in $C$: $l \in \mathcal{L}$ such that $L$ shares variable with $L'$
   (b) **Eliminate** $C$ if there is $C'$ such that $C = C_1 L_1$, $C' = C L_2 C_2$, with $f \in \mathcal{F}$ specifying functional dependency between $L_1$ and $L_2$
3. **Generate** new line for $TABLE$
4. **For** all $C \in \mathcal{C}$
   (a) **Determine** $\Phi(C)$
   (b) **For** all $Att_i \in \Phi(C)$, append $Att_i$ to $TABLE$
5. **For** all $e \in E$
   (a) **Generate** new line for $TABLE$
   (b) **For** all $C \in \mathcal{C}$
       i. Determine $T = val(C, e)$
       ii. Determine $\varphi(C, e, T)$
       iii. For all $v \in \varphi(C, e, T)$ append $v$ to $TABLE$
   (c) **Append** class value of $e$ to $TABLE$
6. **Normalise** feature values of $TABLE$ to $[-1, 1]$ and append those to $TABLEnorm$
7. **Output** $TABLE$ and $TABLEnorm$
```

$\mathcal{L} = \{$ `link(loan:2,account:1),`
`link(account:1,disposition:2),`
`link(disposition:3,client:1),`
`link(disposition:1,card:2)`$\}$.

Now consider
$C_1 =$ `loan(A,B) :- account(B), disposition(C,B,D,X1,X2).`
In a first step, $val(C_1, e, T)$ is determined, which is depicted in Table 2. Here, each line corresponds to a tuple of values of $val(C_1, e, T)$. In a second step, φ and τ are applied and result in Table 3, which shows the propositionalised table of E and B with $\mathcal{C} = C_1$.

Table 2. The value set $val(C_1, e)$

$A\sigma$	$B\sigma$	$C\sigma$	$D\sigma$	$X1\sigma$	$X2\sigma$
1	1	1	1	10	a
1	1	2	2	20	b

Table 3. The propositionalised table based on C_1 $(\tau(C_1, E^+, E^-))$

count	avg(X1)	max(X1)	min(X1)	sum(X1)	count(X2=a)	count(X2=b)	class
2	15	20	10	30	1	1	1

Let C_1 as above, $C_2 =$
`loan(A,B) :- account(B), disposition(C,B,D,X1,X2), client(D,Y).`
Let us assume, the set of functional dependencies $\mathcal{F}$ contains a description of such a dependency between `disposition` and `client`, i.e.

{`disposition(_,_,D,_,_)`} $\triangleright$ `client(D,_)`.

Then, $val(C_2, e)$ produces tuples as depicted in Table 4 on the left. The result of $val(C_2, e)$ differs from $val(C_1, e)$ only in the additional column for `Y`. Especially, the columns for `X1` and `X2` are the same in both tables such that any local aggregate function applied here would not yield different results for $val(C_1, e)$ and $val(C_2, e)$. Hence, we can decide to not consider C_1.

Table 4. The value sets $val(C_2, e)$ and $val(C_3, e)$

$A\sigma$	$B\sigma$	$C\sigma$	$D\sigma$	$X1\sigma$	$X2\sigma$	$Y\sigma$
1	1	1	1	10	a	1,000
1	1	2	2	20	b	2,000

$A\sigma$	$B\sigma$	$C\sigma$	$D\sigma$	$X1\sigma$	$X2\sigma$	$E\sigma$	$Z\sigma$
1	1	1	1	10	a	1	100
1	1	2	2	20	b	2	200
1	1	2	2	20	b	2	200

Let now C_2 as above, $C_3 =$
`loan(A,B) :- account(B), disposition(C,B,D,X1,X2), card(E,Z).`
For this clause, the functional dependency given above does not apply. Table 4 shows $val(C_3, e)$, on the right. Here, there are differences with respect to the columns for `X1` and `X2` of $val(C_2, e)$ and $val(C_3, e)$. This way, there can be different aggregates as well. For example, the average of `X1` for $val(C_2, e)$ is 15, while it is 16.6 for $val(C_3, e)$. This can be viewed as weighting the property `X1` of a disposition in the light of the number of credit cards issued for this disposition. This illustrates why our algorithm will consider both C_2 and C_3 for the computation of the final propositionalised table.

4 Experimental Evaluation

The goal of our experiments was to see whether indeed the business databases that we are focusing on possess properties that distinguish them from other ILP learning problems in such a way that they lead to markedly different behaviour of state-of-the-art ILP learners (which are not optimised for such problems) compared to our own approach which we will call RELAGGS. As the basis of our experiments, we have therefore used eight learning problems originating from business domains, in particular from the discovery challenges that took place in conjunction with PKDD-1999/PKDD-2000 and ECML-1998, respectively.

The first discovery challenge problem from PKDD that we used involves learning to classify bank loans into good and bad loans [1]. The data set comprises of 8 relations and contains for each loan, customer information such as personal data, credit card ownership data, and socio-demographic data, moreover, account information is provided including permanent orders and up to several hundreds of transactions per account. This problem is thus typical of the non-determinacy that was discussed in the introduction.

The application gives rise to four different learning problems. The first two problems feature 234 examples which consist of all loans that are finished, i.e., either paid back in full or defaulted. Since certain learning systems exhibited marked differences in learning result depending on whether the good or the bad loans were used as target concept, we chose to split this task into two problems: learning using the good loans as target concept (Loan234A), and learning using the bad loans as target concept (Loan234B). Similarly, there are two further problems based on a data set of 682 examples which also contain loans that are still “in progress”, but for which a classification of current loan rating (good or bad) is available (Loan682AC uses the good loans, Loan682BD the bad loans as target concept).

The other learning problems are taken from the customer data warehouse of a large Swiss insurance company (cf. [5]), where 10 tables were extracted for the ECML challenge mentioned above. Again, information is included about the customers (called partners), plus information about their contracts, their roles in the contracts, their households etc. In this application, two unspecified learning tasks were provided with challenge data; they involve learning (different) classifications of 17,267 customers (Part1 and Part2 in the table) and 12,934 households (Hhold1 and Hhold2).

In order to evaluate the hypotheses, we chose to compare our approach RELAGGS to two other systems. First, we compared with DINUS-C, a learner based on transformations in the style of DINUS [8], i.e. using only the features that are determinate, and using C4.5rules [13] as propositional learner. Second, we compared to PROGOL [11], a very powerful state-of-the-art ILP learner capable of learning in structurally very complex domains. For PROGOL, we used standard parameter settings. We also experimented with other parameter settings, however, this did not yield better results.

The aggregates produced by RELAGGS comprise of 938 attributes for the Loan tasks, while those for the Part and Hhold tasks consist of 1,688 and 2,232 columns, respectively. In order to examine whether the success of our transformation-based approach depends on the propositional learner that is used, we used two variants of RELAGGS: RELAGGS-C uses C4.5rules [13], whereas RELAGGS-S uses SVM^{light} [3], a fast support vector machine implementation. For C4.5rules, we used standard parameter settings, while for SVM^{light}, we used normalised data and parameter settings as applied in experiments reported in [4].

All experimental evaluations were carried out as ten-fold cross-validations. Table 5 shows, for each of the participating learners, the average error across the ten folds and the standard deviation; the best result on each problem is marked in **bold**. In addition, we provide the order of magnitude of the time taken for each learning run[4].

Table 5. Error rate averages and standard deviations from 10-fold cross-validation, and runtime order of magnitude for single training runs (C – C4.5rules, S – SVM^{light})

Task	**Measurand**	**DINUS-C**	**RELAGGS-C**	**RELAGGS-S**	**PROGOL**
Loan234A	Error rate	14.9 ± 10.3	12.0 ± 6.5	**12.0 ± 5.3**	54.3 ± 10.5
	Runtime	**sec**	min	min	h
Loan234B	Error rate	14.9 ± 10.3	12.0 ± 6.5	**12.0 ± 5.3**	13.3 ± 7.1
	Runtime	**sec**	min	min	min
Loan682AC	Error rate	11.1 ± 3.6	**5.9 ± 3.2**	9.2 ± 3.2	n.a.
	Runtime	**sec**	min	min	d
Loan682BD	Error rate	11.1 ± 3.6	**5.9 ± 3.2**	9.2 ± 3.2	11.3 ± 3.6
	Runtime	**sec**	min	min	h
Part1	Error rate	18.1 ± 0.6	**8.3 ± 0.7**	9.9 ± 0.4	n.a.
	Runtime	**min**	h	h	d
Part2	Error rate	18.1 ± 0.6	**8.3 ± 0.7**	9.9 ± 0.4	n.a.
	Runtime	**min**	h	h	d
Hhold1	Error rate	39.6 ± 1.6	**6.0 ± 0.9**	14.3 ± 1.5	n.a.
	Runtime	**min**	h	h	d
Hhold2	Error rate	39.6 ± 1.6	**6.0 ± 0.9**	14.3 ± 1.5	n.a.
	Runtime	**min**	h	h	d

As can be seen from the table, the business databases used here indeed seem to possess properties that make it appear useful to use an aggregating learner like RELAGGS. On all eight problems, it is one of the RELAGGS variants which shows the lowest error. In addition, it is remarkable that RELAGGS strikes a reasonable balance in runtime between the use of determinate features only and the use of a non-transformation based learner like PROGOL. Interestingly, the complex search performed by PROGOL does not lead to good accuracies here, indicating that such business domains present quite different challenges from the problems on which PROGOL excels.

The differences between the learners are in fact statistically significant. Table 6 shows the win-loss-tie statistics, where a comparison was counted as a win if the difference was significant according to a paired t-test at level $\alpha = 0.05$. As can be seen there, the RELAGGS variant using C4.5rules significantly outperforms both RELAGGS with SVM as well as the other two learners.

[4] Five PROGOL runs were aborted after running more than two days; the corresponding fields are marked n.a.

Table 6. Win-loss-tie for rows vs. columns: differences statistically significant according to paired t-test at level $\alpha = 0.05$ (C – C4.5rules, S – SVM^{light})

	DINUS-C	PROGOL	RELAGGS-S
RELAGGS-C	6-0-2	2-0-1	6-0-2
RELAGGS-S	4-0-4	1-0-2	

5 Related Work

Our approach is based on the general idea to transform multirelational representations of data into representations amenable for efficient propositional learners, as instantiated by LINUS [7], DINUS [8], and others systems [9,6].

LINUS [7] and DINUS [8] both use a restricted class of expressions (constrained and determinate literals), whereas our approach handles arbitrary general classes. In terms of the general transformation framework introduced in this paper, systems such as LINUS and DINUS can be seen as using a suitably restricted set of classes $\mathcal{C}^*$ in order to ensure that $\{val(C, e) \mid (C \in \mathcal{C}, e \in E)\}$ is a singleton set, thus allowing the values of this single tuple to be used as features without further transformations.

Our approach is also related to [9] and [6] where general classes are allowed, but where the subsequent transformation mainly consists of checking whether such a class has instances or not (the $\varphi_\exists$ function defined above). In contrast, our approach uses a more general class of transformation functions including aggregation and counting, thus using the information in $val(C, e)$ in a more sophisticated way.

Finally, our approach of generating the clause set $\mathcal{C}$ is closely related to the declarative bias language of MIDOS [14]. Here, we supplement the set of foreign links $\mathcal{L}$ by information about functional dependencies $\mathcal{F}$ between predicates of examples E and background knowledge B. This supplement ensures efficiency by avoiding redundant features during the propositionalisation process for larger multirelational databases.

Note that our approach is open towards using different ways of generating $\mathcal{C}$, e.g. by stochastic search as suggested in [6], or based on different biases as in [9].

6 Conclusion

In this paper, we have presented RELAGGS, a transformation based ILP learner specifically geared to the challenges presented by business databases, which are often structurally quite simple, but large in size with massive amounts of non-determinacy. We have presented a general framework for such transformation-based learners which encompasses also the simpler transformations of [8,6,9], and have described how aggregation operations can naturally be used to define the transformation functions for our learner. In addition, we have shown how functional dependencies can be used to eliminate redundant features as long as the transformation function is local.

Our experimental evaluation on eight learning problems from two business domains shows that indeed such applications have special properties that make them difficult both for determinate literal based transformation learners as well as for non-transformation based learners that are geared more towards structurally complex domains. While RELAGGS beats these learners in a statistically significant way, our experiments did not show that much of a difference between variants of RELAGGS using C4.5rules and SVMs as propositional learners, with slight advantages for RELAGGS with C4.5rules.

In future work, we will evaluate RELAGGS on further domains, in particular problems with an even larger number of tuples and relations. It will be interesting to see, if the parity between C4.5rules and SVMs holds up when the number of relations and thus the number of features, is further increased. We will then also investigate whether some of the results of selecting or filtering features heuristically during propositionalisation could be of use (cf. [6,10]). Finally, we will work on the transformation function of RELAGGS, incorporating further aggregate descriptors.

Acknowledgements

This work was partially supported by the DFG (German Science Foundation), project FOR345/1-1TP6.

References

1. Petr Berka. Guide to the Financial Data Set. In A. Siebes and P. Berka, editors, *PKDD2000 Discovery Challenge*, 2000.
2. Luce Cabibbo and Riccardo Torlone. A Framework for the Investigation of Aggregate Functions in Database Queries. In C. Beeri and P. Bruneman, editors, *Proceedings of the Seventh International Conference on Database Theory (ICDT).* Springer-Verlag, 1999.
3. Thorsten Joachims. Making Large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1999.
4. Thorsten Joachims. Estimating the Generalization Performance of an SVM Efficiently. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML).* Morgan Kaufmann, 2000.
5. Jörg-Uwe Kietz, Regina Zücker, and Anca Vaduva. Mining Mart: Combining Case-Based Reasoning and Multistrategy Learning into a Framework for Reusing KDD-Applications. In R. S. Michalski and P. Brazdil, editors, *Proceedings of the Fifth International Workshop on Multistrategy Learning (MSL)*, 2000.
6. Stefan Kramer, Bernhard Pfahringer, and Christoph Helma. Stochastic Propositionalization of Non-Determinate Background Knowledge. In D. Page, editor, *Proceedings of the Eighth International Conference on Inductive Logic Programming (ILP).* Springer-Verlag, 1998.

7. Nada Lavrač. *Principles of knowledge acquisition in expert systems.* PhD thesis, University of Maribor, Ljubljana, Slovenia, 1990.
8. Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1993.
9. Nada Lavrač and Peter Flach. An extended transformation approach to Inductive Logic Programming. Technical Report CSTR-00-002, Department of Computer Science, University of Bristol, March 2000.
10. Nada Lavrač, Dragan Gamberger, and Peter Turney. A relevancy filter for constructive induction. *IEEE Intelligent Systems*, 13(2):50–56, 1998.
11. Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
12. Stephen Muggleton and Cao Feng. Efficient Induction of Logic Programs. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proceedings of the First International Workshop on Algorithmic Learning Theory (ALT).* Springer-Verlag/Ohmsha Publishers, 1990.
13. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
14. Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD).* Springer-Verlag, 1997.
15. Stefan Wrobel. Inductive Logic Progamming for Knowledge Discovery in Databases. In N. Lavrač and S. Džeroski, editors, *Relational Data Mining.* Springer-Verlag, Berlin, New York, 2000.