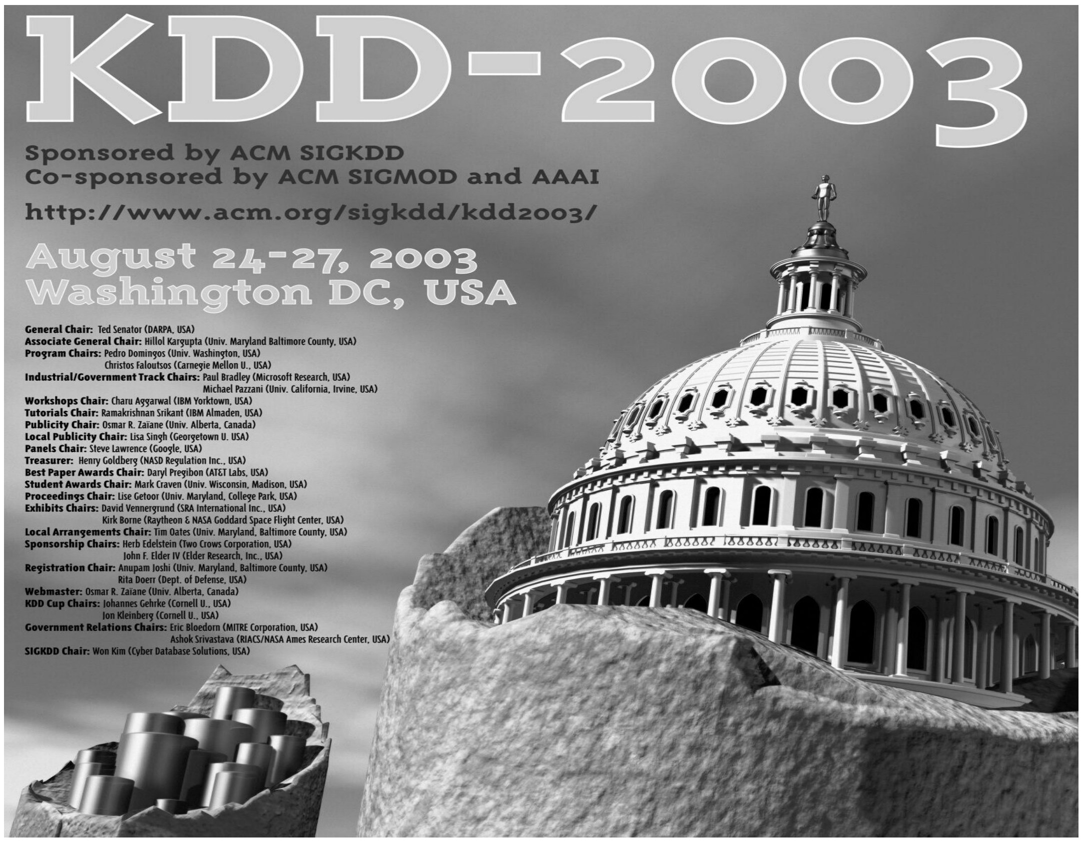
Sašo Džeroski, Luc De Raedt, and Stefan Wrobel, editors

Proceedings of the 2nd International Workshop on Multi-Relational Data Mining (MRDM-2003)

Washington DC, USA, 27 August 2003

Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)


KDD-2003
Sponsored by ACM SIGKDD
Co-sponsored by ACM SIGMOD and AAAI
http://www.acm.org/sigkdd/kdd2003/
August 24-27, 2003
Washington DC, USA
General Chair: Ted Senator (DARPA, USA)
Associate General Chair: Hillol Kargupta (Univ. Maryland Baltimore County, USA)
Program Chairs: Pedro Domingos (Univ. Washington, USA)
Christos Faloutsos (Carnegie Mellon U., USA)
Industrial/Government Track Chairs: Paul Bradley (Microsoft Research, USA)
Michael Pazzani (Univ. California, Irvine, USA)
Workshops Chair: Charu Aggarwal (IBM Yorktown, USA)
Tutorials Chair: Ramakrishnan Srikant (IBM Almaden, USA)
Publicity Chair: Osmar R. Zaïane (Univ. Alberta, Canada)
Local Publicity Chair: Lisa Singh (Georgetown U. USA)
Panels Chair: Steve Lawrence (Google, USA)
Treasurer: Henry Goldberg (NASD Regulation Inc., USA)
Best Paper Awards Chair: Daryl Pregibon (AT&T Labs, USA)
Student Awards Chair: Mark Craven (Univ. Wisconsin, Madison, USA)
Proceedings Chair: Lise Getoor (Univ. Maryland, College Park, USA)
Exhibits Chairs: David Vennergrund (SRA International Inc., USA)
Kirk Borne (Raytheon & NASA Goddard Space Flight Center, USA)
Local Arrangements Chair: Tim Oates (Univ. Maryland, Baltimore County, USA)
Sponsorship Chairs: Herb Edelstein (Two Crows Corporation, USA)
John F. Elder IV (Elder Research, Inc., USA)
Registration Chair: Anupam Joshi (Univ. Maryland, Baltimore County, USA)
Rita Doerr (Dept. of Defense, USA)
Webmaster: Osmar R. Zaïane (Univ. Alberta, Canada)
KDD Cup Chairs: Johannes Gehrke (Cornell U., USA)
Jon Kleinberg (Cornell U., USA)
Government Relations Chairs: Eric Bloedorn (MITRE Corporation, USA)
Ashok Srivastava (RIACS/NASA Ames Research Center, USA)
SIGKDD Chair: Won Kim (Cyber Database Solutions, USA)

Foreword

The 2nd International Workshop on Multi-Relational Data Mining (MRDM-2003) was held in Washington DC, USA, on 27 August 2003 as a part of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003).

Multi-Relational Data Mining (MRDM) is the multi-disciplinary field dealing with knowledge discovery from relational databases consisting of multiple tables. Mining data which consists of complex/structured objects also falls within the scope of this field, since the normalized representation of such objects in a relational database requires multiple tables. The field aims at integrating results from existing fields such as inductive logic programming, KDD, machine learning and relational databases; producing new techniques for mining multi-relational data; and practical applications of such tecniques.

Typical data mining approaches look for patterns in a single relation of a database. For many applications, squeezing data from multiple relations into a single table requires much thought and effort and can lead to loss of information. An alternative for these applications is to use multi-relational data mining. Multi-relational data mining can analyze data from a multi-relation database directly, without the need to transfer the data into a single table first. Thus the relations mined can reside in a relational or deductive database. Using multi-relational data mining it is often also possible to take into account background knowledge, which corresponds to views in the database.

Present MRDM approaches consider all of the main data mining tasks, including association analysis, classification, clustering, learning probabilistic models and regression. The pattern languages used by single-table data mining approaches for these data mining tasks have been extended to the multiple-table case. Relational pattern languages now include relational association rules, relational classification rules, relational decision trees, and probabilistic relational models, among others. MRDM algorithms have been developed to mine for patterns expressed in relational pattern languages. Typically, data mining algorithms have been upgraded from the single-table case: for example, distance-based algorithms for prediction and clustering have been upgraded by defining distance measures between examples/instances represented in relational logic.

MRDM methods have been successfully applied accross many application areas, ranging from the analysis of business data, through bioinformatics (including the analysis of complete genomes) and pharmacology (drug design) to Web mining (information extraction from text and Web sources).

The rationale behind organizing this workshop was as follows. The aim of the workshop was to bring together researchers and practitioners of data mining interested in methods for finding patterns in expressive languages from complex/ multi-relational/ structured data and their applications.

An increasing number of data mining applications involve the analysis of complex and structured types of data (such as sequences in genome analysis, HTML and XML documents) and require the use of expressive pattern languages. There is thus a clear need for multi-relational data mining (MRDM) techniques.

On the other hand, there is a wealth of recent work concerned with upgrading some recent successful data mining approaches to relational logic. A case in point are kernel methods (support-vector machines): the development of kernels for structured and richer data types is a hot research topic. Another example is the development of probabilistic relational representations and methods for learning in them (e.g., probabilistic relational models, first-order Bayesian networks, stochastic logic programs, etc.)

A non-exclusive list of topics from the call for papers, listed in alphabetical order, is as follows:

– Applications of (multi-)relational data mining
– Data mining problems that require (multi-)relational methods
– Distance-based methods for structured/relational data
– Inductive databases
– Kernel methods for structured/relational data
– Learning in probabilistic relational representations
– Link analysis and discovery
– Methods for (multi-)relational data mining
– Mining structured data, such as amino-acid sequences, chemical compounds, HTML and XML documents, ...
– Propositionalization methods for transforming (multi-)relational data mining problems to single-table data mining problems
– Relational neural networks
– Relational pattern languages

The scientific program of the workshop included an invited talk by Raghu Ramakrishan and 10 papers. Of the 15 submissions, only 10 were accepted. We wish to thank the invited speaker, all the authors who submitted their papers to MRDM-2003, the PC members for their help in the reviewing process, and the organizers of KDD-2003 for help with local organization.

Ljubljana/Freiburg/Sankt Augustin | Sašo Džeroski
Luc De Raedt
July 2003 | Stefan Wrobel

Program Chairs

Sašo Džeroski
Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
Email: `Saso.Dzeroski@ijs.si`, URL: `http://www-ai.ijs.si/SasoDzeroski/`

Luc De Raedt
Department of Computer Science, Albert-Ludwigs-Universitaet Freiburg
Georges-Köhler-Allee, Gebude 079, D-79110 Freiburg i. Br., Germany
Email: `deraedt@informatik.uni-freiburg.de`,
URL: `http://www.informatik.uni-freiburg.de/~ml/deraedt`

Stefan Wrobel
Fraunhofer Institute for Autonomous Intelligent Systems (AIS)
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
Email: `stefan.wrobel@ais.fraunhofer.de`
URL: `http://www.ais.fraunhofer.de/~wrobel`
Also professor of Computer Science at the University of Bonn, Germany

Program Committee

Table of Contents

Prolog for First-Order Bayesian Networks: A Meta-interpreter Approach

Hendrik Blockeel

Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
`hendrik.blockeel@cs.kuleuven.ac.be`

Abstract. Several extensions of bayesian belief networks to the first order logic or relational framework have been proposed. Many of these have in common that they are embedded in some kind of probabilistic or other extension of logic programming. In this paper we take yet another approach, which could be called a meta-interpreter approach. We discuss the representation of "first order" bayesian belief networks in standard Prolog. The representation formalism we propose is very simple, does not make use of any extensions to logic programming, allows inference using a simple interpreter written in Prolog, and the formalism has an expressiveness similar to other relational variants of bayesian belief networks. Due to the simplicity of the framework, we believe it may be a suitable reference point to compare other approaches to.

1 Introduction

The integration of probabilistic reasoning with first order logic is of increasing interest in the knowledge representation community (e.g., [5, 14, 11]), and also in machine learning and data mining, increasing attention is being paid to probabilistic models. A particular point of interest is formed by combinations of relational learning (including inductive logic programming) and probabilistic learning methods (e.g., [15]).

Bayesian belief networks are a popular representation formalism for representing probabilistic knowledge. The formalism has been extended to relational contexts in several ways [3, 6, 8, 2]. The relationship between all these different extensions is still under investigation.

In this paper we introduce a new approach, and a new perspective in which existing approaches can be viewed. It is a meta-interpreter approach; that is, bayesian networks are described in standard Prolog, and a meta-interpreter (also programmed in Prolog) is used to perform inference in them. This approach is quite similar to Bratko's approach of representing bayesian networks in Prolog and programming inference for them [1]. It extends it, in that a typical chunk of knowledge describes a set of bayesian networks (or: a "first order" bayesian network) rather than a single one. The networks in such a set share certain similarities, at different possible levels. Consequently, certain types of inference can happen at the level of the whole set of networks instead of its elements.

This paper presents work in progress. We discuss mainly the representation of first order bayesian networks; inference and learning are only touched upon, although it is highly unlikely that any of these would be problematic (given the rather close relationship with existing approaches where concrete algorithms have been proposed). We compare our approach with several existing approaches.

In Section 2, we give some preliminaries and motivate our approach. In Section 3 we present a proposal for representing first order bayesian belief networks in Prolog. In Section 4 we briefly discuss inference and learning. In Section 5 we compare with a number of related approaches, and in Section 6 we conclude.

2 Preliminaries and Motivation

We take a look at bayesian networks from the point of view of the knowledge they represent. A bayesian network is in fact a compact representation of a joint distribution over a set of random variables X_i, $i = 1 \ldots n$. For ease of discussion we focus on variables with finite domains D_i. The joint distribution represents all probabilities $P(X_1 = x_1, \ldots, X_n = x_n)$ with $x_i \in D_i$. There are $\prod_i |D_i|$ such probabilities, hence the representation of the joint distribution in this form is exponential in the number of variables.

In a bayesian network, additional knowledge is represented about the structure of the joint distribution. More specifically, certain independence assumptions are made; because of these assumptions, the joint distribution can be computed as a product of “local” distributions, that is, marginal and conditional distributions that describe probabilities for all combinations of a subset of variables. The size of these representations is exponential in the number of variables they contain, which is typically much lower than the total number of variables.

A bayesian network consists of a directed graph that represents information on dependencies, as follows. Each node of the graph represents a probabilistic variable. We say that x is a parent of y if there is a directed edge (x, y) in the graph. The knowledge represented by a bayesian network is that each variable is statistically independent from its non-descendants, given its parents.

For instance (example taken from Russell and Norvig [12]), we could have random variables `earthquake`, `burglary`, `alarm`, `john_calls`, `mary_calls`. The graph in Figure 1 represents the knowledge that whether the alarm goes off depends on whether there is an earthquake and also on whether there is a burglary; that burglaries and earthquakes happen independently of each other; and that, given a value for `alarm`, the reaction of the neighbours John and Mary is independent (i.e., any correlation between John's and Mary's calling is explained solely by the influence `alarm` has on them, there is no direct influence of the neighbours on each other).

As several authors have mentioned (e.g., Haddawy [4]), the knowledge expressed by this dependency graph can be expressed in propositional logic. Now in many practical situations we may have knowledge that is of a slightly more complex kind. For instance, consider the knowledge that whether a person carries some gene is influenced by whether the person's father and mother carry

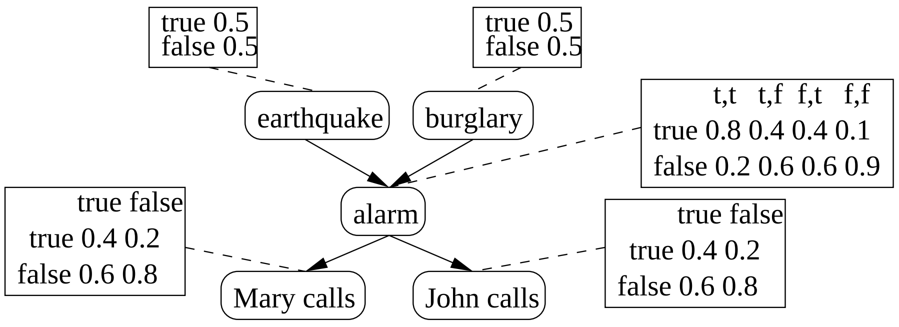


Fig. 1. A simple bayesian network.

the gene. The statement is supposed to hold for all persons and genes in a given universe, whatever that universe is. We can easily express this knowledge in first order logic. In propositional logic, we would have to know the universe in advance, and the dependency has to be stated for each person and gene combination separately; and the same holds for traditional bayesian networks. Our knowledge in this case really describes a set of bayesian networks with similar properties, rather than a single network. The question arises whether we can represent this knowledge in such a way that inference (both deductive and inductive) at the level of both the set and its elements becomes possible. Several existing approaches already demonstrate that this is possible [9, 8, 7, 13].

The difference between the approach we will propose here, and these other approaches, can be summarized as follows. Existing approaches up till now used either a procedural language to model bayesian networks, or a declarative language that is typically an extension of first order logic. Typical for the latter approaches is that the probabilistic reasoning is somehow embedded in the standard inference mechanism of the modelling language. Our approach is closest to the latter, in the sense that it is also declarative; but now, the code *describes* the network, whereas in other approaches the code *is* the network. We use a meta-interpreter to interpret the code, whereas in other approaches the interpretation is part of the inference engine underlying the program.

Our approach has the advantage that one does not need to be familiar with probabilistic extensions of first order logic, constraint logic programming, etc., nor with details of their execution mechanism. A basic understanding of standard Prolog programs is sufficient. We believe this may make the approach suitable as a reference point to compare other approaches to (in a similar way as meta-interpreters in declarative languages can be instructive regarding how inference is performed). Such reference points have been proposed previously, but currently none of them seem to be generally accepted.

3 Representing Bayesian Networks in Prolog

Bratko [1] offers an elegant method for representing standard (propositional) bayesian networks in Prolog. The approach we follow here uses a somewhat different syntax but is not essentially different; it mainly makes some parts of our discussion a bit simpler. We will use the term “first order bayesian network” to refer to a Prolog definition that represents a set of traditional bayesian networks.

We use ground terms to denote random (i.e., stochastic) variables.[1] We define a predicate *depends/3* with the following semantics. Let x be a random variable, l a list of random variables, and t a probability table. `depends(`x`,`l`,`t`)` is true if and only if in the first order bayesian network that is being modelled, x has the variables in l as parents and t is the conditional probability table that lists $P(x|l)$.

We use `input_node(`x`,`t`)` as syntactic sugar for `depends(`x`, [], `t`)`. Actually, t is in this case represented slightly differently (it is an unconditional distribution rather than a conditional one), but the actual representation of t is an implementation detail and not really relevant here.

Example 1. We again refer to Figure 1. The representation of the bayesian network in Figure 1, using the above predicates, is as follows:

```
depends(alarm, [earthquake,burglary],
 [ [  [t,t], [t,f], [f,t], [f,f]],
   [t, 0.8,   0.4,   0.4,   0.1],
   [f, 0.2,   0.6,   0.6,   0.9] ]).
depends(marycalls, [alarm],
 [ [   [t], [f]],
   [t, 0.4, 0.2],
   [f, 0.6, 0.8] ]).
depends(johncalls, [alarm],
 [ [  [t], [f]],
   [t, 0.4, 0.2],
   [f, 0.6, 0.8] ]).
input_node(earthquake, [t:0.5,f:0.5]).
input_node(burglary, [t:0.5,f:0.5]).
```

The mapping between the Prolog program and the graphical representation with probability tables is obvious. Note that no logical variables were used in the above code; this implies that the above Prolog program represents a traditional bayesian network. By introducing logical variables, more general kinds of knowledge can be represented.

Example 2. This example is inspired by the horse breeding farm example also used by Kersting and De Raedt [8]. The Prolog program

[1] This is a first important difference with other logic-based approaches, e.g., $CLP(BN)$ [13] uses skolem constants, and BLPs [8] use literals, to denote stochastic variables.

```
depends(has_gene(X),[has_gene(Y),has_gene(Z)],
    [ [   [t,t], [t,f], [f,t], [f,f]],
      [t,  0.8,   0.4,   0.4,   0.1],
      [f,  0.2,   0.6,   0.6,   0.9] ])
  :- father(Y,X), mother(Z,X).
input_node(has_gene(X), [t:0.5, f:0.5])
  :- (father(Y,X), mother(Z,X) -> fail; true).
```

represents the following information: "The probability that a horse carries a certain gene depends on its father and mother having that gene. If information on its parents is incomplete, we estimate the probability of it carrying the gene as 0.5 ".

This program can be seen as a bayesian network generator. By adding the information

```
father(apollo, chloe).
mother(blaze, chloe).
father(dexter, flash).
mother(ember, flash).
father(flash, galaxy).
mother(chloe, galaxy).
```

the network structure in Figure 2 is generated (for simplicity we write only names in the figure, instead of `has_gene(`*name*`)`).

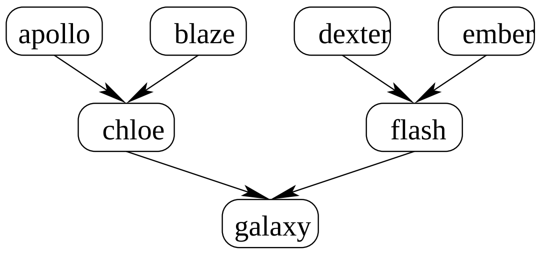


Fig. 2. A "horse farm" bayesian network.

Note that the Prolog program is more structured than the bayesian network. In a traditional bayesian network, a separate conditional probability table is associated with each node. In the Prolog program, such a table is associated with a clause, which defines a set of nodes. We just define which kind of nodes have which kind of probability table. For instance, `has_gene(chloe)` and `has_gene(flash)` share the same conditional probability table (CPT).

We could further generalize this piece of knowledge by defining the bayesian network for different genes, allowing for probability tables that vary with the type of gene but not with the horses.

```
depends(has_gene(X,G), [has_gene(Y,G),has_gene(Z,G)], T)
  :- father(Y,X), mother(Z,X), cpt(G, T).
input_node(has_gene(X,G), T)
  :- (father(Y,X), mother(Z,X) -> fail; true), distr(G, T).

cpt(gene_a, [ [   [t,t], [t,f], [f,t], [f,f]],
              [t,  0.8,   0.4,   0.4,   0.2],
              [f,  0.2,   0.6,   0.6,   0.8] ]).
distr(gene_a, [t:0.5, f:0.5]).
```

We can define several levels of similarity for the elements of a set of bayesian networks: they can all have the same structure and the same probability table; the same structure but different probability tables; or even different structures. The latter is the case, e.g., if the success of a student for some study program depends on the courses taken, and the number of such courses may be variable. Clearly in such cases the computation of the CPT becomes more complex. Nevertheless our representation formalism can easily represent such cases:

```
depends(succeeds(X), L, T) :-
  findall(Course, registered(X, Course), L),
  cpt_for_succeeds(L, T).

cpt_for_succeeds(L, T) :- ...
  % would typically depend on some weighted average
```

Obviously, not any possible definition of `depends` is consistent with its interpretation (defining a bayesian network). For instance, the definition of `depends` must be such that for each node exactly one list of parents and one CPT is generated; also, the generated directed graph must be acyclic. For now we are assuming it is the responsibility of the programmer of the bayesian network to ensure this.

4 Inference and Learning

We briefly discuss inference in first order bayesian networks and how they can be learned from data. This discussion is very brief and no concrete methods are proposed; however, our approach is sufficiently close to some other approaches (see next section) to believe that the methods proposed there can easily be adapted to our framework.

Inference in a first order bayesian network is simple, as it really represents just a large bayesian network. A Prolog meta-interpreter for inference in bayesian networks is easy to write, see, e.g., Bratko's program [1] which handles propositional networks. A simplified meta-interpreter for first order bayesian networks is provided in the appendix; although it performs a limited kind of inference only, it may be instructive regarding the simplicity of this approach.

Learning the parameters of first order bayesian networks is as easy as for propositional networks. For instance, consider the case where CPT entries are computed from counts (e.g., $P(X = x|Y = y) = |D_{X=x,Y=y}|/|D_{Y=y}|$, with D referring to some data set and $|D_c|$ denoting the number of cases fulfilling condition c. The CPT of a clause, which may describe multiple nodes in the network, can be computed by defining as "stochastic variables" not the original stochastic variables but sets of them. That is, instead of having a variable $x(a)$ and $x(b)$, for each of which we could count how many times it takes a certain value, we just have a variable x for which these counts are the sum of the separate counts for $x(a)$ and $x(b)$. In the case of "parameterized" CPT's (as for the `gene_a` example), the stochastic variables are just generalized a bit less; that is, `has_gene(apollo, gene_a)`, `has_gene(blaze, gene_a)`, etc., are generalized to a variable `has_gene(gene_a)` but not to a single variable `has_gene`.

More precisely, the procedure detailed in Figure 3 can be followed. Basically, the procedure looks at each `depends` clause separately. For each such clause, it determines the logical variables in its body that uniquely determine the CPT. Then for each possible value combination v for these variables, all observations for *Node* and *Parents* are taken into account to compute the CPT associated with v.

```
For each clause depends(Node,Parents,CPT) :- Body:
    $V$ := all logical variables in Body that determine the CPT
    $S$ := all $(Node, Parent, V)$ instantiations generated by Body
    for each different value $v$ for $V$ occurring in $S$:
        $S_v := \{(N, P, V) \in S | V = v\}$
        compute $CPT_v$ from $S_v$
```

Fig. 3. Learning the CPT's associated with a clause.

Example 3. Consider the clause

```
depends(has_gene(X,G), [has_gene(Y,G),has_gene(Z,G)], T)
  :- father(Y,X), mother(Z,X), cpt(G, T).
```

According to this clause, the CPT associated with `has_gene(X,G)` depends on G, but not on X. Therefore, a different CPT should be constructed for each different value of G that occurs (`gene_a`, ...). The CPT for `gene_a` will be constructed from all observed values of `has_gene(..., gene_a)`.

Learning the structure of a first order bayesian network boils down to learning the conditional clauses of the networks. These can be learned using ILP approaches, in exactly the same way as proposed by Kersting and De Raedt [7] and Santos Costa et al. [13].

5 Related Work

5.1 Bayesian Logic Programs

Kersting and De Raedt's [8] bayesian logic programs formed in a sense the inspiration for this work. Part of their motivation for introducing BLP's is that they generalize over bayesian networks in the same way first order logic generalizes over propositional logic. In addition, however, BLP's form a kind of unifying framework over Prolog and bayesian networks.

Instead of unifying the logical and probabilistic level, our approach explicitly aims to keep them on several levels. In our opinion this clean separation simplifies the semantics of the knowledge chunks that are represented.

A specific example that illustrates some of the complexity that arises when the logical and probabilistic levels are unified, is the following BLP clause:

```
has_gene(X) | father(Y,X), mother(Z,X), has_gene(Y), has_gene(Z)
```

Kersting and De Raedt define the bayesian networks that such a clause gives rise to, as follows: for all ground instantiations of the clause, a bayesian network exists in which the instantiated head of the clause has as parents the instantiated body literals of the clause. Note that this puts the `father` literal and the `has_gene` literal at the same level, whereas in our approach the `father` literal defines the *structure* of the bayesian network and hence is at the logical level, whereas the `has_gene` literals correspond to random variables, and hence are terms in our approach.

It is easy to see that the bayesian networks generated by BLP's do not generate the propositional bayesian networks that would normally be used if one hand-coded them. Moreover, the semantics of the resulting bayesian networks are quite complex. Consider the instantiation

```
has_gene(a) | father(b,a), mother(c,a), has_gene(b), has_gene(c)
```

and consider it as a bayesian network. The network is drawn in Figure 4, together with a similar network that would be generated by our approach. The bayesian network expresses that `has_gene(a)` is dependent on `has_gene(b)` (and its other parents). However, what is really the case is that `has_gene(a)` depends on `has_gene(b)` *only if* `father(b,a)` has the value *true*. Indeed, if `father(b,a)` has the value false, no meaningful value can be filled in for the conditional probability of `has_gene(a)` given `has_gene(b)`. That is, the conditional probability tables in this approach contain empty fields. Apparently Kersting and De Raedt's implementation handles this situation correctly by "compiling away" the variables that in fact indicate structural properties. Yet, their syntax does not hint at these complexities, and thus seems deceptively simple. In the approach we propose, the fact that `has_gene(a)` depends on `has_gene(b)` only if `father(b,a)` has the value *true*, is exactly what is represented by our Prolog clauses.

It is not obvious how inference should be performed in BLPs if the "structural" variables are not compiled away. In such a situation, the BLP approach

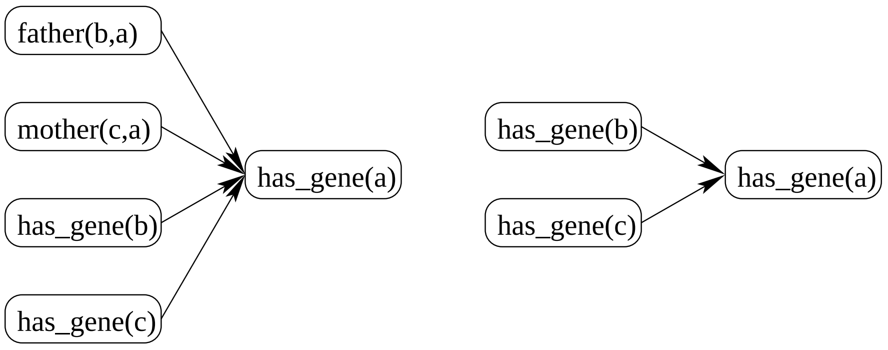


Fig. 4. Left: Bayesian network represented by the instantiation of a bayesian logic program. Right: Bayesian network generated by our approach.

can also be interpreted as follows: by placing the structural and probabilistic information at the same level, one gains the ability to learn the structure of the bayesian net together with its parameters; in fact, a bayesian net with structure s and parameters p is mapped on to a second bayesian net with fixed structure and parameters the value of which determine both s and p. This is an effect that is orthogonal to the upgrade to first order logic. In that sense, it appears BLP's extend bayesian networks in two different ways.

5.2 CLP($\mathcal{BN}$) and PRM's

The existing approach that is closest to the one we propose, is the recently proposed $CLP(\mathcal{BN})$ by Santos Costa et al. [13]. Their approach is much closer to ours than that of Kersting and De Raedt, in that they cleanly separate structural and probabilistic information, and have clauses the meaning of which is very similar to what we use. A difference is that they do not use a meta-interpreter, but use the inference engine of a constraint logic programming system to perform inference. Thus, executing their code is equivalent to performing inference, whereas in our approach, executing the code just generates the bayesian network; to perform inference the meta-interpreter has to be used. Also, Santos Costa et al. use the concept of skolem constants to model stochastic variables, while we just use ground terms. The need for the concept of skolems follows from their embedding of bayesian inference into the constraint logic inference engine. From the point of view of explaining how the approach works, this complicates matters.

We illustrate the similarity between our approach and $CLP(\mathcal{BN})$ with an example. Santos Costa et al. [13] provide the following $CLP(\mathcal{BN})$ program as an example:

```
satisfaction(Reg, Sat) :-
  reg(Reg,Course,_),
  professor(Course,Prof),
  ability(Prof,Abi),
  grade(Reg, Grade),
  sat_table(Abi, Grade, Table),
  {Sat = satisfaction(Reg) with Table}.
```

The program states that the satisfaction associated with a student-course registration (how satisfied is the student with the course?) depends on the ability of the professor and the grade the student got for the course.

This would typically be represented in our framework as follows:

```
depends(satisfaction(Reg), [ability(Prof),grade(Reg)], Table) :-
  reg(Reg, Course, _),
  professor(Course, Prof),
  sat_table(Table).
```

This illustrates that our representation typically has about the same complexity as the representation used by the $CLP(\mathcal{BN})$ approach.

Santos Costa et al. compare their approach with Probabilistic Relational Networks (PRMs) [2], showing that $CLP(\mathcal{BN})$is at least as powerful. This suggests that our meta-interpreter approach could also be used to describe PRMs.

5.3 Further Comparison with BLPs and $CLP(\mathcal{BN})$

We end this section with a small table that summarizes some of the differences between the meta-interpreter approach (MIA), BLPs, and $CLP(\mathcal{BN})$.

	BLP	$CLP(\mathcal{BN})$	MIA
stochastic var.	literal	skolem	term
dependency	|	`:-`	`depends`
prob. inference	engine	engine	meta-interpreter
queries	prob. inf.	prob. inf.	description of bayesian net

The table may clarify somewhat how the meta-interpreter approach treats bayesian nets at a meta-level; it describes a bayesian net, allowing not only standard probabilistic inference (queries about specific probabilities) but also queries regarding the structure of the bayesian net itself. Which variables depends on which other variables is explicitly described by the `depends` predicate, instead of implicitly through the structure of the program. Terms denote stochastic variables, and a meta-interpreter performs probabilistic inference, whereas in the other approaches the inference engine of the programming language takes care of this. Nevertheless, as suggested above, the meta-interpreter approach generalizes bayesian networks to the first order context as easily as the other approaches.

6 Conclusions

The contributions of this paper are as follows. We have proposed a simple formalism for representing first order bayesian networks in Prolog. We have discussed the expressiveness that this method provides; it is clear that it can at least easily handle the examples with which these other approaches are motivated. The formalism has a simple semantics : the connection between a clause and the knowledge it represents is obvious. Understanding the inference that happens in a first order bayesian network, is decoupled into understanding the semantics of a Prolog program (which defines the structure of the bayesian net) and understanding inference in traditional bayesian networks. In other approaches, both are much more intertwined (because of the embedding of the probabilistic inference in the logical inference engine). We believe that, due to its simplicity, this approach holds some promise with respect to clarifying the relationship between the many different approaches that currently exist.

This work is obviously preliminary. Further work is needed on defining the semantics of our representation formalism more formally; implementing a full meta-interpreter for bayesian inference; implementing learning procedures; and finally, comparing the approach to other approaches. This comparison should include the more procedural approaches, where some kind of programming language is used to program probabilistic models, e.g., IBAL [10].

7 Acknowledgements

The author is a post-doctoral fellow of the Fund for Scientific Research of Flanders (FWO-Vlaanderen), Belgium. He thanks Luc De Raedt, Kristian Kersting, Marc Denecker, Joost Vennekens and Sašo Džeroski for useful comments and interesting discussions.

References

1. I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 2001. 3rd Edition.
2. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 7–34. Springer-Verlag, 2001.
3. P. Haddawy. Generating Bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 262–269, 1994.
4. P. Haddawy. An overview of some recent developments in bayesian problem solving techniques. *AI Magazine*, Spring 1999.
5. J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1989.
6. M. Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 266–273. Morgan Kaufmann Publishers, 1997.
7. K. Kersting and L. De Raedt. Adaptive Bayesian logic programs. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th international conference on inductive logic programming*, pages 104–117. Springer-Verlag, 2001.

8. K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th international conference on inductive logic programming*, pages 118–131. Springer-Verlag, 2001.
9. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1316–1323, 1997.
10. A. Pfeffer. IBAL : A probabilistic rational programming language. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 733–740, 2001.
11. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.
12. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.
13. V. Santos Costa, D. Page, M. Qazi, and J. Cussens. Clp(bn): Constraint logic programming for probabilistic knowledge. In *Proceedings of 19th Conference on Uncertainty in Artificial Intelligence*, 2003. To appear. See also http://www.cos.ufrj.br/~vitor/Yap/clpbn/.
14. T. Sato and Y. Kameya. Prism: A symbolic-statistical modeling language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 1330–1335, 1997.
15. T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.

A Meta-interpreter for Inference

The following simplified interpreter performs only forward inference, in bayesian networks without undirected cycles.

```
distr(A, Distr) :- input_node(A, Distr).
distr(A, Distr) :- depends(A, B, [H|T]), sumprobs(A,B,H,T, Distr).

sumprobs(_Var, _Parents, _Values, [], []).
sumprobs(Var, Parents, Values, [[Val|Probs]|Rows], [Val:Prob|Rest]) :-
    sumprobs2(Parents, Values, Probs, Prob),
    sumprobs(Var, Parents, Values, Rows, Rest).

sumprobs2(Parents, Values, Probs,Prob) :-
    distrlist(Parents, Values, Distr),
    vecprod(Probs,Distr,Prob).


distrlist(Parents, Values, Distr) :-
  separate_distr(Parents, SepDistr),
  multiply(Values, SepDistr, Distr).
multiply([], _, []).
multiply([Combination|Rest], SepDistr, [Prob|RestDistr]) :-
  getprobs(Combination, SepDistr, Problist),
  multiply_list(Problist, 1, Prob),
  multiply(Rest, SepDistr, RestDistr).

getprobs([], [], []).
getprobs([A|B], [D1|Rest], [Prob|RestProbs]) :-
  membercheck(A:Prob, D1),
  getprobs(B, Rest, RestProbs).

multiply_list([], Acc, Acc).
multiply_list([A|B], Acc, Res) :- Acc1 is A*Acc, multiply_list(B, Acc1, Res).

vecprod([], [], 0).
vecprod([A|B], [C|D], E) :- vecprod(B,D,F), E is A*C+F.

membercheck(A, [A|_]) :- !.
membercheck(A, [_|C]) :- membercheck(A, C).

separate_distr([], []).
separate_distr([Parent|RestParents], [Distr|RestDistr]) :-
  distr(Parent, Distr),
  separate_distr(RestParents, RestDistr).
```

Towards a Formal Framework for Mining General Patterns from Ordered data *

G. Casas-Garriga

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona Salgado 1-3, Barcelona
gcasas@lsi.upc.es

Abstract. In this paper we present the initial notions of a framework for mining general patterns from complex structured data. To achieve this goal, we start by focusing here on the most basic case: the mining of a plain ordered collection of data, such as sequential databases or time-series data. Our framework develops on the hypothesis of using the *closure* of the Galois connection as a way to reduce the number of the extracted general patterns to the most representative ones. So, we employ formal concept analysis to characterize the concept lattice of these ordered contexts; our main contribution is a new Galois connection that, in this work, allows to exactly derive the closed sequential patterns found by a recent algorithm (CloSpan, [9]).

1 Introduction

Mining structured data is a complex task requiring specific techniques and formalizations different from the ones commonly applied to single normalized tables. The most basic type of structured data is an ordered context, where elements follow in an specific sequential order. These elements in the sequence can come in a simple form, such as a set of items, or also have a more complex structure, such as a hierarchical organization. The kind of patterns that can be extracted from any of these contexts can also range from a plain structure (sequential subsequences matching frequently in the data) to more complex tree-like forms (such as hybrid episodes). As a starting point for the initial foundations of our framework, we will be considering first those ordered contexts whose elements present a simple structure (i.e, sequences of sets of items that can be transactions of a sequential database or also are extended into a time-series-data form). The analysis of these ordered contexts gives rise to the sequential pattern mining problem, a relevant task in Knowledge Discovery in Databases where the identification of frequenly-arising patterns or subsequences is expected to be of interest ([2,

* This work is supported in part by EU ESPRIT IST-1999-14186 (ALCOM-FT), and MCYT TIC 2002-04019-C03-01 (MOISES)

7]). This task has many applications in different fields such as DNA sequences, proteins with similar biological functions, customer shopping sequences, intrusion detection problems, analysis of alarms in a telecomunication network and so on. One problem of the sequential pattern mining task arises when using very low support threshold in the algorithms; then, the number of extracted patterns is usually too large for a thorough examination. This problem also occurs in the task of mining association rules for large masses of unordered data ([1]): an interesting solution, called *closed itemsets* ([3, 8, 10, 11]), uses the closure of the Galois connection to reduce the total number of association rules. The foundations of closed itemsets are based on the mathematical model of concept lattices ([4, 5]), that fits perfectly well in the data mining context of an unstructured binary table. The main idea of the formalization is to present a concept-lattice framework (a lattice of closed concepts where each concept contains a closed itemset) constructed by using the Galois connection primarily introduced in [4, 5]. It can be shown that this set of closed itemsets in the lattice is necessary and sufficient to capture all the information about frequent itemsets, having smaller cardinality than the set of all frequent itemsets.

In this paper, we start by developing our framework on this hypothesis of employing the closure of a Galois connection to represent a minimum cover of all the complex structured patterns that are present in our data. As we know, there is nowadays a considerably lack of theory and formal methods for the structured case, specially in terms of representing and formalizing the idea of closed structured patterns. Thus, our main goal here is evaluate the use of formal concept analysis as a formal method to characterize, for the moment, the ordered contexts (taken as the most basic case for further complex structured data) and so, construct a lattice of closed ordered concepts by defining a new Galois connection. In this work we will show how our proposed concept lattice will serve to exactly characterize the closed patterns extracted by a recent algorithm (CloSpan, [9]), and at the same time, it will be general enough for the slightly different problem of having a single event sequence [7].

1.1 Paper Organization

We divide the work in several sections: in section 2 we present the problem formulation and the preliminary notions. Section 3 explains the previous work in [9] as the only first attempt to tackle the closure of sequences (i.e, plain ordered patterns), and motivates the need of a more formal method. The use of formal concept analysis to generally characterize the ordered context is presented in section 4; this section includes three different subsections, each one of them is a necessary step to get a well-founded lattice of closed concepts with order. Section 5 shows that it is possible to derive the closed sequential patterns in [9] from our lattice model. Finally, last section addresses the conclusions and the future work to motivate that our model can be still general enough for deriving other kind of general patterns.

2 Problem Formulation and other Necessary Definitions

Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of all items, also called the set of all **attributes** of the database. A subset of $\mathcal{I}$, i.e $I \subseteq \mathcal{I}$, is called an itemset. We are now dealing with the problem of having a collection of data $\mathcal{D} = \{s_1, s_2, \ldots s_n\}$, where each s_i is a **sequence**, also called an ordered transaction. It is important for our work that we model each element of the sequence, not as an item, but as an itemset. As we see, this problem definition fits perfectly well in the context of having a *sequential database*; in case we are dealing with *time-series data*, the long string of data can be divided in several sliding windows representing each one a piece of $\mathcal{D}$.

We consider a **sequence** an ordered list of itemsets. It can be represented as a list of pairs, $\langle (I_1, t_1)(I_2, t_2) \ldots (I_n, t_n) \rangle$, where each I_i are subsets of $\mathcal{I}$, and t_i are the order of occurrence of each itemset in the sequence (i.e, position indices). The set of all the possible sequences will be noted by $\mathcal{S}$. Without loss of generality we assume that the elements of a sequence are mapped to a set of contiguous integers and that the items in each itemset are sorted in certain order (such as alphabetic order). So, for short we will note sequences of itemsets $\langle (I_1, t_1)(I_2, t_2) \ldots (I_n, t_n) \rangle$ as $\langle (I_1)(I_2) \ldots (I_n) \rangle$. Some definitions on sequences are the following:

Definition 1. *A sequence $s = \langle (I_1) \ldots (I_n) \rangle$* **is subsequence** *of another sequence $s' = \langle (I'_1) \ldots (I'_m) \rangle$, i.e we note it by $s \subseteq s'$, if there exist integers $j_1 < j_2 \ldots < j_n$ such that $I_1 \subseteq I'_{j_1}, \ldots, I_n \subseteq I'_{j_n}$.*

When s is subsequence of another s', we also say that s **is contained** in s'. For example, the sequence $\langle (C)(D) \rangle$ is contained in $\langle (AC)(D)(B) \rangle$, so the first is a subsequence of the second. A sequence is maximal if it is not contained in any other sequence.

Definition 2. *The* **intersection** *of a collection of sequences $s_1, \ldots, s_n \in \mathcal{S}$, i.e, $s_1 \cap \ldots \cap s_n$, is the set of maximal subsequences contained into all the s_i.*

The intersection of a set of sequences can give rise to more than one sequence (note that this fact did not occur when intersecting a set of itemsets of a single table of binary data). For example, the intersection of $s = \langle (AD)(C)(B) \rangle$ and $s' = \langle (A)(B)(C) \rangle$ is the set of sequences $\{\langle (A)(C) \rangle, \langle (A)(B) \rangle\}$ since both are contained in s and s'. This important fact will determine the characterization of the subsequent lattice model, different from the lattice for unordered data.

The **support** of a sequence s in the ordered data $\mathcal{D}$ is the number of sequences in $\mathcal{D}$ that contain s. Usually, the data mining problem stated for this kind of ordered context is to find all the frequent sequential patterns (i.e, sequences whose support is over a user-specified value). However, the number of generated patterns grows quickly as the support threshold gets lower. An interesting solution is to consider the closure on sequential patterns to represent just the representative ones. We want to characterize here the closure in terms of Galois connection of all potential patterns in our data. For this purpose we are not considering in this paper any minimum support threshold for the concepts (or in other words, we consider that the threshold is set to zero, so that all possible patterns can be represented in the model); in a forthcoming work we will describe how to add support conditions to our framework.

3 Previous Related Work

An initial attempt to tackle the closure of sequences is recently performed by [9]. In that work the authors present an algorithm called CloSpan to find a kind of individual closed sequences in a sequential database. In particular, the authors define a set of **closed sequential patterns** to be found by CloSpan as:

$$CS \; = \; \{s \mid \nexists s' \; s.t \; s \subset s' \; and \; support(s) = support(s')\}$$

The way the algorithm works to find those sequences in CS is actually irrelevant for our purposes. We are trying to do a formalization of a framework for closed structured patterns (that will include in turn the notion of stability expressed by this set CS among others); so, the performance analysis of the algorithm is not important for our work, and we only focus on the definition given by [9] for closed sequential patterns. Actually, we can see that the intuitive notion of closure that can be recognized in the definition of CS is linked with the support of those sequences covering others in the database. We can see then that CS gathers a very similar semantic as the one stated for closed itemsets for simple binary data of a single normalized table. In particular, if we consider Γ to be the closure operator of an itemset closure system, the set of **closed itemsets** is defined as the following ([3, 8, 10, 11]):

$$\begin{aligned} CI = \{X \mid \Gamma(X) = X\} \equiv \{X \mid \delta(\rho(X)) = X\} \equiv \\ \{X \mid \nexists Y \; s.t \; X \subset Y \; and \; support(X) = support(Y)\} \end{aligned}$$

where Γ is the closure operator resulting from the composition of operators δ and ρ that form a Galois connection, such that, $\rho(X)$ returns all the transactions in which the input itemset X occurs in the database, and $\delta(T)$ returns the items common to all transactions T.

So, CS and CI have similar meaning of closure but in a different context. However, we consider here that the set CS does not represent all the particularities of the structured patterns that are hidden in the sequential data. For example, if we consider the database as in figure 1, then the sequence $\langle(A)\rangle$ is a closed sequential pattern and so, it belongs to CS, because there are no super-sequences having the same support; the sequence $\langle(A)(C)\rangle$ is not closed because

Seq id	Sequence
t_1	$\langle(A)(C)(B)\rangle$
t_2	$\langle(A)\rangle$
t_3	$\langle(C)(B)(A)\rangle$
t_4	$\langle(B)(C)(A)\rangle$

Fig. 1. Example of a sequential database

of $\langle(A)(C)(B)\rangle$, that has the same support and it is its supersequence. However, the simplicity of the representation of set CS does not allow to express the relation hold by sequences like $\langle(C)(A)\rangle$ and $\langle(B)(A)\rangle$: both are closed sequential patterns, and both occur in exactly the same transactions and have the same support; but, one is not the inclusion of the other.

Formally, it can exist two closed sequential patterns s and s' such that they occur in the same transactions, so $support(s) = support(s')$, but $s \not\subseteq s'$ and $s' \not\subseteq s$. This fact cannot be captured by the unidimensional representation of CS; however, it can be interesting to keep track of this relation between those different patterns in order to represent still more information. For this reason, we propose the study of the concept lattice theory on the ordered context, and derive from the new representation the set of closed patterns.

4 Concept Lattice Theory for Ordered Data

We pretend here to use formal concept analysis to create a new closure system taking into account the feature of order. We consider that the resulting concept lattice model will be a more general representation to express all the characteristics of our data. For that, we will formalize a definition of ordered context; then we define two derivation operators forming a Galois connection and, finally, we will present the formal closed concepts in a new particular lattice that will be used as a model.

4.1 Ordered Context

We start by specifying the ordered context that will serve as a basis of our lattice.

Definition 3. *An* **ordered context** $\mathbb{K} = (\mathcal{O}, \mathcal{I}, \mathcal{T}, R)$*, consists of sets* $\mathcal{O}, \mathcal{I}$ *and* $\mathcal{T}$*, and a ternary relation* R *such that* $R \subseteq \mathcal{O} \times \mathcal{I} \times \mathcal{T}$.

The elements of $\mathcal{O}$ are called the **objects** of the context (i.e, the set of original transactions), those of $\mathcal{I}$ the **attributes**, and those of $\mathcal{T}$ the **time of occurence** of the attribute. For an entry $(o, i, t) \in R$ we read as "the attribute i occurs at the time t in the object o", so, t represents the *order* of occurrence of attribute

i with respect to the other atributes in the same object. Since an attribute can occur more than once in a object, it could also have more than one order in the same object.

Like the one-valued contexts treated so far (for binary data), ordered contexts can be also represented by a cross table, the rows of which are labelled by the objects and the columns labelled by the attributes. The entry in row o and column i represents the order/s of the attribute in the object. If the attribute i does not have an order of occurrence for the object o, there will be no entry.

	A	**B**	**C**
o_1	1	3	2
o_2	1		
o_3	3	2	1
o_4	3	1	2

Fig. 2. Ordered context $\mathbb{K}$ for database in figure 1

Since each object can be interpreted as a sequence itself, the intersection of a set of objects in the context can give rise to a set of sequences, each one of them intersects with each input object. In the following subsection, we formalize the closure operator working over this proposed context.

4.2 A New Closure Operator

For our goals we define the following two new derivation operators:

Definition 4. *For a set $O \subseteq \mathcal{O}$ of objects we define,*

$$\phi(O) = \{s \in \mathcal{S} |\ s \textbf{ maximal } \textit{contained in } o,\ \forall o \in O\}$$

This $\phi(O)$ is the set of sequences common to all the objects in O, i.e, we can say that $\phi(O)$ represents the set of maximal sequences coming from the intersection of all the input objects O. Correspondingly, for a set $S \subseteq \mathcal{S}$ of sequences we can define,

$$\psi(S) = \{o \in \mathcal{O} |\ s \textit{ contained in } o,\ \forall s \in S\}$$

This $\psi(S)$ is the set of objects containing all the sequences in S.

Given that we are in an ordered context and so, the intersection of a set of objects can result in more than one sequence, we have defined for our purposes that $\phi : 2^{\mathcal{O}} \to 2^{\mathcal{S}}$. It is worth noting that the condition of maximality stated in the definition of the derivation operator ϕ is an important necessary condition for our later need of representing the notion of closure for individual sequences as we will explain. The other derivation operator is defined as $\psi : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{O}}$, since it might be that a given set of sequences S occurred always together, in the same set of objects (for this reason, the intersection of a collection of objects can give rise to a set of sequences). To be able to prove that our two derivation operators form a Galois connection, we need to specify the convenient order on the sets of sequences and on the set of objects.

Definition 5. *We say that a set of sequences S is* **more specific** *than another set of sequences S', i.e $S \preceq S'$, if and only if $\forall s \in S$, $\exists s' \in S'$ s.t. $s \subseteq s'$. Then S' is also said to be* **more general** *than S.*

According to the definition, the set of sequences $\{\langle(B)\rangle, \langle(C)\rangle, \langle(A)\rangle\}$ is more specific than the set $\{\langle(B)(A)\rangle, \langle(C)(A)\rangle\}$. Finally the two orders we consider on the sets of objects and sequences are the following: 1/ We can partially order sets of sequences by the relation $\preceq$; 2/ We can partially order sets of objects by the relation $\subseteq$ (standard inclusion). The following proposition 1 shows that the two derivation operators using the proposed orders above, form a Galois connection between the power-set lattices on $\mathcal{O}$ and on $\mathcal{S}$.

Proposition 1. *The maps $(\phi, \subseteq)$ and $(\psi, \preceq)$ form a* Galois connection.

Proof. See proof as appendix A.

Once the Galois connection is proved, we can get a new closure operator, from the composition of both derivation operators. Proposition 2 exactly states this result.

Proposition 2. *Composition of ϕ and ψ, $\Delta = \psi \cdot \phi$, or $\Delta = \phi \cdot \psi$, is a closure operator.*

According to [4] or [5], the following properties on the composition of ψ and ϕ have to be proved to show that we have a closure operator: 1/ Monotonicity: $S \preceq S' \Rightarrow \Delta(S) \preceq \Delta(S')$; 2/ Extensivity: $S \preceq \Delta(S)$; 3/ Idempotency: $\Delta(\Delta(S)) = \Delta(S)$. These conditions can be easily obtained by combining well-known properties stemming from any Galois connection. Thus, for simplicity purposes, we refer the reader to [4, 5] to prove proposition 2. Now, with the new closure operator Δ we can obtain two clousure systems: one on $\mathcal{O}$ (from $\psi \cdot \phi$) and another on $\mathcal{S}$ (from $\phi \cdot \psi$), that are dually isomorphic to each other. These two closure systems will lead to the characterization of the concept lattice as we will see in the next section.

4.3 Formal Concepts

How do we assign concepts of the lattice from an ordered context? We could first think of forgetting any new definition of the derivation operators, and try to transform the ordered context into a single normalized one-valued table, as [5] proposed to do for the multi-valued contexts. This transformation process in [5] consists of applying a "conceptual scaling" that is not at all uniquely determined. In particular, the scaling consists on expanding each original atribute with the labels of any of its possible values, in such a way that the context is translated into a binary one. So, each attribute is transformed in as many attributes as original values it has, making the intersection of objects occurring naturally when applying the closure operator. In the ordered context this scaling would not work: the new labels of each attribute would be the different positions that the attribute can take in the different sequences; however, attributes do not always occur in the same position in all the objects, which this can make difficult their intersection. For instance, a subsequence $\langle (A)(B) \rangle$ can occur in positions $\langle (A,1),(B,2)\ldots \rangle$ in one object, but in positions $\langle \ldots (A,3),(B,5)\ldots \rangle$ in another object. The intersection of these two objects should clearly give the sequence $\langle (A)(B) \rangle$; however, if we expand the labels of each attribute with their values as in the scalling, then they will never intersect since labels $A3$, $A1$ and $B5$, $B2$ will be different. We also note that the different ordinal contexts presented in [5] do not serve either to represent and express the order we want on the attributes of an object. In fact, an ordinal contex is just an specific case of multi-valued contexts where the order is specified among all the possible values of a single attribute (intra-attribute), and we would better need an inter-attribute order. For this reason, we discard the former methods in our analysis, and we present here our own definition and proposals for the formal concepts.

Definition 6. *A* **formal concept** *of the ordered context $\mathbb{K}$ is a pair (O, S) where $O \subseteq \mathcal{O}$, $S \subseteq \mathcal{S}$, and $\phi(O) = S$ and $\psi(S) = O$. Then O is the extent and S is the intent of the concept. These concepts are called* **closed** *since $S = \Delta(S)$ and $O = \Delta(O)$.*

The important difference of this formal concept with the one-valued concept treated so far is mainly in the the intent part, that consists on a *set* of *maximal* sequences contained in the objects of the extent. In other words, the intent *is not* a single sequence of ordered itemsets, as it would to be expected as a direct comparison with the concept of the one-valued context, but a set of sequences.

Extent	**Intent**
1234	$\{\ \langle (A) \rangle\ \}$
134	$\{\ \langle (A) \rangle,\ \langle (B) \rangle,\ \langle (C) \rangle\ \}$
34	$\{\ \langle (B)(A) \rangle,\ \langle (C)(A) \rangle\ \}$
13	$\{\ \langle (C)(B) \rangle,\ \langle (A) \rangle\ \}$
1	$\{\ \langle (A)(C)(B) \rangle\ \}$
3	$\{\ \langle (C)(B)(A) \rangle\ \}$
4	$\{\ \langle (B)(C)(A) \rangle\ \}$

Fig. 3. Closed concepts

So, for every set $O \subseteq \mathcal{O}$, $\phi(O)$ is the intent of some concept, since $(\psi(\phi(O)), \phi(O))$ will be always a concept. Consequently, a set $O \subseteq \mathcal{O}$ is an extent if an only if $O = \psi(\phi(O)) = \Delta(O)$. The same applies to intents. So, for the intent part, we are trying to find closed set of sequences, that is $S = \phi(\psi(S)) = \Delta(S)$ as a result of all the intersections of objects where each sequence in S occurs as a subsequence. All the sequences of events belonging to the same intent are called a **closed set of sequences**, and we say that these sequences form an **equivalence class**.

Given the context of figure 2, the set of all possible closed concepts are shown in figure 3. The intent of each concept is composed by the set of maximal sequences that result from the intersection of the set of objects in the intent (these objects in the intent are also the maximum number whose intersection generates those closed set of sequences). The extent is the set of indexes of the objects where the intent appears.

The basic properties of lattices still hold in our model: the union of extents does not result in another extent. On the other hand, the intersection of any number of extents/intents is always an extent/intent.

Proposition 3. *The intersection of intents/extents is another intent/extent.*

Proof. See proof as appendix B.

From the set of all our formal concepts we can construct now the concept lattice for the ordered context. Actually, the concept lattice representation is just a way to graphically express the partial orders on the concepts defined by:

Definition 7. *If (O_1, S_1) and (O_2, S_2) are concepts of a context, (O_1, S_1) is a* subconcept *of (O_2, S_2) if $O_2 \subseteq O_1$ (which is equivalent to $S_1 \preceq S_2$, i.e, S_2 is more general). Then (O_2, S_2) is a superconcept of (O_1, S_1) and we can write $(O_1, S_1) \leq (O_2, S_2)$.*

The set of all concepts of the context $\mathbb{K} = (\mathcal{O}, \mathcal{I}, \mathcal{T}, R)$ ordered by $\leq$ is denoted as $\underline{\mathfrak{B}}(\mathcal{O}, \mathcal{I}, \mathcal{T}, R)$ and is called the concept lattice of the context. This lattice contains just the summary of all the closed set of sequences in the context and the objects where each equivalence class is contained. In figure 4 we show the representation of the final lattice for the context of figure 2: each node is a concept, and concepts are ordered by $\leq$.

As we see, this final concept lattice can capture all the special characteristics of a sequential database in the intents of each concept: those sequences in the same equivalence class appear in the same transactions (objects) and have the same support, but they are not comparable by $\preceq$. Now we are interested in this lattice as a model to derive interesting patterns: what do the sequences in each concept mean? and, which general patterns can be derived from the set of sequences belonging to the same intent?

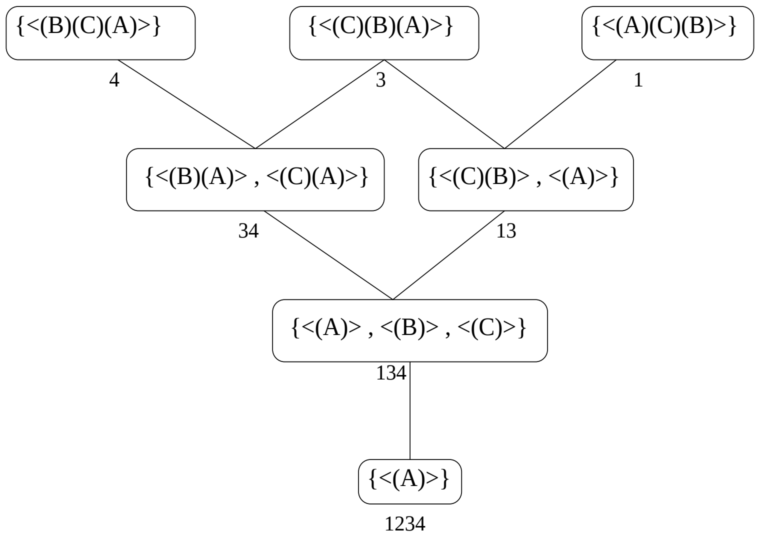


Fig. 4. Example of a concept lattice $\underline{\mathfrak{B}}(\mathcal{O}, \mathcal{I}, \mathcal{T}, R)$

5 Closed Sequential Patterns in the Lattice

Once the concept lattice is characterized, we have in each of the concepts a notion of *closed set of sequences* (the set of sequences in the same equivalence class that fullfils $S = \Delta(S)$). In other words, the closure operator Δ of the model can be only used for a set of sequences, not single individual sequences. However, we also did want in our model a notion of closure for a single sequence, like in CloSpan, in order to also characterize from our lattice the individual closed patterns that belong to set CS. It may be tempting to say that a single sequence s is closed when the set $\{s\}$ (containing only the individual s) is closed, but this definition does not work: for instance, in the sequential database of figure 1, the individual sequence $\langle(C)\rangle$ is closed according to algorithm CloSpan (since $\langle(C)\rangle \in CS$); however, if we apply the closure operator to the set $\{\langle(C)\rangle\}$ we get: $\Delta(\{\langle(C)\rangle\}) = \{\langle(A)\rangle, \langle(B)\rangle, \langle(C)\rangle\}$, so, we would conclude that is not closed because $\Delta(\{\langle(C)\rangle\}) \neq \{\langle(C)\rangle\}$. We can see then that the direct use of our closure operator that builds the lattice does not work to distinguish individual closed sequences; however, we would like to derive those sequences in CS also from our model using the closure operator. In order to achieve this goal, we first start by defining the following relation on the individual sequences s and the closure operator Δ. This relation states an important property that always holds in our lattice model and that will help in the subsequent analysis.

Proposition 4. *For any individual sequence s, it always true that $\{s\} \preceq \Delta(\{s\})$.*

Proof. See proof as appendix C.

From this last proposition, we conclude that if we try to close a single sequence s with the closure operator $\Delta(\{s\})$, we get a set of sequences where at least one of them is a supersequence of s. In other words, any s always has a "representative" in some concept. Another important and necessary property always true in our lattice model is stated by the following proposition.

Proposition 5. *For S a closed set of sequences, we have that $\nexists s, s'$ s.t $s \in S$, $s' \in S$ and $s \subseteq s'$.*

Proof. See proof as appendix C.

So, the individual sequences in the concepts of the lattice are maximal. Intuitively, this last proposition 5 can be only proved because of the maximality condition imposed on the derivation operator $\phi(O)$. In other words, the definition of $\phi(O)$ is made with the aim of avoiding that two different sequences, being one the inclusion of the other, can belong to the same closed set S. This important fact will validate our subsequent analysis. For a more detailed explanation about this refer to proof in appendix C.

The importance of these two last properties is that they will allow to formulate the subsequent characterization of closure for individual sequences. Hence, we are now interested in deriving from our lattice those sequences that are stable in support, in the same way that CloSpan produces the set CS (individual sequences covering the support of others). For our formalization purposes, we define the following notion of stability.

Definition 8. *A single sequence s is* **stable** *under in tersection if $s \in \Delta(\{s\})$*

This notion of stability gathers all those *maximal* sequences occuring in a *maximal* set of objects of the context $\mathbb{K}$. In other words, this maximal s is stable in a set of objects O when we cannot add any other object to the set O without losing this s in the intersection of all objects in O. Not all the sequences are stable (later we will prove that just the sequences of CloSpan follow this notion of stability and so, CloSpan is related to Δ). For example, sequence $\langle(B)(C)\rangle$ from the sequential database of figure 1 is not stable since $\Delta(\{\langle(B)(C)\rangle\}) = \{\langle(B)(C)(A)\rangle\}$. Now we can already propose the following lemma that exactly stablishes the location of stable sequences in the lattice.

Lemma 1. *Let S be a closed set of sequences, then $\forall s \in S$, s is an stable sequence.*

Proof. See proof as appendix C.

Lemma 1 shows us that the intents of the concepts of our lattice (that are by defition closed set of sequences), also contain the individual stable sequences, i.e, for all $s \in S$ where $\Delta(S) = S$, then s is stable.

Now that we have characterized all the stable sequences in our lattice, we are interested in defining exactly how the notion of stability can be equivalent to the sequences in CS found by CloSpan. This following theorem proves that the closed set of sequences in the lattice are exactly composed by the closed sequential patterns discovered in the set CS by CloSpan.

Theorem 1. *The set of all stable sequences is exactly the set of closed sequential patterns extracted by the algorithm CloSpan [9].*

Proof. See proof as appendix C.

From theorem 1 we reach the conclusion that the notion of closed sequences of CloSpan can be captured by each one of those sequences belonging to the equivalence classes in the concepts of our lattice, and so, the individual notion of closure is also contained in the concepts. The reason to consider an individual notion of closure interesting for our problem, is that these individual closed patterns represent a **cover in terms of support**, i.e, each stable sequence s is the maximal belonging also to the set CS (that is, all non-stable sequences s' such that $s' \subset s$ are covered by s, that has the same support). And actually, CloSpan is currently the only algorithm that mines stable sequences. It is worth noting that defining a concept lattice where the intents are single individual stable sequences (closed sequences of CS) is not possible. This is due to the derivation operators ϕ and ψ that would not form a Galois connection if we restrict their definition. The only way to create a closure system is through a set of closed sequences, where each element in the set is stable. This representation will be more general and will conceive the possibility to derive other kind of structured patterns from the concepts as we will motivate in the next section.

6 Conclusions and Future Work

In this paper we present the initial notions of a framework for mining general patterns from structured data. We develop on the hypothesis that the most representative of these general patterns are the closed ones; for that, we take here the basic ordered case and we characterize a new concept lattice in terms of Galois connection that exactly represents all the characteristics of our sequential context. The new lattice model turns to have quite useful properties and this will allow us to derive interesting closed patterns from the concepts in the nodes. In particular, we show in this work how the individual plain stable sequences in the concepts of the model exactly characterize the closed sequences found by CloSpan. This fact makes this algorithm useful to construct our lattice model as well.

There are many interesting research problems that can stem from this first work. For example, the formalization of how to derive other kind of complex patterns from this proposed lattice; in particular, patterns such as closed episodes (serial, parallel and other tree-like forms such as hybrid episodes in [7]). Intuitively, if each object in the context represents an sliding window in a long sequence of events, then, stable sequences could be transformed into closed serial episodes (that in turn would be related to CloSpan as well). Closed parallel and hybrid episodes would be derived from each one of the equivalence classes in the nodes of the lattice. Furthermore, it can be also interesting in this line of research to prove how similar are these closed patterns to the first closed episodes proposed by [6].

Finally, we expect that this model can be also extended to the processing of data organized in more complex partially ordered structures (such as hierarchical elements). We also expect that the framework could be completed by deriving knowledge from other kinds of multi-relational data always using the hypothesis of closure systems.

References

1. R.Agrawal, T. Imielinski and A.Swami. Mining Association Rules Between Sets of Items in Large Databases. *Proc. of the ACM SIGMOD Int'l Conference on Management Dada*, pages 207-216. 1993.
2. R.Agrawal and R.Srikant. Mining Sequential Patterns. *Proc. Int'l Conference on Data Engineering*, pages 3-14. 1995.
3. Y.Bastide, N.Pasquier, R.Taouil, G.Stumme and L.Lakhal. Mining Minimal Non-Redundant Association Rules using Frequent Closed Itemsets. *First Int'l Conference on Computational Logic.* 2000.
4. B.A.Davey and H.A.Priestley. *Introduction to Lattices and Order.* Cambridge, 2002.
5. B.Ganter and R.Wille. *Formal Concept Analysis. Mathematical Foundations.* Springer, 1998.
6. S.Harms, J.Deogun, J.Saquer and T.Tadesse. Discovering Representative Episodal Association Rules from Event Sequences Using Frequent Closed Episode Sets and Event Constraints. *Int'l Conference on Data Miminig.* 2001.
7. H.Mannila, H.Toivonen and I.Verkamo. Discovery of frequent episodes in event sequences. *Proc. of the Int'l Conference on Knowledge Discovery and Data Mining.* 1995.
8. N.Pasquier, Y.Bastide, R.Taouil, and L.Lakhal. Closed set based discovery of small covers for association rules. In *Proc. of the 15th. Conference on Advanced Databases*, pages 361-381. 1999.
9. X.Yan, J.Han and R.Afshar. CloSpan: Mining Closed Sequential Patterns in Large Databases. *Int'l Conference SIAM Data Mining.* 2003.
10. M.Zaki and M.Ogihara. Theoretical Foundations of Association Rules. *Third SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.* 1998.
11. M.Zaki. Generating non-redundant Association Rules.In *Proc. of the Sixth Int'l Conference on Knowledge Discovery and Data Mining*, pages 34-43. 2000.

Appendix A

Proposition 1 *The maps* $(\phi, \subseteq)$ *and* $(\psi, \preceq)$ *form a* Galois connection.

Proof. For sets of objects $O, O_1, O_2 \subseteq \mathcal{O}$, and sets of sequences $S, S_1, S_2 \subseteq \mathcal{S}$ we should prove:

1) $O_1 \subseteq O_2 \Rightarrow \phi(O_2) \preceq \phi(O_1)$	1') $S_1 \preceq S_2 \Rightarrow \psi(S_2) \subseteq \psi(S_1)$
2) $O \subseteq \psi(\phi(O))$	2') $S \preceq \phi(\psi(S))$

1) *For all $s' \in \phi(O_2)$ we have that, for all $o' \in O_2$, s' is subsequence of o', that is, in particular s' is subsequence of o for all $o \in O_1$, if $O_1 \subseteq O_2$, and thus $\exists s \in \phi(O_1)$ s.t $s' \subseteq s$, which means $\phi(O_2) \preceq \phi(O_1)$.*
2) *For all $o \in O$ we have that, s is subsequence of o for all $s \in \phi(O)$, and thus $o \in \psi(\phi(O))$.*
1') *For all $o' \in \psi(S_2)$ we have that, for all $s' \in S_2$, s' is subsequence of o', that is, in particular s is subsequence of o' for all $s \in S_1$, if $S_1 \preceq S_2$, and thus $o' \in \psi(S_1)$, which means $\psi(S_2) \subseteq \psi(S_1)$.*
2') *For all $s \in S$ we have that, s is subsequence of o for all $o \in \psi(S)$, and thus $\exists s' \in \phi(\psi(S))$ s.t $s \subseteq s'$ which implies $S \preceq \phi(\psi(S))$.*

□

Appendix B

In this appendix we show that the property on the intersection of intents/extents of the classical one-valued model, also holds for our new lattice on sets of sequences. We consider that the intersection of two set of sequences $S_1 \bigcap S_2$ is the cross intersection of single sequences $s_1 \cap s_2$ for all $s_1 \in S_1$ and all $s_2 \in S_2$.

Proposition 3 *The intersection of intents/extents is another intent/extent.*

Proof. We want to reject the following false hypothesis: from S_1 and S_2 two closed set of sequences, we suppose that $S_1 \bigcap S_2$ is not closed, i.e, it is not an intent, which can we written as:

$$S_1 \bigcap S_2 \preceq \Delta(S_1 \bigcap S_2) \quad \textit{and} \quad S_1 \bigcap S_2 \neq \Delta(S_1 \bigcap S_2)$$

In particular, this implies that we can have one of the two following situations:

1/ for some $s \in S_1 \bigcap S_2$, $\exists s' \in \Delta(S_1 \bigcap S_2)$ s.t $s \subset s'$, which implies that $s' \notin S_1 \bigcap S_2$.

2/ it exists $s' \in \Delta(S_1 \bigcap S_2)$ s.t $s' \notin S_1 \bigcap S_2$.

In any case, we get that a sequence $s' \notin S_1 \bigcap S_2$ and at the same time $s' \in \Delta(S_1 \bigcap S_2)$. But we have supposed that S_1 and S_2 are two closed set of sequences, and we have that by property of monotony of the closure operator Δ:

1/ $S_1 \bigcap S_2 \preceq S_1 \Rightarrow \Delta(S_1 \bigcap S_2) \preceq \Delta(S_1) = S_1$, and

2/ $S_1 \bigcap S_2 \preceq S_2 \Rightarrow \Delta(S_1 \bigcap S_2) \preceq \Delta(S_2) = S_2$

Then we get that $s' \in S_1$ and $s' \in S_2$, which contradicts the fact that $s' \notin S_1 \bigcap S_2$. Intersection of extents can be proved similarly. □

Appendix C

Following propositions 4 and 5 describe two important properties that hold in our lattice model.

Proposition 4 *For any given sequence s, it always true that $\{s\} \preceq \Delta(\{s\})$.*

Proof. Let $O = \psi(\{s\})$, then by definition of the derivation operator ψ we have that $\forall\ o \in O$, s is contained in o. If we now calculate the intersection of all these objects O, we get $\phi(O) = S$, where $\forall\ s' \in S$, s' is maximal contained in O.

So, $\phi(\psi(\{s\}))$ is a set of sequences s' such that each s' is maximal contained in all $o \in O$. Since we know by definition that s is also a sequence contained in all $o \in O$, then, we can conclude that at least one of these maximal $s' \in \phi(\psi(\{s\})) = \Delta(\{s\})$ is a superset of s, that is, at least exists $s' \in \Delta(\{s\})$ such that $s \subseteq s'$.

□

Proposition 5 *For S a closed set of sequences, we have that $\nexists s, s'$ s.t $s \in S$, $s' \in S$ and $s \subseteq s'$.*

Proof. If S is a closed set of sequences, then $\Delta(S) = S$ which can be rewritten as $\phi(\psi(S)) = S$, that is, $O = \psi(S)$ and $S = \phi(O)$. So, $\forall s \in S$, s is maximal contained in o, $\forall o \in O$. It means that all the sequences $s \in S$ are *maximal* by definition of our derivation operators, and it cannot exist $s' \subseteq s$ and at the same time $s' \in S$ and $s \in S$, otherwise s' would not be maximal.

□

We see with this proof that the fact that the derivation operator $\phi(O)$ is defined to return a set of *maximal* sequences common to all O is important otherwise we could have that two different sequences (but being one the inclusion of the other), can belong to the same set of closed sequences S. Now we have the tools to prove the following lemma, that characterizes the relation between stable sequences and our lattice model.

Lemma 1 *Let S be a closed set of sequences, then $\forall s \in S$, s is an stable sequence.*

Proof. Let S be the closed set of sequences of the proposition, then we have by definition that $S = \Delta(S)$, that is, $\psi(S) = O$ and $\phi(O) = S$. Now, for each one of the single sequence $s \in S$ we can find its closure as a single element of a set, i.e, we want to find the result of $\Delta(\{s\}) = \phi(\psi(\{s\}))$ to see if s is stable.

If we examine the first the part $\psi(\{s\})$, and suppose that $\psi(\{s\}) = O'$.

- If $O' = O$, then $\phi(\psi(\{s\})) = S$, and since $s \in S$ then we have that s is stable.
- If $O \subset O'$ (since it could well be that $\{s\}$ alone would be contained in more objects than S, since $\{s\} \preceq S$), then by definition of Galois connection we have that $O \subseteq O' \Rightarrow \phi(O') \preceq \phi(O)$. Thus, $\phi(O') \preceq S$, which can be rewritten as $\phi(\psi\{s\}) \preceq S$. Since $s \in S$, then we get along with proposition 4 the following relation on the sets $\{s\} \preceq \phi(\psi(\{s\})) \preceq S$; finally, by proposition 5 we know that s is *maximal* in the set S and $\nexists s' \in S$ s.t $s \subseteq s'$. So, $s \in \phi(\psi(\{s\})) = \Delta(\{s\})$, and we conclude that s is stable.
- It cannot occur that $O' \subset O$, since $\{s\} \preceq S$ and so $\{s\}$ has to be contained in at least the same objects where S is contained.

We can apply the same reasoning to all $s \in S$.

□

Next theorem serves to relate CloSpan to our lattice model.

Theorem 1 *The set of all stable sequences is exactly the set of closed sequential patterns extracted by the algorithm CloSpan [9].*

Proof.

$\Rightarrow$) If a sequence s is stable then $s \in \phi(\psi(\{s\}))$. This implies that: 1/s is a subsequence of o for all $o \in O = \psi(\{s\})$, then, $|O| = support(s)$ by definition of ψ; and, 2/ s will be *maximal* in objects O because $s \in \phi(O)$ by definiton of ϕ.

Then we can say that, for this stable s, $\nexists s'$ *s.t* $s \subseteq s'$ and s' is also contained in o $\forall o \in O$, which implies that $\nexists s'$ *s.t* $s \subseteq s'$ and $support(s') = |O|$. Since we have stated that $|O| = support(s)$, then, $\nexists s'$ *s.t* $s \subseteq s'$ and $support(s') = support(s)$. Thus, $s \in CS$.

$\Leftarrow$) Let s be a sequence belonging to set CS and let T be the set of all transactions in the database where this s appears ($|T| = support(s)$). Then, we can say that $\nexists s'$ *s.t* $s \subseteq s'$ and $support(s') = |T|$ by definition of CS, which implies that, $\nexists s'$ *s.t* $s \subseteq s'$ and s' appears in the set of transactions T; so, s is the *maximal* contained in the set of transactions T.

But we have defined a context $\mathbb{K}$ where from each transaction in T we can create an object o in O: let O be the set of objects stemming from transactions T in the original database, then, we can rewrite "s maximal contained in the transactions T" as $s \in \phi(O)$ and $O = \psi(\{s\})$, which implies that $s \in \phi(\psi(\{s\}))$; so, s is stable.

□

Towards feature selection for disk-based multirelational learners: a case study with a boosting algorithm

Susanne Hoche and Stefan Wrobel

University of Bonn, Informatik III, Römerstr. 164, 53117 Bonn, Germany, *and*
Fraunhofer AiS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany
E-Mail susanne.hoche@ais.fraunhofer.de, wrobel@ais.fraunhofer.de

Abstract. Feature selection is an important issue for any learning algorithm, since reduced feature sets lead to an improvement in learning time, reduced model complexity and, in many cases, a reduced risk of overfitting. When performing feature selection for RAM-based learning algorithms, we typically assume that the cost of accessing each feature is uniform. In multirelational data mining, especially when data are to be held in a relational database management system (RDBMS), this is no longer the case. The dominant cost in such a setting is the scan of a relation, so that the cost of using a feature from a relation that needs to be scanned anyway is comparatively small, whereas adding a feature from a relation that has not been used before is high. This means that existing work on feature selection using the uniform cost assumption may not be applicable in a disk-based setting. In this paper, we report the results of a case study that extends prior work on multirelational feature selection, in particular, in the context of a boosting algorithm. As shown by our study, using the previously developed strategies on average leads to larger numbers of relations that need to be considered and loaded into memory, and thus higher cost in a disk-based setting. Instead, a simple relation-oriented strategy can be used to minimize cost of accessing additional relations. We describe experimental results to show how this basic strategy interacts with the feature selection variants proposed previously, and show that significant gains are made even in a main-memory setting.

1 Introduction

Feature selection is an important issue for any learning algorithm, since reduced feature sets lead to an improvement in learning time, reduced model complexity and, in many cases, a reduced risk of overfitting. In multirelational learning and data mining, the problem is especially prominent, since the available features come from multiple relations that may need to be joined together before they can be used. Not surprisingly, the topic of feature selection has found significant interest recently, and a number of publications have shown that indeed feature selection is especially beneficial in multirelational learning (e.g. [14, 15, 9, 8]).

Surprisingly, however, existing work on feature selection in multirelational learning has assumed that the cost of accessing each feature is uniform. Even in a setting where all data fit in main memory, this may not be the case due to the cost of join operations. Even more, when data are to be held in a relational database management system (RDBMS), the dominant cost in accessing a feature will be the join and the scan of a relation, so that the cost of using a feature from a relation that needs to be scanned anyway is comparatively small, whereas adding a feature from a relation that has not been used before is high. This means that existing work on feature selection using the uniform cost assumption may not be applicable in a disk-based setting.

In this paper, we report the results of a case study which compares feature selection strategies in both a uniform cost setting and in the non-uniform cost setting that results from taking into account relation join and scan times. As the basis for our comparison, we are using the learning system $C2RIB^D$, [10, 9], a boosted multirelational learner capable of performing active feature selection during the learning run. In [8], we showed for the uniform cost setting that when using informed selection strategies, feature selection is indeed beneficial to the quality of learning results. In the present paper, we take two of the most successful strategies from [8], and compare them with variants that take into account from which relations the features to be added are taken, and prefer to add features from an already used relation first. Our experimental evaluation on five multirelational domains shows that even in a main memory setting, the computational complexity of the uniform cost strategies is significantly higher than what is observed for their non-uniform counterparts. As expected, the effects are most pronounced in domains with a larger number of relations of higher arity, which might be an important hint as to which data representation to choose for a multirelational problem.

The paper is organized as follows. In section 2, we briefly the recall the basics of multirelational boosting. In Section 3, we give a detailed summary of the work done on uniform cost active feature selection originally reported in [9] and [8]. In section 4, we describe the two active learning strategies chosen for this study in more detail, and introduce their relation oriented variants. Our experimental evaluation is discussed in section 5, followed by a discussion of related work and the conclusion in sections 6 and 7.

2 Constrained Confidence-Rated ILP-Boosting: C^2RIB

Boosting is a method for improving the predictive accuracy of a learning system by combining a set of base classifiers constructed by iterative calls to a base learner into one single hypothesis [6, 18, 3, 17, 10]. The idea is to "boost" a weak learner performing only slightly better than random guessing into an arbitrarily accurate learner by constructing an ensemble of base hypotheses and combining them into one final hypothesis.

To this end, a base learner is repeatedly called on reweighted versions of a set E of training instances. In each round t of boosting, a probability distribution D^t over E is maintained which models the weight D_i^t associated with each training example e_i in the t-th iteration. D_i^t indicates the influence of an instance when constructing a base classifier h_t. Initially, all instances have equal influence on the construction of a base hypothesis, i.e. the probability distribution D^1 is uniform. In each iterative call t to the base learner, a base hypothesis h_t is learned based on E weighted according to the current distribution D^t over E, and used to update the distribution for the next iteration. The weights of misclassified instances are increased while the weights of correctly classified instances are decreased, in order to focus on the examples which have not yet been correctly classified. Finally, all base hypotheses learned are combined into one final hypothesis H by a weighted majority vote of the base hypotheses.

In [10], we extended a specific approach to boosting known as constrained confidence-rated boosting, first introduced in [3], to multirelational problems. Combined with an appropriate refinement operator and search heuristics, constrained confidence-rated boosting is an effective approach to producing highly accurate multi-relational models while at the same time ensuring limited complexity of the final ensemble and acceptable induction times, as shown in [10] with the system C^2RIB (Constrained Confidence-Rated ILP Boosting).

Since for the feature selection strategies developed and evaluated in this paper, we have chosen C^2RIB as the basic algorithm and point of reference, we provide a summary of the algorithm below; for more detail, please refer to [10].

C^2RIB accepts as input the total number of iterations of the base learner, and a set $E = \{(x_1, y_1), \cdots, (x_N, y_N)\}$ of positive training examples $(x_i, 1)$ and negative training examples $(x_i, -1)$, where each x_i belongs to an instance space X. Additionally, background knowledge may be provided.

In each iterative call t of the base learner, a base hypothesis h_t is learned on E, based on the current distribution D^t. In the framework of confidence-rated boosting, the prediction of a base hypothesis h_t is confidence-rated. A prediction confidence $c(h_t, E)$ is assigned to each base hypothesis h_t. The sign of $c(h_t, E)$ indicates the label predicted by h_t to be assigned to an instance, whereas the absolute value of $c(h_t, E)$ is interpreted as the confidence in h_t's prediction. This prediction confidence is used to update D^t for the next iteration, and as h_t's vote in the final hypothesis H.

The constrained form of confidence-rated boosting which we apply here is such that the base learner is restricted to induce only hypotheses predicting the positive class with a positive prediction confidence for all examples covered by the hypothesis, and to abstain on all examples not covered by it. Additionally, the so called default hypothesis is admissible, just comprising the target predicate to be learned and satisfying all examples. The confidence assigned to the default hypothesis conforms to the sign of the class of examples with the largest sum of probabilities according to the current distribution D^t. This weighted majority class may change over the course of iterations, depending on the learned base hypotheses and thus the examples on which the learner is currently focusing.

The prediction confidence of the current base hypothesis is used to update the current probability distribution D^t such that misclassified instances will have higher weights in the next iteration of the learner. After the last iteration of the base learner, the final, strong, hypothesis is derived from all base hypotheses induced from the training instances. To classify an instance x the prediction confidences of all hypotheses h_t covering x are summed up. If this sum is positive, the strong hypothesis H classifies x as positive, otherwise x is classified as negative.

3 Uniform-Cost Feature Selection Strategies

In this section, we summarize our prior work on active uniform-cost feature selection strategies. Feature selection is a well established method to deal with the issue of efficiency issues of learning systems in the presence of large feature sets. As a standard approach, a feature selection method can be applied in an a priori fashion, and the learner can be run with a small subset of selected features. However, deciding a priori on the number of features to be included in the learning process might lead to inferior results since it is often difficult to decide just how many features to include. If too many features are included the learner is unnecessary slow, if too few features are included the learning result might not be sufficiently accurate.

In [9], we developed an approach to *actively* determine the right balance between speed and accuracy of a boosted learner based on its learning progress. We showed that, using a very simple strategy, it is possible to exploit the characteristics of boosting in order to perform active feature selection, resulting in lower induction times and reduced complexity of the ensemble. Since boosting tries to increase the certainty with which examples are classified by the ensemble of hypotheses — this is expressed in terms of the so-called *margin* —, as shown in [9], one can monitor the development of the margin in order to determine when new features or relations might be needed.

The algorithm C^2RIB^D described in [9] uses quite a simple strategy. The available features in the different relations are ordered based on an heuristic relevance measure (mutual information). Boosting is then started with a minimal set of top features, and proceeds until the development of the margins indicates that progress is slowing down at which moment the next feature on the list is added to the representation. Based on this simple approach, more advanced and informed strategies – which assumed that the cost of accessing each feature is uniform – were thoroughly investigated in [8].

In the following, we will first outline how the simple baseline active feature selection strategy is integrated into the algorithm of C^2RIB described in the preceding section (for a complete description see [9]). We will then discuss two of the most successful strategies of more informed active feature selection developed and evaluated in [8]. A concise description of the baseline algorithm C^2RIB^D is given in Table 1. References to steps in Table 1 will be indicated by "T1:_".

C^2RIB^D accepts as input, in addition to the input of C^2RIB, the set $\mathcal{F}$ of features present in the training examples, sorted in descending order according to some criterion, and a subset $\mathcal{F}'$ of the top most features of $\mathcal{F}$.

In order to actively select features depending on the requirements of the problem, and thus accelerate the learning process of the boosted learner C^2RIB

Table 1. C^2RIB^D Algorithm

```
Input:
  – The number $T$ of iterations of the base learner
  – A set of positive and negative training instances $e_i = (x_i, y_i) \in E = E^+ \cup E^-$,
    $y_i = 1$ for $e_i \in E^+$ and $y_i = -1$ for $e_i \in E^-$, with $|E| = N$
  – The set $\mathcal{F}$ of features present in the examples in $E$ sorted in descending order
    according to some criterion
  – A set $\mathcal{F}' \subseteq \mathcal{F}$ of the top features in $\mathcal{F}$

Let $D$ denote a probability distribution over $E$ with $D_i^t$ the probability of $e_i$ in the $t$-th
iteration.

1.1 Set $D_i^1 := \frac{1}{N}$ for $1 \le i \le N$
1.2 For $t = 1 \ldots T$
    (a) $C_t$ =LearnBaseHypothesis($E, D^t$)
    (b) $h_t : X \to \Re$ is the function
          $h_t(x) = \begin{cases} c(C_t, E) & \text{if e = (x, y) is covered by } C_t \\ 0 & \text{otherwise,} \end{cases}$
        where $c(C, \mathcal{S})$ is defined as $c(C, \mathcal{S}) = \frac{1}{2} \ln \left( \frac{w_+(C,\mathcal{S}) + \frac{1}{2N}}{w_-(C,\mathcal{S}) + \frac{1}{2N}} \right)$, with
        $w_+(C, \mathcal{S}) = \sum_{(x_i,1) \in \mathcal{S} \text{ covered by } C} D_i^t$, $w_-(C, \mathcal{S}) = \sum_{(x_i,-1) \in \mathcal{S} \text{ covered by } C} D_i^t$
    (c) Update the probability distribution:
        $D_i^{t'} = \frac{D_i^t}{e^{(y_i \cdot h_t(x_i))}}$, and $D_i^{t+1} = \frac{D_i^{t'}}{\sum_i D_i^{t'}}$, $1 \le i \le N$
    (d) If $t > 2$ and $\mathcal{F}' \ne \mathcal{F}$
        i.  Let $H_t := \{h_1, \cdots, h_t\}$, with base classifier $h_k$ of iteration $1 \le k \le t$
        ii. $\mathcal{F}' = CheckLearningProgress(H_t, t, E, N, \mathcal{F}, \mathcal{F}',)$ as detailed below

Output: The final hypothesis
    $H(x) := sign\left(\sum_{h_t} h_t(x)\right) = sign\left(\sum_{C_t : (x,y) \text{ covered by } C_t} c(C_t, E)\right)$
```

```
$CheckLearningProgress(H_t, t, E, N, \mathcal{F}, \mathcal{F}')$ returns $\mathcal{F}''$

2.1 Compute for $E$ the examples' average margin $AM_t = \frac{1}{N} \sum_{i=1}^{N} margin(H_t, e_i)$
2.2 Let $gradient(t)$ be the slope of the line determined by the least square fit to the
    $AM_k$ in $k, 1 \le k \le t$
2.3 Compute $trend(t) := \begin{cases} \frac{1}{T_l} \sum_{j=1}^{T_l} gradient(t-j) & \text{if } t > T_l \\ \frac{1}{t-2} \sum_{j=2}^{t-1} gradient(j) & \text{if } t \le T_l, \end{cases}$
    where $T_l$ denotes the number of iterations over which the gradients are averaged
2.4 Compute $ratio(t) := \frac{trend(t)}{gradient(t)}$
2.5 If $t > 3$:
    (a) If $ratio(t-1)$ exhibits a local maximum, estimate the slowdown in the margins'
        improvement in the form of $predict(x) := a \frac{1}{ln(\frac{x}{b})}$, where $a, b$ are chosen such
        that $predict(2) = ratio(t-1)$ and $predict(3) = ratio(t)$; $offset := t - 3$
    (b) If $a, b$ have already been determined, compute $predict(t) := a \frac{1}{ln(\frac{t-offset}{b})}$
    (c) Else $predict(t) := ratio(t)$
    (d) If $\frac{predict(t)}{ratio(t)} > \alpha$, select the first element $F$ of $\mathcal{F}$, i.e. the feature with the next
        greatest mutual information with the training examples' class; $\mathcal{F}'' := \mathcal{F}' \cup \{F\}$
    (e) Else $\mathcal{F}'' := \mathcal{F}'$
```

without a deterioration of its prediction accuracy, we start the learner with the features in $\mathcal{F}'$ and the relations in which these features occur, monitor the learning progress and include additional features and relations into the learning process only by demand (T1:1.2d, T1:2._).

The learning progress is monitored in terms of the development of the training examples' mean margins. The margin of an example $e_i = (x_i, y_i)$ under an ensemble H_t of base classifiers $h_1, h_2, \cdots, h_t$ is a real-valued number $margin(H_t, e_i) \in [-1, 1]$ indicating the amount of disagreement of the classifiers in H_t with respect to e_i's class. For the binary case we deal with here, we can define the margin of e_i under H_t as the difference between the sum of the absolute weights of those base classifiers in H_t predicting for e_i its correct class y_i, and the sum of the absolute weights of those base classifiers in H_t predicting for e_i the incorrect class $y \neq y_i$ [9].

Large positive margins indicate a “confident” correct classification. The more negative a margin is, the more confident an incorrect classification is indicated. Boosting is known to be especially effective at increasing the margins of the training examples [20, 7]. By increasing their probabilities, boosting forces the focus on misclassified instances which show small or even negative margins. The learner is forced to search for base hypotheses which correctly classify these hard examples and thus increase their margins. Since the margins are increasing in the course of iterated calls to the base learner, the gradient of the mean margins can be assumed to be positive and be employed to monitor the quality of the learning process.

For monitoring the learning success, we define in each iteration t of boosting the gradient $gradient(t)$ of t as the slope of the line determined by the least square fit to the average margins in each single iteration 1 to t (T1:2.1, T1:2.2). We then average the gradients over the last T_l iterations as to smooth temporary fluctuations in the margins' development (T1:2.3), and compute the ratio of the averaged previous gradients and the current gradient (T1:2.4).

The margins' improvement is measured by this ratio which increases from one iteration to the next as long as the margins increase significantly. As soon as the ratio starts to decrease, an estimate for the slowdown in the margins' improvements is determined (T1:2.5a). This estimate predicts the expected decrease of the ratio (T1:2.5b, T1:2.5c) and is used to determine when a new feature has to be presented to the learner. Whenever the actual decrease of the ratio exceeds the predicted decrease by a certain threshold, a new feature is included into the learning process (T1:2.5d).

In [8], we showed that more complex strategies of feature ordering and selection can further improve the learning results. Instead of relying on the sequence that had been initially determined on the entire training set based on the features' mutual information with the class, a new feature order was established, by reweighting features based on the current distribution over the training examples, every time a feature was requested by the base learner. Such a new feature order can be arrived at by considering:

– the entire training set for feature reweighting, or only the fraction of examples which are misclassified by the current ensemble of base hypotheses;
– the entire original feature set or just the features which have not been presented to the learner yet;
– not only the mutual information of a feature with the class but also the *conditional* mutual information of a feature with the class, given the values of other features.

Moreover, features can be simply presented incrementally to the learner or, alternatively, features that are no longer needed can be substituted by the currently most relevant features.

In [8], we investigated several approaches to feature ordering and selection based on the considerations mentioned above. The different strategies were categorized with respect to the following properties and are summarized in Table 2 (see [8] for a detailed discussion):

– Considered features (column 3 of Table 2): Each ensemble member is specialized on a certain region of the instance space. As the distribution over the examples changes, the required special knowledge might shift. We considered the question whether prediction accuracy can be improved by dismissing features that no longer meet the very current requirements of the learning task.
– Considered examples (column 4 of Table 2): Since one of boosting's basic principle is to focus the base learner on examples which are misclassified by the current ensemble, we considered the question whether predictive accuracy can be improved by presenting to the base learner features which are especially helpful to correctly classify the examples which have been misclassified so far.
– Sorting criterion (column 5 of Table 2): In addition to the mutual information of each single feature with the class we computed the features' conditional mutual information (CMI) with the class, given another feature, to investigate whether learning results can be improved by considering groups of features which optimally separate the given examples.
– Set evolution strategy (column 6 of Table 2): We considered the question whether we can improve prediction accuracy by substituting features that are no longer needed by currently most relevant features instead of augmenting the set of features to be considered for refinement.
– Partitioning continuous features (last column of Table 2): We investigated whether learning results can be improved by discretizing continuous features in a way that reflects the current distribution over the training examples.

For the purpose of this paper, we have selected two of the most successful strategies according to the results of [8], and adopted them to the non-uniform cost setting. These two strategies are discussed in more detail in the next section.

4 Non-Uniform Cost Feature Selection

When data are to be held in a relational database management system (RDBMS), we can no longer assume that the cost of accessing each feature is uniform. The dominant cost in such a setting is the scan of a relation, so that the cost of using a feature from a relation that needs to be scanned anyway is comparatively small, whereas adding a feature from a relation that has not been used before is high. On this account, we extend the more informed feature selection strategies investigated in [8] to the setting where costs are allocated to joins of relations in a database. To this end, we modify two of the most successful strategies, V2 and V4 in Table 2, into variants that take into account from which relations the features to be added are taken, and prefer to add features from an already used relation first.

Table 2. Strategies to feature ordering and selection based on the distributional information present in boosting. The strategies differ with respect to the features and examples, respectively, considered for establishing a new feature order, the applied sorting criterion, the set evolution strategy, and the discretization strategy for continuous features. The n in Set Evolution Strategy "Substitute n" denotes the number of features already presented to the learner.

Group ♯	Version ♯	Considered Features	Considered Examples	Sorting Criterion	Set Evolution Strategy	Partitioning Continuous Features
I	V1	Remaining	All	MI	Add 1	N
	V2	Remaining	Misclassified	MI	Add 1	N
II	V3	Remaining	All	MI	Add 1	Y
	V4	Remaining	Misclassified	MI	Add 1	Y
	V5	All	All	MI	Substitute n, Add 1	Y
	V6	All	Misclassified	MI	Substitute n, Add 1	Y
III	V7	All	All	MI	Substitute 1	Y
	V8	All	Misclassified	MI	Substitute 1	Y
IV	V9	Remaining	All	CMI	Add 1	Y
	V10	Remaining	Misclassified	CMI	Add 1	Y
	V11	All	All	CMI	Add 1	Y
	V12	All	Misclassified	CMI	Add 1	Y
V	V13	Remaining	All	CMI	Substitute 2	Y
	V14	Remaining	Misclassified	CMI	Substitute 2	Y
	V15	All	All	CMI	Substitute 2	Y
	V16	All	Misclassified	CMI	Substitute 2	Y

In V2 and V4, the available features present in the different relations are – exactly as in the baseline strategy C^2RIB^D – initially ordered based on the heuristic relevance measure of mutual information (MI) [22, 16]. The mutual information $MI(F_1, F_2)$ between two features F_1, F_2 is defined as the difference between the entropy of F_1 and the entropy of F_1 given F_2, i.e. as the amount of information about the possible values $(f_{11}, f_{12}, ...)$ of feature F_1 that is obtained when the value $f \in \{f_{21}, f_{22}, ...\}$ of feature F_2 is known:

$$\begin{aligned} MI(C, F_j) &= E(C) - E(C|F_j) \\ &= \sum_{i=1}^{m_j} \sum_{c=1}^{k} p(C = c, F_j = f_i) \ \ln \ \frac{p(C = c, F_j = f_i)}{p(C = c)p(F_j = fi)} \end{aligned} \qquad (1)$$

with k possible classes and m_j possible values of feature F_j.

In all cases, the learner starts with the top two features according to the order initially determined. In V2 and V4, opposed to the baseline method, every time a feature is requested by the learner, a new feature order is determined based on the current distribution. In both these versions, the remaining features are re-ordered based on the misclassified examples' current weights, and the best feature with respect to the new ranking is added to the set of active features.

In V2, as in the baseline strategy, continuous features are once initially discretized using an entropy based method [5]. By comparison, in V4, before a new feature order is determined based on the MI relevance measure, continuous features are discretized in a way that reflects the current distribution over the training examples. To this end, an objective function of the constrained confidence-rated boosting framework (cf. [10]) is applied to partition the continuous range of a feature F such that the training examples are optimally separated by F according to the current distribution.

Here, the strategies of V2 and V4 are modified into relation oriented variants $V2_R$ and $V4_R$, respectively. In $V2_R$ and $V4_R$, the features are foremost activated according to their position in the current feature order under the constraint that features belonging to a relation for which another feature has already been presented to the learner are activated before any feature belonging to a relation not yet known to the learner. In this way, the two strategies avoid accessing additional relations as long as possible.

5 Empirical Evaluation

We evaluated the different feature set evolution strategies on a total of six learning problems: two classical ILP domains, Mutagenicity [23] (prediction of mutagenic activity of 188 molecules (description $\mathcal{B}_4$)) and QSARs, Quantitative Structure Activity Relationships, [12, 11] (prediction of a greater-activity relationship between pairs of compounds based on their structure), one artificial problem, the Eastbound Trains[1] proposed by Ryszard Michalski (prediction of trains' directions based on their properties), and three general knowledge and data mining tasks, Task A and AC of the PKDD Discovery Challenge 2000 [1] (classification of loans, where Task AC is based on all loans, and Task A only on the closed loans from Task AC), and Task 2 of the KDD Cup 2001 [2] (prediction of gene functions).

[1] The examples were generated with the Random Train Generator available at http://www-users-cs-york.ac.uk/~stephen/progol.html

The standard version C^2RIB^D and each of the versions described in Section 4 is run with $T = 50$ iterations of the base learner. In all experiments, the threshold α is set to 1.01 (cf. 2.5d in Table 1). The value 1.01 has been empirically determined on the domain of Mutagenicity [23], and has not been modified for subsequent experiments on the other domains in order to ensure proper cross validation results.

As in [8], the gradients of the examples' mean margins are averaged for the standard version C^2RIB^D over the last $T_l = 10$, and for the more complex feature set evolution strategies over the last $T_l = 3$ iterations (cf. 2.3 in Table 1) (see [8] for a detailed discussion).

Table 3. Accuracy ± standard deviation, learning time ± standard deviation, and number of requested features ± standard deviation after 50 iterations for C^2RIB^D and two of the most successful feature set evolution strategies, V2 and V4, investigated in [8], together with their respective relation oriented variants $V2_R$ and $V4_R$ on several multirelational domains

V ♯	KDD01	Mutagenicity	PKDD-A	PKDD-C	QSARs	Trains
	Acc±SD	Acc±SD	Acc±SD	Acc±SD	Acc±SD	Acc±SD
	Time±SD	Time±SD	Time±SD	Time±SD	Time±SD	Time±SD
	♯Fs±SD	♯Fs±SD	♯Fs±SD	♯Fs±SD	♯Fs±SD	♯Fs±SD
C^RIB^D	90.15±7.92	83.50±5.80	86.70±6.64	88.57±2.95	78.76±1.76	80.00±15.32
	13.9±5.64	0.55±1.45	1.28±0.55	11.91±8.46	22.11±7.17	0.09±0.012
	5.5±3.4	4.4±2.0	5.5±1.8	6.3±1.8	3.4±1.5	5.9±1.9
V2	90.33±7.73	85.60±4.45	86.70±6.64	88.87±3.35	80.77±2.93	81.67±16.57
	12.0±3.68	1.7±2.74	3.88±4.13	9.64± 6.71	25.37±0.83	0.07±0.009
	5.4±3.9	7.4±1.2	6.1±2.5	8.6±3.0	2.3±0.5	5.7±1.2
$V2_R$	89.12±8.42	85.60±4.45	84.09±8.78	88.87±3.6	82.0±1.56	93.75±12.4
	14.6±4.95	0.7±7.2	0.83±0.12	1.12±0.18	22.34±0.32	0.09±0.017
	9.4±2.42	5.2±2.7	10.6±3.1	7.7±4.3	4±1.4	5.75±2.32
V4	90.57±7.32	86.19±5.01	86.70±6.64	88.87±3.35	80.96±2.81	83.33±11.11
	11.78±3.46	0.93±0.66	6.16±6.57	9.95±10.46	33.70±4.93	0.08±0.009
	3.5±2.3	4.9±1.7	5.9±2.6	5.7±4.1	6.0±2.31	6.6±1.0
$V4_R$	89.91±8.95	86.89±4.80	86.70±6.6	88.87±3.35	81.19±2.79	79.17±17.25
	6.12±1.84	2.99±6.8	3.79±7.89	1.25±0.16	12.07±6.06	0.06±0.007
	8±3.35	5.7±1.8	9.2±4.26	8±4.08	4±2.83	4±2

The resulting predictive accuracies, learning time, and the average number of features required by the learner are depicted in Table 3 together with the standard deviations. The predictive accuracy is estimated by 10-fold-cross validation with the exception of the QSARs domain, where 5-fold-cross validation is used[2], and the Eastbound Trains, where the data is split into one training and test set partition, and the results are averaged over 10 iterations of the experiment.

[2] The folds correspond exactly to the data described in [12].

The results indicate that our approach to active relation oriented feature selection indeed seems to be feasible. Although an increase in the number of features ultimately used for learning is noticeable from the informed active feature selection strategies to their relation oriented variants, we can see that positive effects on the learning time can be achieved, especially in domains with a larger number of relations of higher arity. For example, in the domain of the PKDD Discovery Challenge 2000, runtime is reduced to almost 1/10th for task C. The data about the loan status of clients which we use in our experiments are given in form of seven relations with a total number of 24 features, i.e. most of the relations are of high arity.

Classification accuracies obtained with the relation oriented variants, $V2_R$ and $V4_R$, are en par with their respective baseline versions, V2 and V4. V2 and V4 outperform C^2RIB^D in all but one domains where the performance is the same.

6 Related Work

To our knowledge, orienting feature selection towards the relations that are used has not been considered before in the literature. We therefore briefly summarize other work on feature selection in multirelational learning and with boosting algorithms.

The work probably most related to our work is [14], where AdaBoost [19] is combined with MOLFEA, an inductive database for the domain of biochemistry [13]. In [14], AdaBoost is employed to identify particularly difficult examples for which MOLFEA constructs new special purpose structural features. AdaBoost re-weighting episodes and MOLFEA feature construction episodes are alternated. In each iteration, a new feature constructed by MOLFEA is presented to a propositional learner, the examples are re-weighted in accordance to the base classifier learned by it, and a new feature is constructed by MOLFEA based on the modified weights. In contrast, our approach actively decides when to include new features from the list of ranked existing features with the central goal of including new features only when absolutely necessary in order to be maximally efficient. This means that in principle the two approaches could be easily combined, for example by calling a generator of new features whenever the list of existing features has been exhausted.

[21] propose a wrapper model utilizing boosting for feature selection. In their approach, alternative feature subsets are assessed based on the underlying booster's optimization criterion. The feature subset optimal according to this criterion is then presented as a whole to a learner.

[4] introduces a *Boosting Based Hybrid* approach to feature selection. In each iteration of AdaBoost, the feature is selected that has, among all previously unselected features, the highest information gain on the training set weighted according to AdaBoost and the current feature subset. This process terminates when the training error does not further decrease by selecting additional features.

Similar to [21] and [4], we combine heuristic relevance measures and the distributional information present in boosting to determine optimal features. However, we embed the feature selection process into the learning process and thus arrive at an *active* strategy to feature selection.

7 Conclusion

If multirelational learning algorithms are to be used for larger scale data mining problems, it will ultimately be required that they run on the basis of data stored on disk, e.g. in a relational database management system. In such a setting, access costs for different features and relations will vary widely. In this paper, in the context of a boosted multirelational learner, we have proposed a very first small step towards cost-oriented feature selection for multirelational learning. We have considered the issue that accessing features from relations that need to be scanned anyway might be significantly less expensive than using features from additional relations. Based on this assumption, we have modified two previously proposed active feature selection strategies to prefer features from relations that have already been used over features from additional relations. As shown by the empirical results reported in the preceding section, assuming that the cost of accessing each feature of different relations is uniform is not appropriate even when the data are stored in main memory. In our experiments, runtime was reduced to almost 1/10th, indicating that in a disk-based setting, the gains might even be higher. As was to be expected, the effect was most pronounced in domains which had several relations of higher arities, as might be typical in a standard (non-star schema) database design.

Of course, the approach presented here is quite simple, and it is easy to think of more sophisticated ways of addressing the issue of feature access cost in multirelational learning. In the proposed strategies, we have simply assumed that relations that are already in use are to be preferred over additional relations, which is equivalent to a very simple two class cost model. In future work, we plan to refine this model to take into account *a priori* estimates of access cost based on relation size and on knowledge about the available indices on the relation. As an alternative, in a boosting setting, we can collect runtime information during the early rounds of boosting to arrive at more informed estimates of the access costs resulting from a particular feature set. Finally, we will integrate our approach with a disk base storage system in order to verify which gains are possible in this setting, and whether they are sufficient to make boosted multirelational learning feasible without storing all data in main memory.

This work was partially supported by DFG (German Science Foundation), projects WR 40/1-3 ("Active Learning") and WR 40/2-1 ("Hybrid Methods"). We would like to thank Jörg Kaduk for valuable discussions and helpful comments on previous versions of this paper.

References

1. P. Berka. Guide to the financial Data Set. In: *A. Siebes and P. Berka, editors, PKDD2000 Discovery Challenge*, 2000.
2. J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, Sh. Morishita, D. Page, and J. Sese. KDD Cup 2001 Report. In: *SIGKDD Explorations, 3(2):47-64*, 2002.
3. W. Cohen and Y. Singer. A Simple, Fast, and Effective Rule Learner. *Proc. of 16th National Conference on Artificial Intelligence*, 1999.
4. S. Das. Filters, Wrappers and a Boosting-based Hybrid for Feature Selection. *Proc. of 18th International Conference on Machine Learning*, 2001.
5. U.M. Fayyad, and K.B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. *Proc. of 13th Int. Joint Conf. on AI*, 1993.
6. Y. Freund, and R.E. Schapire. Experiments with a New Boosting Algorithm. *Proc. of 13th International Conference on Machine Learning*, 1996.
7. A.J. Grove, and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. *Proc. of 15th National Conf. on AI*, 1998.
8. S. Hoche, and S. Wrobel. A Comparative Evaluation of Feature Set Evolution Strategies for Multirelational Boosting. *Proc. of 13th Int. Conf. on Inductive Logic Programming (ILP)*, 2003.
9. S. Hoche, and S. Wrobel. Scaling Boosting by Margin-Based Inclusion of Features and Relations. *Proc. 13th European Conf. on Machine Learning (ECML'02)*, 2002.
10. S. Hoche, and S. Wrobel. Relational Learning Using Constrained Confidence-Rated Boosting. *Proc. 11th Int. Conf. on Inductive Logic Programming (ILP)*, 2001.
11. R.D. King, A. Srinivasan, and M. Sternberg. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):411-434, 1995.
12. R.D. King, S. Muggleton, R.A. Lewis, and M.J.E. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proc. of the National Academy of Sciences of the USA* 89(23):11322-11326, 1992.
13. S. Kramer, and L. De Raedt. Feature construction with version spaces for biochemical applications. *Proc. of the 18th ICML*, 2001.
14. S. Kramer. Demand-driven Construction of Structural Features in ILP. *Proc. 11th Int. Conf. on Inductive Logic Programming (ILP)*, 2001.
15. M.-A. Krogel and S. Wrobel. Feature Selection for Propositionalization. In: S. Lange, K. Satoh, and C. H. Smith (eds.) *Proceedings of the Fifth International Conference on Discovery Science (DS). LNCS 2534, Springer-Verlag*, 2002.
16. W.J. McGill. Multivariate information transmission. *IRE Trans. Inf. Theory*, 1995.
17. D. Opitz, and R. Maclin. Popular Ensemble Method: An Empirical Study. *Journal of Artificial Intelligence Research 11, pages 169-198*, 1999.
18. J.R. Quinlan. Bagging, boosting, and C4.5. *Proc. of 14th Nat. Conf. on AI*, 1996.
19. R.E. Schapire. Theoretical views of boosting and applications. *Proceedings of the 10th International Conference on Algorithmic Learning Theory*, 1999.
20. R.E. Schapire, Y. Freund, P.Bartlett, and W.S. Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics, 26(5):1651-1686*, 1998.
21. M. Sebban, and R. Nock. Contribution of Boosting in Wrapper Models. In: *J.M. Zytkow, and J. Rauch, eds, Proc. of the PKDD'99*, 1999.
22. C.E. Shannon. A mathematical theory of communication. *Bell. Syst. Techn. J.*, 27:379-423, 1948.

23. A. Srinivasan, S. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 1996.

A Structural GEM for Learning Logical Hidden Markov Models

Kristian Kersting[1], Tapani Raiko[2], and Luc De Raedt[1]

[1] Institute for Computer Science, Machine Learning Lab, University of Freiburg
Georges-Koehler-Allee 079, 79112 Freiburg, Germany
[2] Helsinki University of Technology, Laboratory of Computer and Information Science, P.O. Box 5400,02015 HUT, Finland

Abstract. Traditional hidden Markov models (HMMs) specify probability distributions over sequences of *flat* symbols. Recently, logical hidden Markov models (LOHMMs) have been introduced to deal with sequences of *structured* symbols. Within LOHMMs, logical atoms are used to represent output and state symbols. Together with unification, this kind of abstraction can lead to LOHMMs that have a much smaller number of parameters equivalent HMMs (typically an order of magnitude). However, the compactness of LOHMMs comes at the expense of a more complex structure learning problem. Indeed, different abstraction levels have to be explored. In this paper, we propose for the first time a method for learning the structure of LOHMMs from data. The method essentially adapts Friedman's *structural EM* (SEM) algorithm. More precisely, it combines *generalized expectation maximization* (GEM), which optimizes parameters, with structure search for model selection using *inductive logic programming* refinement operators. We provide convergence and experimental results that show its effectiveness.

Keywords: Multi-Relational Data Mining, Inductive Logic Programming, Expectation Maximization, Hidden Markov Models

1 Introduction

Hidden Markov models [26] (HMMs) are extremely popular for analyzing sequential data. Areas of application include computational biology, user modelling, speech recognition, empirical natural language processing, and robotics. Despite their successes, HMMs have a major weakness: they handle only sequences of flat, i.e., unstructured symbols. However, in many applications the symbols occurring in sequences are structured, e.g., in computational biology [17], web mining [1], information extraction from structured doucments, and user modelling. Consider, e.g., the sequence of UNIX commands `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... Such data has been used to train hidden Markov models for *anomaly detection* [19]. Anomaly detection is the task to develop a model or profile of the normal working state of a computer system user and to detect anomalous conditions as deviations from expected behaviour patterns. The user's current behavioral state, i.e., the hidden state, might be "idle", "writing", or "hacking". However, as the above command sequence shows, UNIX commands may have parameters (such as filenames), may return information (such as a return code) and may have other properties (such as current working directory, cost, etc.). Thus, commands are essentially *structured* symbols. Traditional HMMs cannot easily deal with this type of structured sequences. Typically, the application of HMMs requires either 1) ignoring the structure of the commands (i.e., the parameters), or 2) taking all possible parameters explicitly into account. The former approach results in serious information loss; the latter leads to a combinatorial explosion in the number of parameters and as a consequence inhibits generalization.

The above sketched problem with HMMs is akin to the problem of dealing with structured examples in traditional machine learning algorithms. This problem has extensively been studied in the fields of inductive logic programming [23] and multi-relational learning [6]. Recently, Kersting et. al [16] proposed *logical hidden Markov models* (LOHMMs) as an inductive logic programming approach to overcome the problem and have proven the usefulness of LOHMMs on a computational biology application [17]. The key idea underlying LOHMMs is to employ logical atoms as structured (output and state) symbols. Using logical atoms, the above UNIX command sequence can be represented as `emacs(lohmms.tex)`, `ls`, `latex(lohmms.tex)`, ... There are two important motivations for using logical atoms at the symbol level. First, *variables* in the atoms allow us to make abstraction of specific symbols. E.g., the logical atom `emacs(X, luc)` represents all files that user Luc could edit using emacs. Secondly, *unification* allows us to share information among hidden states and between hidden states and observations. E.g., the sequence `emacs(X, luc)`, `latex(X, luc)` represents that the same file is used as an argument for both Emacs and LaTeX. Due to the use of abstraction, one can devise LOHMMs that are an order of magnitude smaller than equivalent HMMs. For instance, the trained LOHMM in [17] had only 120 parameters, corresponding to an instantiated traditional HMM with more than 62000 parameters.

However, the compactness of LOHMMs comes at the expense of a more complex structure learning, i.e., model selection problem. Indeed, different abstraction levels have to be explored. So far, no method for solving the problem has been proposed. This is a significant problem for several reasons. First, eliciting LOHMMs from experts can be a laborious and expensive process. Second, traditional hidden Markov models are commonly learned by estimating the maximum likelihood parameters of a fixed, fully connected model. Such an approach is not useful for LOHMMs because different levels of abstraction should be explored. Finally, logical hidden Markov models naturally contain *hidden* variables, i.e. no data case will describe the value of these variables. In such a case, learning is more difficult than in the *complete* data case. To evaluate the optimal choice of parameters for a candidate LOHMM, we must perform nonlinear optimization using e.g. the EM algorithm. In each iteration, the EM algorithm computes the probabilities of several events for each data case. Thus, learning parameters with the EM is significantly more difficult. The contribution of the present paper is a novel method for learning logical hidden Markov models. We tackle the problems by adapting Friedman's *structural EM* (SEM) algorithm [9, 10]. More precisely, our approach combines a generalized expectation maximization (GEM) algorithm, which optimizes parameters, with structure search for model selection using ILP refinement operators. In doing so, we explore different abstraction levels due to the *inductive logic programming* (ILP) refinement operators (see e.g. [23, 6]), and due to SEM's underlying idea, we reduce the learning problem to one that is similar to learning in the complete data case, which can be solved more efficiently.

The paper is organized as follows. In Section 2, we briefly review some logical concepts; in Section 3, we review logical hidden Markov models; in Section 4, we formalize the learning setting; in Section 5, we present a naïve learning algorithm; in Section 6, we introduce a structural, generalized EM for learning LOHMMs from data. The procedure is experimentally evaluated in Section 7. Before we conclude, we discuss related work.

2 Preliminaries

A *first-order logic alphabet* Σ is a set of relation symbols $\mathtt{r}$ with arity $m \geq 0$, written $\mathtt{r/m}$, and a set of functor symbols $\mathtt{f}$ with arity $n \geq 0$, written $\mathtt{f/n}$, a. If $n = 0$ then $\mathtt{f}$ is called a constant, if $m = 0$ then r is called a propositional variable. (We assume that at least one constant is given.) An *atom* $\mathtt{r}(\mathtt{t_1}, \ldots, \mathtt{t_m})$ is a relation symbol $\mathtt{r}$ followed by a bracketed m-tuple of terms $\mathtt{t_i}$. A *term* is a variable $\mathtt{V}$ or a functor symbol $\mathtt{f}$ of arity n immediately followed by a bracketed n-tuple of terms $\mathtt{s_j}$, i.e., $\mathtt{f}(\mathtt{s_1}, \ldots, \mathtt{s_n})$. An *iterative clause* is a formula of the form $\mathtt{A} \leftarrow \mathtt{B}$ where $\mathtt{A}$ and the $\mathtt{B}$ are logical atoms. In the present paper, a *logic program* consists of a set of iterative clauses. A substitution $\theta = \{\mathtt{V_1/t_1}, \ldots, \mathtt{V_k/t_k}\}$, e.g. $\{\mathtt{X/tex}\}$, is an assignment of terms $\mathtt{t_i}$ to variables $\mathtt{V_i}$. Applying a substitution σ to a term, atom or clause e yields the instantiated term, atom, or clause $e\sigma$ where all occurrences of the variables $\mathtt{V_i}$ are simultaneously replaced by the term $\mathtt{t_i}$, e.g. $\mathtt{ls(X)} \leftarrow \mathtt{emacs(F, X)}\{\mathtt{X}/tex\}$ yields $\mathtt{ls(tex)} \leftarrow \mathtt{emacs(F, tex)}$. A term, atom or clause e is called *ground* when it contains no variables, i.e., $vars(E) = \emptyset$. The *Herbrand base* of Σ, denoted as hb_Σ, is the set of all ground atoms constructed with the predicate and functor symbols in Σ. The set $G_\Sigma(\mathtt{A})$ of an atom $\mathtt{A}$ consists of all ground atoms $\mathtt{A}\theta$ that belong to hb_Σ.

3 Logical Hidden Markov Models

The logical component of a traditional hidden Markov model corresponds to a *Mealy machine* [13], i.e., to a finite state machine where the output symbols are associated with transitions [3]. This is essentially a propositional representation

[3] Often the output symbols are associated with states within HMMs. In this case, the logical component actually corresponds to a *Moore* machine. However, Moore and

because the symbols used to represent states and outputs are flat, i.e. not structured. The key idea underlying *logical* hidden Markov models is to replace these flat symbols by abstract symbols. An abstract symbol $\mathtt{A}$ is — by definition — a logical atom. It is abstract in that it represents the *set* of ground atoms $G_\Sigma(\mathtt{A})$. We assume that the alphabet is typed which means that constant symbols can appear as arguments of some particular relation symbols only. Therefore, assume a set $D_1, \ldots, D_n$ of (finite) domains (of constants), and let $\mathrm{D}(\mathtt{r/m})_i$ refer to the domain associated with the ith argument of a relation $\mathtt{r/m}$. Ground atoms then play the role of the traditional symbols used in a hidden Markov model.

Example 1 *Consider the alphabet Σ_1 which has as constant symbols* `tex`*,* `prog`*,* `hmm1`*, and* `lohmm1`*, and as relation symbols* `ls/0`*,* `emacs/1`*,* `latex/1`*,* `ls/1`*,* `emacs/2`*, and* `latex/2`*. Then the atom* $\mathtt{emacs(File,tex)}$ *represents the set* $\{\mathtt{emacs(hmm1,tex)}, \mathtt{emacs(lohmm1,tex)}\}$*. The alphabet is typed such that the constant symbols* `hmm1` *and* `lohmm1` *are used as filenames, i.e.,* $\mathrm{D}(\mathtt{latex/1})_1 = \mathrm{D}(\mathtt{latex/2})_1 = \mathrm{D}(\mathtt{emacs/1})_1 = \mathrm{D}(\mathtt{emacs/2})_1 = \{\mathtt{hmm1}, \mathtt{lohmm1}\}$ $= D_1$*, and* `tex` *and* `prog` *are used as users, i.e.,* $\mathrm{D}(\mathtt{ls/1})_2 = \mathrm{D}(\mathtt{latex/2})_2 = \mathrm{D}(\mathtt{emacs/2})_2 = \{\mathtt{tex}, \mathtt{prg}\} = D_2$*. We thus avoid useless instantiations such as* $\mathtt{emacs(tex,tex)}$*. Two different versions of the relation symbol encode that the* user *argument is not observed but only represented in the hidden states.*

The use of atoms instead of flat symbols allows us to analyze logical and structured sequences such as $\mathtt{emacs(hmm1)}, \mathtt{ls}, \mathtt{latex(hmm1)}$. Thus, the sequences of a logical hidden Markov model are structured. In addition, atoms are crucial in defining *abstract transitions*.

Definition 1 *(Abstract Transition) An* abstract transition *is an expression of the form* $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ *where* $p \in [0,1]$*, and* $\mathtt{H}$*,* $\mathtt{B}$ *and* $\mathtt{O}$ *are atoms.*

In this definition, the atoms $\mathtt{H}$ and $\mathtt{B}$ represent abstract states and $\mathtt{O}$ represents an abstract output symbol. The semantics of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ is that if one is in one of the states in $G_\Sigma(\mathtt{B})$, say $\mathtt{B}\theta_\mathtt{B}$, one will go to one of the states in $G_\Sigma(\mathtt{H}\theta_\mathtt{B})$, say $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H}$, while emitting one of the output symbols in $G_\Sigma(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H})$, say $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O}$.

Example 2 *Consider the abstract transition* $0.8 : \mathtt{latex(File,tex)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File,tex)}$ *and assume that we are in state* $\mathtt{emacs(hmm1,tex)}$*, i.e.,* $\theta_\mathtt{B} = \{\mathtt{File/hmm1}\}$*. Then the abstract transition specifies that there is a probability of* 0.8 *that the next state will be in* $G_{\Sigma_1}(\mathtt{latex(hmm1,tex)}) = \{\mathtt{latex(hmm1,tex)}\}$ *(i.e., the probability is* 0.8 *that the next state will be* $\mathtt{latex(hmm1,tex)}$*), and that one of the output symbols in* $G_{\Sigma_1}(\mathtt{emacs(hmm1)}) = \{\mathtt{emacs(hmm1)}\}$ *(i.e.,* $\mathtt{emacs(hmm1)}$*) will be emitted.*

The above example was interesting because it uses unification between the filename arguments. On the other hand, it is simple because $\theta_\mathtt{H}$ and

Mealy machines are interconvertible. We decided to adapt Mealy machines because they fit our logical setting more intuitively.

$\theta_{\mathtt{O}}$ are both empty. The situation becomes more complicated when these substitutions are not empty because then the resulting state and output symbol sets are not necessarily singletons. Indeed, for the transition $0.8 : \mathtt{emacs(File', tex)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$ the resulting state set would be $G_{\Sigma_1}(\mathtt{emacs(File', tex)}) = \{\mathtt{emacs(hmm1, tex)}, \mathtt{emacs(lohmm1, tex)}\}$. Thus the transition is non-deterministic because there are two possible resulting states. We therefore need a mechanism to assign probabilities to these possible states. This is realized using the notion of a selection probability.

Definition 2 *(Selection Distribution) The* selection distribution μ *specifies for each abstract state (respectively observation)* $\mathtt{A}$ *over the alphabet* Σ *a distribution* $\mu(\cdot \mid \mathtt{A})$ *over the possible instantiations of* $\mathtt{A}$*, i.e., over* $G_{\Sigma}(\mathtt{A})$*.*

To continue our example, assume that we are in a state $\mathtt{latex(hmm1, tex)}$ and consider the transition $0.8 : \mathtt{emacs(File', tex)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$. Let $\mu(\mathtt{emacs(hmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.4$ and $\mu(\mathtt{emacs(lohmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.6$. Then there would be a probability of $0.4 \times 0.8 = 0.32$ that the next state would be $\mathtt{emacs(hmm1, tex)}$ and of $0.6 \times 0.8 = 0.48$ that it would be $\mathtt{emacs(lohmm1, tex)}$.

Taking into account the selection distribution μ, the meaning of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ can be summarized as follows. Let $\mathtt{B}\theta_{\mathtt{B}} \in G_{\Sigma}(\mathtt{B})$, $\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}} \in G_{\Sigma}(\mathtt{H}\theta_{\mathtt{B}})$ and $\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}} \in G_{\Sigma}(\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}})$. Then the model makes a transition from state $\mathtt{B}\theta_{\mathtt{B}}$ to $\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}}$ and emits symbol $\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}}$ with probability

$$p \cdot \mu(\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}} \mid \mathtt{H}\theta_{\mathtt{B}}) \cdot \mu(\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}} \mid \mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}). \tag{1}$$

To represent μ, any probabilistic representation can — in principle — be used, e.g. a Bayesian network. To reduce the model complexity, we will however use a naïve Bayes approach throughout the remainder of the this paper. In this approach, we associate to each domain D_i used as type a probability distribution P_{D_i} over D_i. Let $\mathrm{vars}(\mathtt{A}) = \{\mathtt{V_1}, \ldots, \mathtt{V_l}\}$ be the variables occurring in some atom $\mathtt{A}$ over $\mathtt{r/m}$, and let $\sigma = \{\mathtt{s_1/V_1}, \ldots \mathtt{s_l/V_l}\}$ be a substitution grounding $\mathtt{A}$. Each $\mathtt{V_j}$ is then considered a random variable over the domain of the first argument of $\mathtt{r/m}$ it appears in, denoted by $D_{\mathtt{V_j}}$. Then, $\mu(\mathtt{A}\sigma \mid \mathtt{A}) = \prod_{j=1}^{l} P_{D_{\mathtt{V_j}}}(\mathtt{V_j} = \mathtt{s_j})$. For instance for Σ_1, $\mu(\mathtt{emacs(hmm1, tex)} \mid \mathtt{emacs(F, E)})$ is computed as the product of $P_{D_1}(\mathtt{F} = \mathtt{hmm1})$ and $P_{D_2}(\mathtt{E} = \mathtt{tex})$.

So far, the semantics of a single abstract transition has been defined. A logical hidden Markov model will usually consist of multiple abstract transitions and this creates a further complication.

Example 3 *Consider the abstract transitions* $0.8 : \mathtt{latex(File, tex)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File, tex)}$ *and* $0.4 : \mathtt{ls(User)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File, User)}$*. These two transitions make conflicting statements about the state resulting from* $\mathtt{emacs(hmm1, tex)}$*. Indeed, according to the first transition, the probability is* 0.8 *that the resulting state is* $\mathtt{latex(hmm1, tex)}$ *and according to the second one it is* 0.4 *that it is* $\mathtt{ls(tex)}$*.*

This complication can be solved by taking into account the subsumption (or generality) relation among the B-parts of the two abstract transitions. Indeed, the B-part of the first transition $\mathtt{B}_1 = \mathtt{emacs(File,tex)}$ is more specific than that of the second transition $\mathtt{B}_2 = \mathtt{emacs(File,User)}$ because there exists a substitution $\theta = \{\mathtt{User/tex}\}$ such that $\mathtt{B}_2\theta = \mathtt{B}_1$, i.e., $\mathtt{B}_2$ subsumes $\mathtt{B}_1$. Therefore $G_\Sigma(\mathtt{B}_1) \subseteq G_\Sigma(\mathtt{B}_2)$. The first transition can therefore be regarded as more informative than the second one. It should therefore be preferred over the second one when starting from `emacs(hmm1,tex)`. We will also say that the first transition is *more specific* than the second one. Remark that this *generality* relation imposes a partial order on the set of all transitions. These considerations lead to the strategy of only considering the maximally specific transitions that apply to a state in order to determine the successor states. This implements a kind of *exception handling* or *default reasoning*, e.g., in the above example, the first transition is an exception to the second one. This conflict resolution strategy will work properly provided that the bodies of all maximally specific transitions (matching the starting state) represent the same abstract state. This can be enforced by requiring the set of abstract transitions to be *well-founded* (for each predicate), i.e., by requiring that every subset of the set of abstract transitions has a unique maximally specific abstract body state. E.g., if the body of the second abstract transition in our example would have been replaced by `emacs(hmm1,User)` then the set of abstract transactions would not be well-founded.

By now we are able to formally define logical hidden Markov models.

Definition 3 *(Logical Hidden Markov Model) A* logical hidden Markov model *is a tuple $M = (\Sigma, \mu, \Delta)$ where Σ is a logical alphabet, μ a selection probability over Σ and Δ is a set of well-founded abstract transitions. Let $\mathbf{B}$ be the set of all atoms that occur as the body part of transitions in Δ. We require*

$$\forall \mathtt{B} \in \mathbf{B} : \sum_{\substack{\mathrm{cl}\in\Delta,\\ \mathrm{body(cl)}=\mathtt{B}}} \Pr(\mathrm{cl}) = 1.0. \qquad (2)$$

The semantics of LOHMMs are as follows.

Theorem 1 (Semantics). *A logical hidden Markov model over a language Σ defines a discrete time stochastic process where the domain of the random variables is* $\mathrm{hb}(\Sigma)$*, i.e., the observation and hidden states. The induced probability measure over* $\mathrm{hb}(\Sigma)$ *is unique.*

Note that traditional HMMs are a special type of LOHMMs in which Σ contains only relation symbols of arity 0 and the selection probability is irrelevant. Thus, LOHMMs generalize traditional HMMs. However, LOHMMs allow to go beyond the expressivity of HMMs. Consider again the UNIX command sequence `emacs lohmms.tex, ls, latex lohmms.tex`. The filename `lohmms.tex` might be meaningless. What more matters is that both, emacs and LaTeX get the same file as input. The filename itself bears no information, i.e. it is an *identifier.* Assume a countably infinite set of constant symbols $i_1, i_2, \ldots$ Each time we would select a specific filename such as `lohmms.tex`, we select one of the i_j in increasing order if the filename `lohmms.tex` was not encountered before. Otherwise we

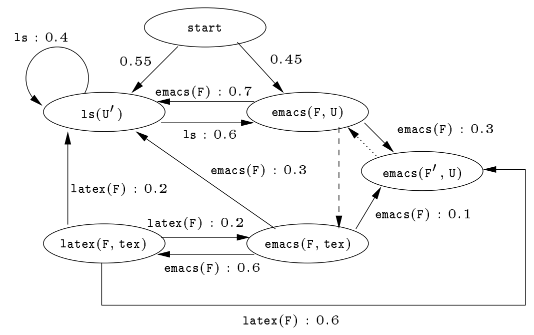


Fig. 1. A logical hidden Markov model.

select the same i_k, $k \leq j$, as before. What remains is the selection probability. Clearly, we do not want to specify a probability for each i_j. One solution is to use $P(i_1 \vee i_2 \vee \ldots)$. This translates to selecting a specific filename with out knowing which one. Consider e.g. our language Σ_1. Using identifiers instead of `hmm1` and `lohmm1`, the selection probability would be 1.0. Treating `lohmm1` exceptionally – e.g. because we would like to track it over time – its selection probability is p whereas that of `hmm1` would be $1-p$. In this way, a countably infinite set of (unspecific) inputs can be encoded in LOHMMs. Furthermore, using *functors* one can incorporate memory mechanisms into LOHMMs so that e.g. long distance correlations can be encoded.

Finally, let us note that logical hidden Markov models can be represented graphically. Figure 1 contains an example for analyzing Unix sequence commands. The underlying language Σ_2 consists of Σ_1 together with the constant symbols `other` denoting a not LaTeX user. In this graphical notation, abstract states are represented by vertices, abstract transitions by *solid edges*, and *dashed edges* represent the generality or subsumption ordering between abstract states. Furthermore, *dotted edges* connect identical abstract states. If we follow a transition to an abstract state with an outgoing dotted edge then the dotted edge will always be followed. Dotted edges are needed because the same abstract state can occur under different circumstances. Indeed, consider the transition $p : \mathtt{latex}(\mathtt{File}', \mathtt{User}') \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{User})$. Even though the atoms in the head and body of the transition are syntactically different they represent the same abstract state. Furthermore, to accurately represent the meaning of this transition we cannot use a solid edge from $\mathtt{latex}(\mathtt{File}, \mathtt{User})$ to itself, because this would actually represent the abstract transition $p : \mathtt{latex}(\mathtt{File}, \mathtt{User}) \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{User})$, whose semantics is is different. Indeed, the second transition only allows for transitions between identical states whereas the first abstract transition also allows for transitions between equivalent states.

The graphical representation and the conflict resolution technique realize one of our design principles: locally interpretable transitions.

4 The Learning Setting

So far, we assumed that the LOHMM is given. Learning LOHMMs from data will often be easier than constructing them by hand. In this section, we investigate and formally define the problem of learning logical hidden Markov models.

Definition 4 *(Learning Problem)* **Given** *a set* $\mathbf{O} = \{O_1, \ldots, O_m\}$ *of data cases, a set* $\mathcal{M}$ *of LOHMMs, and a scoring function* $score_{\mathbf{O}} : \mathcal{M} \mapsto \mathbb{R}$, **find** *a hypothesis* $M^* \in \mathcal{M}$ *that maximizes* $score_{\mathbf{O}}$.

Each *data case* $O_i = \mathtt{o_{i,1}o_{i,2} \ldots o_{i,T}}$ consists of a sequence $\mathtt{o_{ij}}$ of observations, i.e. of ground atoms. We write each observation in lower case to stress that they are ground atoms. For instance in the user modelling domain an example data case is `emacs(lohmms)`, `ls`, `emacs(lohmms)`. By analogy with the traditional hidden Markov model learning setting, a single data case $O_i \in \mathbf{O}$ only describes the observations evolving over time. The corresponding sequence $H_i = \mathtt{h_{i,0}h_{i,1} \ldots h_{i,T_i+1}}$ describing the evolution of the system's state over time is hidden, i.e., not specified in O_i. For instance, we do not know whether `emacs(lohmms)` has been generated by `emacs(lohmms,prog)` or `emacs(lohmms,tex)` (cf. Example 3).

The *hypothesis space* $\mathcal{M}$ consists of all candidate LOHMMs to be considered during search. We assume Σ to be given. Consequently, the domain declarations, i.e. the possible constants which can be selected by μ are apriori known. Furthermore, each model $M \in \mathcal{M}$ can be considered to be parameterized by a vector $\boldsymbol{\lambda}_M$ such that each (legal) choice of values $\boldsymbol{\lambda}_M$ defines according to Theorem 1 a probability distribution $P(\cdot \mid M, \boldsymbol{\lambda}_M)$ over $\mathrm{hb}(\Sigma)$. For the sake of simplicity, we will denote the underlying logic program (i.e., the set of abstract transitions without associated probability values) by M and abbreviate $\boldsymbol{\lambda}_M$ by $\boldsymbol{\lambda}$ as long as the model M is clear from the context. Furthermore, we also impose certain syntactic restrictions, i.e., a syntactic bias on the transitions to be induced. At this point, we wish to stress that the particular syntactic bias selected is a parameter of our framework, see e.g. [24]

To evaluate different candidates M, we assume a scoring function $score_{\mathbf{O}} : \mathcal{M} \mapsto \mathbb{R}$ which evaluates how well a given candidate M fits a given set $\mathbf{O}$ of data cases. The data cases are assumed to be independently sampled from identical distributions. A commonly employed scoring function is

$$score_{\mathbf{O}}(M, \boldsymbol{\lambda}) = \log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) - Pen(M, \boldsymbol{\lambda}, \mathbf{O}). \tag{3}$$

The scoring function can be derived from Bayesian analysis assuming MAP-estimates, a uniform prior over the parameters and a prior over model structures that prefers simpler ones. The term $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda})$ is the *log-likelihood* of the data $\mathbf{O}$ given the current choice of model $(M, \boldsymbol{\lambda})$. The log-likelihood has a statistical interpretation: the higher the log-likelihood, the closer $(M, \boldsymbol{\lambda})$ models the probability distribution induced by the data. The second term $Pen(M, \boldsymbol{\lambda}, \mathbf{O})$ is a penalty function that biases the scoring function to prefer simpler models. Motivated by the *minimum description length* score for Bayesian networks [18], we use the simple penalty $Pen(M, \boldsymbol{\lambda}, \mathbf{O}) = \log(m)/2 \cdot |\Delta|$ in the present paper. The penalty is independent of the model parameters and therefore can neglect it during parameter estimation.

5 A Naïve Learning Algorithm

For traditional HMMs, the learning problem collapses to parameter estimation (i.e., estimating the transition probabilities) because HMMs are usually fully connected. For LOHMMs, however, we have to account for different abstraction levels. Therefore, we split the learning problem into

1. searching for a model structure, i.e., the underlying logic program, and
2. parameter estimating where the transition and selection probabilities consitute the parameters of a LOHMM,

and apply the following informed, greedy search algorithm. It is a direct solution to the learning problem described in Definition 4. It takes as input an initial model M^0 and the data $\mathbf{O}$, and performs:

```
1:  Let $\boldsymbol{\lambda}^0 = \text{argmax}_{\boldsymbol{\lambda}}\, score_{\mathbf{O}}(M^0, \boldsymbol{\lambda})$
2:  **Loop** for $k = 0, 1, 2, \ldots$
3:    **Find** model $M^{k+1} \in \rho(M^k)\}$ that maximizes $score_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
4:    Let $\boldsymbol{\lambda}^{k+1} = \text{argmax}_{\boldsymbol{\lambda}}\, score_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
5:  **Until** convergence, i.e., no improvement in score
```

That is, at each stage k we choose a model structure and parameters among the currently best model and its neighbours (see below) that have the highest score. It stops, when there is no improvement in score. Note that in practice, we have to initialize the parameters of each model scored in lines 1 and 3 (e.g. randomly). To apply this algorithm to learning LOHMMs, we have to make line 3 more concrete. In the following two subsections, we will show (1) how to traverse the hypotheses space and (2) how to score hypotheses.

5.1 Traversing the Hypotheses Space

Let us start with the selection of the initial hypothesis M^0. It is governed by the idea that M^0 should in principle cover all possible observations and hidden state sequences (over the given language Σ). Therefore, an obvious candidate for M^0 (which we also used in our experiments) is the fully connected LOHMM built over all maximally general atoms over Σ. The maximally general atoms are expressions of the form $\mathtt{r}(\mathtt{X}_1, ..., \mathtt{X}_\mathtt{m})$, where the $\mathtt{X}_\mathtt{i}$ are different variables.

Now, to traverse the hypothesis space $\mathcal{M}$, we have to compute all neighbours of the currently best hypothesis M^k. To do so, we employ refinement operators traditionally used in inductive logic programming. More precisely, for the language bias considered and the experiments conducted in present paper, we used the refinement operator $\rho : \mathcal{M} \mapsto 2^{\mathcal{M}}$ which selects a single clause

$\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \mathcal{M}$ and adds a minimal specialization $\mathrm{cl}' \equiv p : \mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B}'$ of cl to $\mathcal{M}$ (w.r.t. to θ-subsumption). Specializing a single abstract transition amounts to instantiating and to unifying variables, i.e., $\mathrm{cl}' \equiv \mathrm{cl}\,\theta$ for some substitution θ. When adding cl' to M^k, we have to ensure that (1) all possible observations and hidden state sequences are covered and (2) the list of bodies $\mathbf{B}'$ after applying $\rho(M)$ should remain well-founded. Condition (1) can only be violated if $\mathtt{B}' \not\in \mathbf{B}$. In this case, we add transitions with maximally general heads and observations. Condition (2) is established analogously. We complete the body lattice by adding new bodies (and therefore abstract transitions) in a similar way as described above. Both conditions together guarantee that the most specific body corresponding to a state is always unique.

5.2 Scoring Hypotheses

In principle, any maximum log-likelihood (ML) parameter estimation algorithm can be used to score hypotheses. However, in the presence of hidden variables ML estimation is a numerical optimization problem, and all known algorithms involve nonlinear, iterative optimization and multiple calls to an inference algorithm as subroutines. The most common ML parameter estimation technique for hidden Markov models is the Baum-Welch algorithm (see Appendix B for an adaptation) which is an instance of the Expectation-Maximization (EM) algorithm.

The EM algorithm [5] is a classical approach to maximum likelihood estimation in the presence of missing values. The basic observation underlying the EM algorithm is: we can easily find the ML parameters of a fixed model structure M^k if the data is complete, i.e., if we know the values for all the random variables. Therefore, it performs in each iteration $l+1$ two steps:

(E-Step) Based on the current parameters $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$, the algorithm computes a distribution over all possible completions of each partially observed data case.

(M-Step) Each completion is then treated as a fully observed data case weighted by its probability. New parameters $\boldsymbol{\lambda}^{k,l+1}$ are computed.

More formally, the E-step consists of computing the expectation of the log-likelihood given the old model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$, i.e.,

$$Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) = E\left[\log P(\mathbf{O}, \mathbf{H} \mid M^k, \boldsymbol{\lambda}) \mid M^k, \boldsymbol{\lambda}^{k,l}\right] . \qquad (4)$$

Here, $\mathbf{O}, \mathbf{H}$ denotes the completion of the data cases $\mathbf{O}$. The current model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$ give us the conditional distribution governing the unobserved states $\mathbf{H}$. The expression $E[\cdot|\cdot]$ denotes the expectation over this conditional distribution. The function Q is called the *expected score*. In the M-step, the expected score $Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$ is maximized w.r.t. $\boldsymbol{\lambda}$, i.e.,

$$\boldsymbol{\lambda}^{k,l+1} = \mathrm{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda} | M^k, \boldsymbol{\lambda}^{k,l}) . \qquad (5)$$

In fact, it is not necessary to maximize the expected score in each iteration. It is sufficient to choose $(M^k, \boldsymbol{\lambda}^{k,l+1})$ such that

$Q(M^k, \boldsymbol{\lambda}^{k,l+1} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$. Such an algorithm is called *generalized EM*. As Dempster et al. [5] have shown, the (generalized) EM algorithm improves the objective score in each iteration.

Theorem 2. *If* $Q(M^k, \boldsymbol{\lambda}^{k,l+1} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $score_{\mathbf{O}}(M^k, \boldsymbol{\lambda}^{k,l+1}) > score_{\mathbf{O}}(M^k, \boldsymbol{\lambda}^{k,l})$ *holds.*

Thus, if we choose in each iteration $(M^k, \boldsymbol{\lambda}^{k,l+1})$ that has a higher expected score than $(M^k, \boldsymbol{\lambda}^{k,l})$, we are bound to improve the objective score, see also [20]. The above naïve greedy algorithm can easily be instantiated to use the EM algorithm to estimate the ML parameters.

The problem with the naïve learning algorithm are its huge computational costs. In each structural iteration k, we evaluate all neighbours $M^{k'}$. For each neighbour, we need to run the EM algorithm for a reasonable number of iterations in order to get a reliable ML estimate of the parameters $\boldsymbol{\lambda}^{k'}$. Each EM iteration requires a full LOHMM inference on all given data cases. In total, the run time complexity per structural iteration is at least $\mathcal{O}(m \cdot \textit{EM iterations} \cdot \textit{costs of LOHMM inference})$ where *costs of LOHMM inference* depends on the length of data sequences, too. Friedman proposed the *structural EM* (SEM) to reduce the computational complexity [9, 10]. In the next section, we will show how to adapt the ideas underlying SEM to learning LOHMMs. We will follow the notation used in [9].

6 Structural Generalized EM

The idea underlying SEM is as follows. We take our current model $(M^k, \boldsymbol{\lambda}^k)$ and run the EM algorithm for a while to get reasonably completed data. We then fix the completed data cases and use them to compute the ML parameters $\boldsymbol{\lambda}^{k'}$ of each neighbour $M^{k'}$. We choose the neighbour with the best improvement of the score as $(M^{k+1}, \boldsymbol{\lambda}^{k+1})$ and iterate. More formally, we have

```
1:  Initialize $\boldsymbol{\lambda}^{0,0}$ randomly
2:  **Loop** for $k = 0, 1, 2, \ldots$
3:    **Loop** for $l = 0, 1, 2, \ldots$
4:      Let $\boldsymbol{\lambda}^{k,l+1} = \text{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
5:    **Until** convergence **or** $l = l_{\max}$
6:    **Find** model $M^{k+1} \in \rho(M^k)$ that maximizes $Q(\cdot \mid M^k, \boldsymbol{\lambda}^{k,l})$
7:    Let $\boldsymbol{\lambda}^{k+1,0} = \text{argmax}_{\boldsymbol{\lambda}}\, Q(M^{k+1}, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
8:  **Until** convergence
```

The hypotheses space is traversed as described in Section 5.1, and again we stop if there is no improvement in score. The following theorem shows that even when the structure changes in between, improving the expected score Q always improves the log-likelihood as well.

Theorem 3. *If* $Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) > \log P(\mathbf{O} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds.*

The proof is a simple extension of the argumentation by [20, Section 3.2 ff.]. However, to apply this algorithm to learning LOHMMs, we still need to show how to choose the best neighbour, cf. line 6.

Let us first simplify the expected score in line 6. To do so, let $c(\mathtt{b}, \mathtt{h}, \mathtt{o})$ denote the counts, i.e., the number of times the systems proceeds from ground state b to ground state h emitting ground observation o. Now, it holds that

$$
\begin{aligned}
Q(M, \boldsymbol{\lambda} | M^k, \boldsymbol{\lambda}^{k,l}) &= E\left[\log P(\mathbf{O}, \mathbf{H} | M, \boldsymbol{\lambda}) \middle| M^k, \boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\log \prod_t P(\mathtt{h_{t+1}}, \mathtt{o_{t+1}} | \mathtt{h_t}, M, \boldsymbol{\lambda}) \middle| M^k, \boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\log \prod_{\mathtt{b},\mathtt{h},\mathtt{o}} P(\mathtt{h}, \mathtt{o} | \mathtt{b}, M, \boldsymbol{\lambda})^{c(\mathtt{b},\mathtt{h},\mathtt{o})} \middle| M^k, \boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} c(\mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot \log P(\mathtt{h}, \mathtt{o} | \mathtt{b}, M, \boldsymbol{\lambda}) \middle| M^k, \boldsymbol{\lambda}^{k,l}\right] \\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \underbrace{E\left[c(\mathtt{b}, \mathtt{h}, \mathtt{o}) \middle| M^k, \boldsymbol{\lambda}^{k,l}\right]}_{=: ec(\mathtt{b},\mathtt{h},\mathtt{o})} \cdot \log P(\mathtt{h}, \mathtt{o} | \mathtt{b}, M, \boldsymbol{\lambda}) \,. \qquad (6)
\end{aligned}
$$

The term $ec(\mathtt{b}, \mathtt{h}, \mathtt{o})$ in (6) denotes the expected counts of making a transition from ground state b to ground state h emitting ground observation o. The expectation is taken according to $(M^k, \boldsymbol{\lambda}^{k,l})$. Of course, it is sufficient to consider the expected counts of only those tuples $(\mathtt{b}, \mathtt{h}, \mathtt{o})$ which are possible given the data. All other tuples have some default expected counts, e.g. 0. The expected counts can be computed using the adapted Baum-Welch algorithm, see Appendix B.

In order to improve the parameters given the expected counts, we apply a gradient-based optimization technique to account for the subliminal nondeterminism LOHMMs possesses. Multiple abstract transitions with the same body can match a given tuple $(\mathtt{b}, \mathtt{h}, \mathtt{o})$ so that an analytical solution of the M-step seems to be difficult. In principle, a gradient-based optimization technique iteratively performs the following two steps. First, it computes the *gradient* vector $\nabla_{\boldsymbol{\lambda}}$ of partial derivatives of the scoring function w.r.t. the parameters of a LOHMM at a given point. Then, it takes a step in the direction of the gradient to the point $\boldsymbol{\lambda} + \delta\nabla_{\boldsymbol{\lambda}}$ where δ is the step-size. Thus, we have to compute the gradient vector. For LOHMMs, the gradient vector consists of partial derivatives w.r.t. abstract transition probabilities and to selection probabilities.

Assume that λ is the transition probability associated with some abstract transition cl. Now, the partial derivative of (6) w.r.t. some parameter λ is

$$
\begin{aligned}
\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda} &= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} ec(\mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot \frac{\partial \log P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda} \\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \frac{ec(\mathtt{b}, \mathtt{h}, \mathtt{o})}{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \frac{\partial P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda} \qquad (7)
\end{aligned}
$$

The partial derivative of $P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})$ w.r.t. λ can be computed as follows [4]:

$$\begin{aligned}
&\frac{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial\lambda} = \\
&= \frac{\partial}{\partial\lambda} \sum\nolimits_{\mathrm{cl}} P(\mathrm{cl} \mid M, \boldsymbol{\lambda}) \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \\
&= \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \qquad (8)
\end{aligned}$$

Substituting (8) back into (7) yields

$$\begin{aligned}
&\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial\lambda} \\
&= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Big(\frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \Big) \qquad (9)
\end{aligned}$$

The selection probability follows a naïve Bayes approach. Therefore, one can show in a similar way as for transition probabilities that

$$\begin{aligned}
&\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial\lambda} \\
&= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Big(\frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \sum_{\mathrm{cl}} c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot P(\mathrm{cl} \mid M, \boldsymbol{\lambda}) \cdot \qquad (10) \\
&\qquad \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \Big) \qquad (11)
\end{aligned}$$

where $c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o})$ is the number of times that the domain element associated with λ is selected to ground cl w.r.t. $\mathtt{h}$ and $\mathtt{o}$.

Note that in the problem at hand, the described method has to be modified to take into account that the parameter vector $\boldsymbol{\lambda}$ consists of probability values, i.e. $\lambda \in [0,1]$ and that corresponding parameters sum to 1.0. There are two ways to enforce this: (1) projecting the gradient onto the constraint surface, and (2) reparameterizing the problem so that the new parameters automatically respect the constraints on $\boldsymbol{\lambda}$ no matter what their values are. We choose the latter approach as the reparameterized problem is fully unconstrained. More precisely, we define the parameters $\beta_{ij} \in \mathbb{R}$ for a set of corresponding λ's such that $\lambda_{ij} = (\beta_{ij}^2)/(\sum_k \beta_{ik}^2)$. This enforces the constraints given above, and a local maximum w.r.t. $\boldsymbol{\beta}$ is also a local maximum w.r.t. $\boldsymbol{\lambda}$, and vice versa. The gradient w.r.t the β_{ij}'s can be found by computing the gradient w.r.t the λ_{ij}'s and then deriving the gradient w.r.t. $\boldsymbol{\beta}$ using the chain rule, see e.g. [2, 15].

What do we gain from the structural GEM over the original naïve learning algorithm? We save many runs of the EM algorithm because we use the same ground counts for scoring all neighbouring hypotheses. Each iteration of the

[4] For the sake of simplicity, we will not check in the following equations that a transition is *maximally specific* for some ground states as done in the algorithms presented in the appendix.

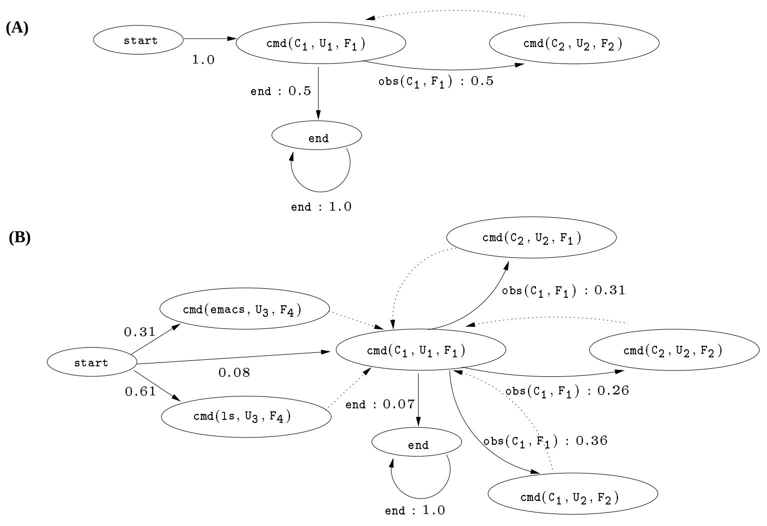


Fig. 2. The initial logical hidden Markov model (A) and the learned one (B).

gradient algorithm is computationally less expensive than one EM iteration. Essentially, considering the evaluation of the neighbours, we have made the run time complexity independent of the number and length of the data cases — a feature which is important for scaling up. However, maximizing the expected score (6) based on a different model structure does not only maximize likelihood but also tries to minimize the difference in the distribution of the hidden states sequences between the old and the new hypothesis. The SEM and SGEM learning schemes thus prefer changes in the model structure that do not change the role of the hidden states to those that do. Nevertheless, we believe that the expected score is a good heuristic in learning.

7 Experimental Prospects

The proposed method is intended as a generic framework for learning logical hidden Markov models. As such, it leaves several issues open. These include: the actual language bias (and corresponding refinement operator), the scoring function, possible pruning methods, etc. Nevertheless, in order to show the validity of our framework, we report on some experiments that proves that the underlying principles work. To this aim, we made a simple implementation in Sicstus Prolog 3.9.0. The implementation features a beam-search using the simple penalized log-likelihood (see Section 4) of the data as score. The chosen syntactic language bias did not allow for functors to be used. To perform the maximum likelihood estimation, we implemented the Baum-Welch algorithm as described in Appendix B (but scaled to avoid numerical imprecision). For the improvement of expected score, we adapted the scaled conjugate gradient (SCG) as implemented in Bishop and Nabney's Netlab library (`http://www.ncrg.aston.ac.uk/netlab/`,

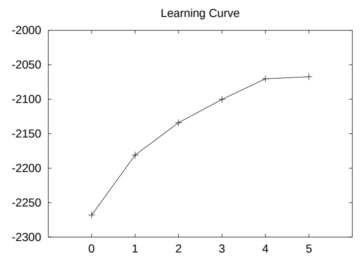


Fig. 3. Iteration (x axis) versus the penalized log-likelihood (y axis) achieved.

see also [3]). SCGs are well-known from the field of learning neural networks and avoid the time consuming line search of more traditional gradient-based methods by employing an approximation of the Hessian of the scoring function to quadratically extrapolate the minimum instead of doing a line search. The number of expected score evaluations is at most twice per iteration. We set $l_{\max} = 10$. Within the gradient-based optimization, we stopped when a limit of 10 iterations was exceeded or a change in penalized log-likelihood (reps. expected score) was less than 10^{-2} from one iteration to the next.

Data were generated from a LOHMM inspired by the user-modelling LOHMM, cf. Figure 1. Instead of modelling each command by its own predicate, we used the predicates `cmd(Command, User, Filename)` (to model internal states) and `obs(Command, Filename)` (to model the observations). There were 2 filenames, 2 users types, and 3 commands. From this model, we sampled 100 sequences of length 20. We selected the LOHMM shown in Figure 2 (A) as initial hypothesis. The learned model is shown in Figure 2. The learning curve in Figure 3 clearly proves the principle: the objective score increases in each iteration. The learned model is interesting for three reasons. First, it has less parameters than a a fully connected HMM modelling this domain. The latter consists of 156 parameters whereas the learned LOHMM consists of 20 parameters only. Second, the algorithm managed to adjust the starting probability distribution (`start` node). Third and more importantly, it unifies variables to model the distribution more accurately. For example, sequences of commands (even more than two) using the same filename are ran. Such knowledge cannot easily be read off from traditional hidden Markov models.

8 Related Work

LOHMMs can be analyzed from two different directions. On the one hand, they are related to several extensions of HMMs that have been investigated, such as hierarchical HMMs [7], factorial HMMs [12], and HMMs based on tree automata [8]. On the other hand, they are also related to the recent interest in combining inductive logic programming [23] principles with probability theory, see [4] for an overview.

In the first type of approach, the underlying idea is to decompose the state variables into smaller units, i.e. to *upgrade* HMMs to represent more structured state spaces. In hierarchical HMMs states themselves can be HMMs, in factorial HMMs they can be factored into k state variables which depend on one another only through the observation[5], and in tree based HMMs the represented probability distributions are defined over tree structures. The key difference with LOHMMs is that these approaches do not employ the logical concept of unification. Unification is essential because it allows us to introduce abstract transitions, which do not *consist* of more detailed states. Thus, structure learning algorithm devised for these approaches cannot directly applied to LOHMMs.

In the second type of approach, most attention has been devoted to developing highly expressive formalisms, such as e.g. stochastic logic programs [22], PRISM [27], probabilisitic relational models [11], and Bayesian logic programs [14]. LOHMMs can be seen as an attempt towards *downgrading* such highly expressive frameworks. As a consequence, LOHMMs represent an interesting position on the expressiveness scale. Whereas they retain most of the essential logical features of the more expressive formalisms, they seem easier to understand, adapt and learn. This is akin to many contemporary considerations in *inductive logic programming* [23] and multi-relational data mining [6]. In any case, to the best of our knowledge no structure learning algorithm based on the *structural EM* as been proposed.

LOHMMs are most closely related to the recently introduced *relational Markov models (RMMs)* [1]. Here, states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured. The main differences are that RMMs do not facilitate variable binding and unification nor hidden states. Therefore, they are more a first type than a second type approach. Furthermore, they do not select the most-specific transition to resolve conflicting transitions. Instead, they interpolate between conflicting ones. This is an interesting option for LOHMMs because it makes parameter estimation more robust. On the other hand, it also seems to make it more difficult to adhere one of our design princples: locally interpretable transitions. Structure learning has been addressed for RMMs based on *probability estimation trees* [25], a kind of decision trees.

Finally, our learning approach is to some extend contrary to some more advanced technique for learning the structure of hidden Markov models, namely model merging, see e.g. [28]. Here, the induction process starts with the most specific model consistent with the training data and generalizes by successively merging states. However, abstract transitions aim at good generalization, and the most general clauses can be considered to be the most informative ones.

[5] Factorial HMMs could be viewed as a special type of LOHMMs, where the hidden state is summarized by a $2 \cdot k$-ary abstract state. The first k arguments encode the k state variables, and the last k arguments serve as a memory of the previous joint state. The selection distribution of the i-th argument is conditioned on the $i + k$-th argument.

9 Conclusions

A method for learning logical hidden Markov models has been presented. The method essentially adapts Friedman's *structural EM* algorithm. More precisely, it combines standard *generalized EM*, which optimizes parameters, with structure search for model selection using refinement operators from the field of inductive logic programming. Experiments have been presented that show that the presented techniques work in principle.

As future work, we plan to conduct a more thorough experimental evaluation. The experiments were realized by relying on a simple and basic instantiation of our framework. Future work will be concerned with making more refined and state-of-the-art choices for many parameters. These include: the investigation of other objective scores, other refinement operators e.g. handling functors, deleting transitions, and generalizing hypotheses, logical pruning criteria for hypotheses, the incorporation of syntactic bias mechanisms, efficient storing of ground counts (e.g. using AD-trees [21]) etc. Nevertheless, the authors hope that the presented framework will inspire further research at the intersection of inductive logic programming and hidden Markov models.

References

1. C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In D. Hand, D. Keim, O. R. Zaïne, and R. Goebel, editors, *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, Edmonton, Canada, 2002. ACM Press.
2. J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive Probabilistic Networks with Hidden Variables. *Machine Learning*, 29(2–3):213–244, 1997.
3. C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford Univ. Press, 1995.
4. L. De Raedt and K. Kersting. Probabilistic Logic Learning. *SIGKDD Explorations: Special Issue on Multi-Relational Data Mining*, 5(1), 2003. (To appear).
5. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B 39:1–39, 1977.
6. S. Džeroski and N. Lavrač. *Relational Data Mining.* Springer-Verlag, 2001.
7. S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: analysis and applications. *Machine Learning*, 32:41–62, 1998.
8. P. Frasconi, G. Soda, and A. Vullo. Hidden Markov Models for Text Categorization in Multi-Page Documents. *Jour. of Intel. Information Systems*, 18:195–217, 2002.
9. N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In D. H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-1997), Nashville, Tennessee, USA, July 8-12, 1997*, pages 125–133. Morgan Kaufmann, 1997.
10. N. Friedman. The Bayesian structural EM algorithm. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 129–138, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

11. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pages 1300–1309, Stockholm, Sweden, 1999. Morgan Kaufmann.
12. Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.
13. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
14. K. Kersting and L. De Raedt. Bayesian Logic Programs. Technical Report 151, University of Freiburg, Institute for Computer Science, April 2001. (submitted).
15. K. Kersting and N. Landwehr. Scaled Conjugate Gradients for Maximum likelihood: An Empirical Comparison with the EM Algorithm. In J. A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02)*, pages 89–98, Cuenca, Spain, 2002. `http://www.info-ab.uclm.es/isl/cuenca/`.
16. K. Kersting, T. Raiko, and L. De Raedt. Logical Hidden Markov Models (Extended Abstract). In *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02)*, pages pp. 99–107, Cuenca, Spain, November 6–8 2002.
17. K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden Markov models. In R. B. Altman, A. K. Dunker, L. Hunter, T. A. Jung, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 192 – 203, Kauai, Hawaii, USA, 2003. World Scientific.
18. W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(4), 1994.
19. T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. In Åsa Rudström, editor, *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden, July 1999.
20. G. J. McKachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Eiley & Sons, Inc., 1997.
21. Andrew Moore and Mary Soon Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, March 1998.
22. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.
23. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
24. C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative Bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.
25. F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.
26. L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.
27. T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.
28. A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufman, 1993.

A Evaluation of LOHMMs

Consider the probability of the partial observation sequence $O = \mathtt{o}_1\mathtt{o}_2\dots\mathtt{o}_\mathtt{T}$ and (ground) state $\mathtt{h}$ at time t, given the model M, i.e. $\alpha_t(\mathtt{h}) := P(\mathtt{o}_1\mathtt{o}_2\dots\mathtt{o}_\mathtt{t}, q_t = \mathtt{h} \mid M)$ where $q_t = \mathtt{h}$ denotes that the system is in state $\mathtt{h}$ at time t. Clearly, $P(O \mid M) = \sum_{\mathtt{h}\in S_{T+1}} \alpha_{T+1}(\mathtt{h})$ where S_t denotes the set of reachable states at time t. As for traditional HMMs, $\alpha_t(\mathtt{h})$ can be computed using a dynamic programming approach. The only difference is that in LOHMMs the substitutions have to be taken into account.

```
1:  $S_0 := \{\mathtt{start}\}$        /* initialize the set of reachable states*/
2:  $\alpha_0(\mathtt{start}) := 1.0$    /* initialize the $\alpha$ value for start*/
3:  for $t = 1, 2, \dots, T+1$ do
4:    $S_t = \emptyset$              /* initialize the set of reachable states at clock $t$*/
5:    foreach $\mathtt{b} \in S_{t-1}$ do
6:      foreach maximally specific $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ such
                that $\sigma_\mathtt{b} = mgu(\mathtt{b}, \mathtt{B})$ exists do
7:        foreach $\mathtt{h} = \mathtt{H}\sigma_\mathtt{b}\sigma_\mathtt{h} \in G_\Sigma(\mathtt{H}\sigma_\mathtt{b})$ such
                  that $\mathtt{o}_\mathtt{t-1}$ unifies with $\mathtt{O}\sigma_\mathtt{b}\sigma_\mathtt{h}$ do
8:          if $\mathtt{h} \notin S_t$ then
9:            $S_t := S_t \cup \{\mathtt{h}\}$
10:           $\alpha_t(\mathtt{h}) := 0.0$
11:         $\alpha_t(\mathtt{h}) := \alpha_t(\mathtt{h}) + \alpha_{t-1}(\mathtt{b}) \cdot p \cdot \boxed{\mu(\mathtt{h} \mid \mathtt{H}\sigma_\mathtt{b}) \cdot \mu(\mathtt{o}_\mathtt{t-1} \mid \mathtt{O}\sigma_\mathtt{b}\sigma_\mathtt{h})}$
12: return $P(O \mid M) = \sum_{\mathtt{s}\in S_{T+1}} \alpha_{T+1}(\mathtt{s})$
```

The boxed terms constitute the main difference to the corresponding HMM formula. The computational complexity is $\mathcal{O}((T+1) \cdot s \cdot (|\Delta| + o \cdot g))$ where $s = \max_{t=1,2,\dots,T+1} |S_t|$, $|\Delta|$ is the number of abstract transitions, o is the maximal number of outgoing abstract transitions with regard to an abstract state, and g is the maximal number of ground instances of an abstract state. In a completely analogous manner, one can devise a *backward procedure* to compute $\beta_t(\mathtt{h}) = P(\mathtt{o}_{\mathtt{t+1}}\mathtt{o}_{\mathtt{t+2}}\dots\mathtt{o}_\mathtt{T} \mid q_t = \mathtt{h}, M)$. Based on $\alpha_t(\mathtt{h})$ and $\beta_t(\mathtt{h})$, we can devise an adaption of the Baum-Welch algorithm.

B The Baum-Welch Algorithm for LOHMMs

We have to estimate the maximum likelihood transition probabilities and selection distributions. To estimate the former ones, we upgrade the *Baum-Welsh* approach for HMMs. There, an improved estimate $\overline{p}$ of the transition probability of some (ground) transition $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ is computed by taking the ratio between the expected number $\xi(\mathrm{cl})$ of making the transitions cl at any time given the model M and an observation sequence O, and the total number of transitions starting from $\mathtt{B}$ at any time given the model M and an observation sequence O. Basically the same applies when cl is an abstract transition, however we have to be a little bit more careful because we have no direct access to $\xi(\mathrm{cl})$. To compute

$\xi_t(\mathrm{cl})$ in case that cl is abstract, let $\xi_t(\mathrm{gcl}, \mathrm{cl})$ be the probability of following the abstract transition cl via its ground instance $\mathrm{gcl} \equiv p : \mathtt{h} \xleftarrow{\mathtt{o}} \mathtt{b}$, i.e.

$$\xi_t(\mathrm{gcl}, \mathrm{cl}) = \frac{\alpha_t(\mathtt{b}) \cdot p \cdot \beta_{t+1}(\mathtt{h})}{P(O \mid M)} \cdot \boxed{\mu(\mathtt{h} \mid \mathtt{H}\sigma_\mathtt{b}) \cdot \mu(\mathtt{o_{t-1}} \mid \mathtt{O}\sigma_\mathtt{b}\sigma_\mathtt{h})} \tag{12}$$

where $\sigma_\mathtt{b}$, $\sigma_\mathtt{h}$, and $\sigma_\mathtt{o}$ are defined as before, and $P(O \mid M)$ is the probability that the model generated the sequence O. Again, the boxed terms constitute the main difference to the corresponding HMM formula. Now, it holds that $\xi_t(\mathrm{cl}) = \sum_{\mathrm{gcl}} \xi_t(\mathrm{gcl}, \mathrm{cl})$ where the sum runs over all ground instances of cl. This leads to the following reestimation formula where we assume that the sets S_t of reachable states are reused from the computations of the α- and β-values:

```
1:  /* initialization of expected counts */
2:  foreach cl ∈ $\Delta$ do
3:    $\xi(\mathrm{cl}) := 1$  /* or 0 if not using simple pseudocounts */
4:  /* compute expected counts */
5:  foreach $t = 0, 1, \dots, T$ do
4:    foreach $\mathtt{b} \in S_t$ do
5:      foreach maximally specific $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ such
                 that $\sigma_\mathtt{b} = \mathrm{mgu}(\mathtt{b}, \mathtt{B})$ exists do
6:        foreach $\mathtt{h} = \mathtt{H}\sigma_\mathtt{b}\sigma_\mathtt{h} \in G_\Sigma(\mathtt{H}\sigma_\mathtt{b})$ such
                 that $\mathtt{h} \in S_{t+1}$ and $\mathtt{o_{t-1}}$ unifies with $\mathtt{O}\sigma_\mathtt{b}\sigma_\mathtt{h}$ do
7:          $\xi(\mathrm{cl}) := \xi(\mathrm{cl}) + \boxed{\big(\alpha_t(\mathtt{b}) \cdot p \cdot \beta_{t+1}(\mathtt{h}) / P(O \mid M)\big)}$
                                      $\boxed{\cdot \mu(\mathtt{h} \mid \mathtt{H}\sigma_\mathtt{b}) \cdot \mu(\mathtt{o_{t-1}} \mid \mathtt{O}\sigma_\mathtt{b}\sigma_\mathtt{h})}$
11: /* estimate of new transition probabilities */
11: foreach $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ do
12:   $\gamma(\mathtt{B}) := 0$
13:   foreach $\mathrm{cl}' \equiv p' : \mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B} \in \Delta$ do
14:     $\gamma(\mathtt{B}) := \gamma(\mathtt{B}) + \xi(\mathrm{cl}')$
12:   $\overline{p} = \xi(\mathrm{cl}) / \gamma(\mathtt{B})$
```

where equation (12) can be found in line 7. To estimate the selection probabilities, recall that we used a naïve Bayes scheme to define the selection probability implicitly. Therefore, the estimated probability for a domain element $d \in D$ for some domain D is the ratio between the number of times d is selected and the number of times any $d' \in D$ is selected. The procedure for computing the ξ-values can mainly be reused.

Altogether, our EM scheme is: While not converged, (1) estimate the abstract transition probabilities and (2) the selection probabilities. Since our algorithm is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to [20], it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. The computational complexity of parameter reestimation (per iteration) is $\mathcal{O}(4 \cdot k \cdot \alpha + d)$ where k is the number of sequences, α is the complexity of computing the α-values (see above), and d is the sum over the sizes of the domains associated to the predicates.

A Simple Relational Classifier

Sofus A. Macskassy and Foster Provost

NYU Stern School of Business
44 W. 4th Street
New York, NY 10012, U.S.A.
{smacskas,fprovost}@stern.nyu.edu

Abstract. We analyze a Relational Neighbor (RN) classifier, a simple relational predictive model that predicts only based on class labels of related neighbors, using no learning and no inherent attributes. We show that it performs surprisingly well by comparing it to more complex models such as Probabilistic Relational Models and Relational Probability Trees on three data sets from published work. We argue that a simple model such as this should be used as a baseline to assess the performance of relational learners.

1 Motivation

In recent years, we have seen remarkable advances in algorithms for relational learning, especially statistically based algorithms. These algorithms have been developed in a wide variety of different research fields and problem settings. Relational data differ from traditional data in that they violate the instance-independence assumption. Instances can be related, or linked, in various ways. The label of an instance might depend on the instances it is related to either directly or through arbitrarily long chains of relations. This relational structure further complicates matters as it makes it harder, if not impossible, to separate the data cleanly into test and train sets without losing much relational information. Recent work has begun to investigate foundational issues within relational learning, such as the dimensions across which learners can be compared [11, 14, 25] as well as issues of link dependencies [13]. We broaden these investigations by describing a baseline method to which relational learners should be compared when assessing how well they have extracted a useful model from the given relational structure—beyond what can be achieved by looking only at known class labels of related neighbors.

Recent probabilistic relational learning algorithms—e.g., Probabilistic Relational Models (PRMs) [16, 10, 27], Relational Probability Trees (RPTs) [22] and Relational Bayesian Classifiers (RBCs) [23]—search the relational space for useful attributes and relational structure of neighbors (possibly more than one link away). While there are other relational learning algorithms available [7, 9, 6], we focus in this paper on the three named algorithms.

We know from classical machine learning that even very simple statistical methods such as naive Bayes can perform remarkably well even when compared to more complex methods. However, a question that has yet to receive much attention is how much of the performance of relational learners is due to their complexity and how much can be attributed to the relational structure of the data. In the latter case, even a simple model using no learning may perform quite well.

While results reported on the relational classifiers (PRMs, RPTs and RBCs) have been compared to non-relational baseline learners (e.g., the naive Bayes classifier [5, 15, 19] or C4.5 [26]), a simple relational classifier is an equally important, and perhaps a more appropriate, point of comparison. We analyze here the Relational Neighbor (RN) [25] classifier as such a simple classifier which uses only class labels of known related instances and does no learning. We show that it performs competitively to these related learning algorithms when compared against their published results on three different relational data sets. Although we believe that the complex methods can add value, we argue that in order to assess how well they have learned from relations, they should be compared against baseline methods such as RN.

The rest of the paper is outlined as follows. We first describe a simple relational neighbor (RN) classifier, followed by three studies comparing to reported results using the PRM, RPT and RBC relational learners. We then propose a probabilistic version of the RN and show that unexpectedly it does not add value in the cases outlined in this paper, though it can do so in other domains. We describe and report results on one such domain, and conclude with final remarks.

2 A Relational Neighbor Classifier

The Relational Neighbor (RN) classifier estimates class probabilities solely based on entities of the same type whose class labels are known.[1] The classifier works by making two strong, yet often reasonable, assumptions: (1) some entities' class labels are known within the same linked structure (see [25] for more discussion), and (2) the entities exhibit homophily—entities related to each other are similar and likely belong to the same class along one or more dimensions [2, 18]. The classifier may not perform well if entities are isolated or if no labels are known.

Definition. The relational-neighbor classifier estimates $P(c|e)$, the class-membership probability of an entity e belonging to class c, as the (weighted) proportion of entities in D_e that belong to class c. We define D_e as the set of entities that are linked to e. Thus,

$$P(c|e) = \frac{1}{Z} \sum_{\{e_j \in D_e | \text{label}(e_j)=c\}} w(e, e_j), \tag{1}$$

where $Z = \sum_{e_i \in D_e} w(e, e_i)$, and $w(e, e_i)$ is the weight of the link[2] between entities e and e_i. Entities in D_e that are not of the same type as e are ignored. If D_e is empty or has no entities with known class labels, then the RN will estimate e based on the class prior (of the known labels).

For example, consider a graph of linked web pages, each belong to a class (e.g., homepage vs. non–homepage). If we consider a link to be undirected (e.g., two pages

[1] We have based RN on the Relational Vector Space Model (RVSM) using the $s_{\text{wend}}(e, i)$ RVS scoring function [1].

[2] Note that the notion of "linked to" is domain dependent and, as we will show, different definitions can lead to (very) different performance.

are related if one links to the other, regardless of link directionality), then a RN classifier would classify a candidate page, p, as a homepage if the majority of pages related to p—either through being linked to p or linking to p—were known to be homepages.

However, we have a potential problem if all (or many) entities in D_e are unknown—we do not truly take the nature of relational data into account. For example, known information should propagate through the network to related instances. This idea of propagation has been used successfully in other work—e.g., iterative classification [21], relaxation labeling [3] and belief propagation [24], which is used in PRMs among others.

Definition. The iterative relational-neighbor classifier (RN*) iteratively classifies entities using the RN classifier in its inner loop. We define RN^i as the model at iteration i, where RN^0 defines what is initially known and RN^1 is equivalent to RN. At iteration i, RN^i uses the labels given by RN$^{(i-1)}$ to predict class-membership of currently unknown instances. In the case where the class-membership probability of a neighboring entity, $e_j \in D_e$, is a prediction from RN* and was not initially known, the class with the highest probability score is used. Thus, an entity e will be classified as `unknown` if the (weighted) majority is `unknown`.[3] For this paper, RN* stops when no unknown entities are left or when no new entities can be labeled (as could be the case when there are isolated components with no known labels).

3 Case Studies

We compare, in this section, the relational neighbor classifier to three different published results on relational learning. The question we ask in each of these case studies is whether we perform well enough to warrant the use of such a simple model as a baseline comparison for relational learning. To have a fair comparative study, we tried to replicate the original test environments closely.

3.1 CoRA

The first case study is based on the CoRA data set [17], which is a data set of academic papers within Computer Science. This corpus includes the full citation graph as well as labels for the topic of each paper (and potentially sub- and sub-sub-topics). We focused on 4240 papers within the machine learning topic with the classification task of predicting which of seven sub-topics a given paper belongs to [27]. We used all 4007 unique authors that we could identify in this subset. Thus, our graph differed from the original study [27] in which they report using 4187 papers and 1454 authors.

Papers can be linked in one of two ways, using a common author, or a citation. Thus, classification can be based on using only one of the given link types, or using both links between the two papers. We chose the latter, where we assign the 'weight' of a relation as the sum of the number of authors two papers have in common and the number of citations that link them to each other. This latter weight would ordinarily only be zero or one unless the two papers cite each other.

[3] In cases where too little propagation takes place, because of too much weight from unknown labels, the need for a majority of weight from a known class can be weakened. This was not necessary for the cases presented in this paper.

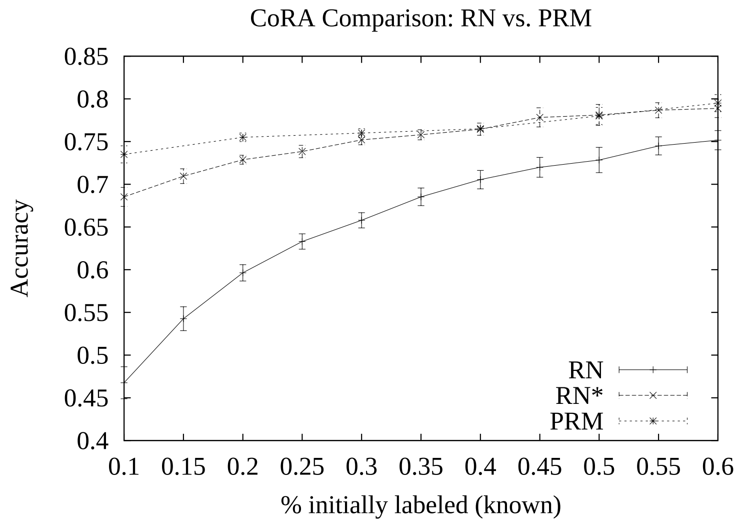


Fig. 1. Comparison of RN, RN* and PRM on the CoRA data set.

Using Probabilistic Relational Models (PRMs), the prior work reported accuracies ranging from 0.68 to 0.79, as the ratio of initially known labels increased from 10% to 60%. The best performance reported was a PRM that used both author and citation links as well as the words in the paper.

Using the same methodology as reported in the PRM study, we varied the proportion of papers for which the class is initially known from 10% to 60%. We varied in 5% increments; we performed a 10–fold cross-validation at each setting (the previous study performed a 5–fold cross-validation, varying the training set in 10% increments). Figure 1 compares the classification accuracy of RN with the reported results from the PRM, where the PRM used both citation links and author links.[4] Although RN only has an accuracy of 46% initially, it is able to reach relatively high performance once 50% of the papers are labeled, though it is still not as accurate as the PRM. Also shown in Figure 1, RN* estimates the classes of the unknown papers by repeatedly updating the class-probabilities of unknown nodes (using RN) until all nodes have been labeled. This took 3–5 iterations, decreasing as we increased the labels that were known initially. We see a marked improvement throughout the whole graph. RN* is competitive with the PRM, matching its performance once 35% of the labels are known initially.

We will only report on the RN* classifier in the next two studies due to its clear superiority.

[4] The PRM values were approximated from the graphs in the original paper.

3.2 IMDb

The second data set case study is based on the Internet Movie Database (IMDb)[5] data set, where we build models to predict movie box-office receipts [12]. We focus on movies released in the United States between 1996 and 2001 with the goal of predicting whether a movie "will be" a blockbuster (the opening weekend box-office receipts exceed $2 million) [22]. We used the database from the authors of the original study to extract "blockbuster" classifications. However, it was non–trivial to recreate the complete graph as described in the original work, which used 1364 movies (45% of those being blockbusters) [22]. We thus used a data set from the IMDb web-site. We identified 1441 movies released between 1996 and 2001 that we were able to link up with a "blockbuster" classification in the original database. 615 of the 1441 movies (42.6%) were classified as "blockbuster."

The IMDb data consists of movies, which have relations such as 'actor', 'director', 'producer', etc. Movies can be linked through these intermediate objects. RN can be based on a particular link type ($\mathrm{RN}_{<\mathrm{link\_type}>}$). Links between movies are through various other entities (actor, studios, production companies, etc.), and we consider the links to be typed by the entity through which they pass (e.g., $\mathrm{RN}_{\mathrm{producer}}$ means: how often does the producer produce blockbusters). Based on a suggestion from David Jensen, we consider four types of links: {`actor, director, producer, production company`[6]}.

Using Relational Probability Trees (RPTs) and Relational Bayesian Classifiers (RBCs), the prior work reported areas under the ROC curve (AUCs) of 0.82 and 0.85 for RPTs and RBCs, respectively, using a set of eight attributes on related entities, such as the most prevalent genre of the movie's studio.

We used a 10-fold cross-validation to generate predictions for all training examples. It is then straightforward to generate an ROC curve—using the class-membership probabilities produced by RN^*—and its AUC by pooling these predictions and sorting the prediction scores for the primary class ("blockbuster", in our case) [8]. In order to account for variance, we ran the 10-fold cross-validation 10 times, each time generating new folds. Table 1 shows the mean AUCs and their variances for the RN^* classifier on each of the four link types.

RN^* link-type	**AUC**	(variance)
actor	0.766	(0.003)
director	0.658	(0.007)
producer	0.850	(0.005)
prodco	0.862	(0.003)

Table 1. AUCs of RN^* using only 1 link type.

As is clear from Table 1, RN^* is very competitive, outperforming the RBC AUC score of 0.85. However, it may be possible to perform even better by considering more

[5] `http://www.imdb.com`

[6] We shorten 'production company' to 'prodco' when describing our results below.

than one link type as each link type obviously contribute some evidence of class-membership. Thus, $\text{RN}^*_{\text{actor+director}}$ would consider edges both along the `actor` as well as along the `director` links. To test this, we ran a simple forward feature-selection search: For each remaining (unused) link-type/feature, add it to the current best performer—starting with prodco, the best performer from Table 1—and keep the combination that reported the best performance. Keep adding one feature at a time until it stops improving the performance.[7] Using this methodology, we end up with the AUCs presented in Table 2.[8]

RN^* link-type(s)	**AUC**	(variance)
prod+prodco	0.884	(0.003)
dir+prod+prodco	0.885	(0.003)

Table 2. AUCs of RN^* using a forward-selection feature-based search to combine multiple link types.

We see that even with a relatively naive feature-based search, we were able to increase performance over that of using only one link type.

3.3 WebKB

The last case study we present is based on the data set collected by the WebKB Project [4].[9] It consists of a set of web pages from four computer science departments, with each page manually labeled into the categories: course, department, faculty, person, project, staff, student or other. This data set includes clearly defined `link-to` relations between pages. The classification task is to predict whether a page belongs to a student [22]. As with the prior study, we extracted the pages that have at least one incoming and one outgoing link, and kept remaining pages that either link to a page or are linked to by a page in this subset of pages. This resulted in a data set of 920 pages and 3036 background pages, giving us a total of 3956 pages. This differs from the prior work which had 910 extracted pages and a total of 3877 pages, including the background pages [20].

We create an edge between two pages if one page links to the other. We weight these edges by summing the number of links from one page to the other and vice versa. Thus, we do not take directionality into account, but treat these as generic links.

We performed a preliminary investigation, using the same 10-fold cross-validation methodology as described in the previous study—we used the 920 pages identified earlier to create the training folds, but allowed paths to any background page. This resulted

[7] The relational structure of the data complicates things, making it unclear what it means to use only the training set to perform the feature-selection. We circumvented this problem by using all the data in this study. These feature-selection results therefore might be optimistic.

[8] We also performed a brute-force analysis of all possible combinations of link types. For this study, the AUCs reported in Table 2 were the best two results among all possible combinations.

[9] We use the WebKB-ILP-98 data set.

in a low AUC score of 0.310 with a variance of (0.045)—worse even than random. However, when considering the relational structure of the domain, this is not so surprising. How often does a student link to another student? An observation in earlier work in this domain states that it is more likely that a student will link to her advisor or a group/project page rather than to her peers [4]. This would indicate that it is more likely that student pages would have intermediaries in common (e.g., they are likely to both link to their advisor, department or project page)—and that student pages are really 2 edges apart. Thus, we define "neighbors" for this domain as pages linked through some other page. The way we weigh these type of paths is to multiply the edge weights along this path (e.g., if a student page has 2 links to a group page, and a fellow student has 3 links to the same group page, then the weight along that path between those 2 students would be 6). This weight represents how many possible ways two pages could reach each other. Running the RN* classifier using the same cross-validation methodology as before resulted in a mean AUC of 0.948 with a variance of (0.003), a dramatic improvement in performance.

Using RPTs and RBCs, the prior work shows AUCs ranging from 0.716 to 1.0 for RPTs and 0.432 to 0.493 for RBCs [20]. The study used a set of ten attributes on related entities, such as the URL path and host, as well as structural attributes such as the number of in-links and out-links of each page. However, doing a direct comparison to this study is not possible. The results reported were based on a 4-fold cross-validation methodology in which one university is used as a holdout set while the remaining three are used for training. This methodology is obviously not appropriate for our simple classifier, as it needs to have direct links to known classes. Instead, the question we ask is how many instance labels we do need to know in order to perform comparably. Considering each university as a separate data set, we perform a similar study to that of the CoRA data set: we randomly pick $x\%$ of the pages and label them. The remaining labels are unknown. Running RN*, we can calculate the resulting AUC. We do this 10 times, each time randomly picking $x\%$ pages to label, giving us an average and standard deviation for the expected AUC for a given x. Doing this for $x \in \{\frac{1}{2}, 1, 5, 10, 15, 20, 25, 30, 40, \ldots, 90\}$, we graph the resulting AUC scores as x increases and compare these to the AUCs reported in the earlier study. Figure 2 shows, for each university, the resulting graphs.

Three immediate observations can be noted: in all cases the RN* was able to get close to its best performance even when given only 5% of the data. Second, in all cases, though less so on the Cornell data set, the RN* was competitive with RPT even having seen only 5% of the data. In fact, it was able to outperform RPT on the Washington data set, having seen only 5% of the data, and having seen 30% of the Wisconsin data made it perform on par with RPT. Third, in all cases, even when knowing the label of only 1 page (e.g., $x = \frac{1}{2}$), RN* was able to outperform RBC.

One important point that this comparison brings out is that even when a direct comparison is not possible, it is still possible to quantify the effectiveness of an algorithm by seeing how much data RN needs in order to perform comparably. In this case, having seen only 5% of the data yielded very close performance on 3 out of 4 data sets.

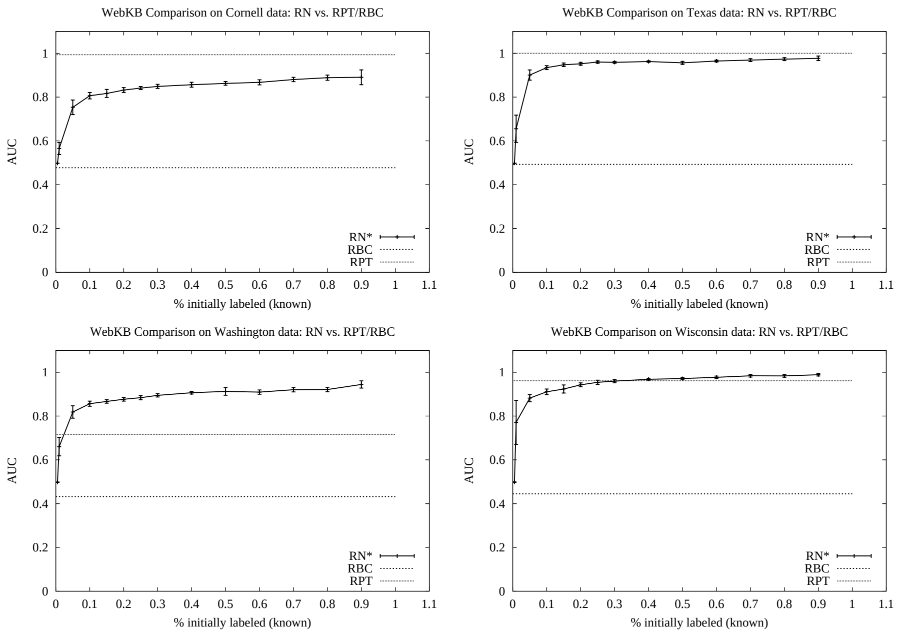


Fig. 2. Comparison of RN*, RPT and RBC on the four universities in the WebKB data set.

4 The Probabilistic Relational Neighbor Classifier

In Section 2 we described how RN generates class-membership probabilities based on known background entities in the set of neighbors, D_e. Intuitively it seems that even background entities whose labels are unknown should offer some evidence of class-membership. What if we could make some initial estimate of their class? Even the class prior might be useful, if no better initial estimation is available.

Definition. The probabilistic relational-neighbor classifier (pRN) estimates $P(c|e)$ as the (weighted) mean of the class-membership probabilities of the entities in D_e.

$$P(c|e) = \frac{1}{Z} \sum_{e_j \in D_e} w(e, e_j) * P(c|e_j), \tag{2}$$

where D_e, Z and $w(e, e_j)$ are defined as before. Background entities whose class labels are not known will be assigned the class priors.

What about an iterative version of pRN? As with RN, RN* seems sub-optimal—if we have properly estimated probabilities of class-membership, why not make use of them? Further, if an instance is unknown, why not give it some prior—such as the class priors from the known instances—and use that as the base to propagate? Might such a more evenly distributed scoring not perform better than the hard labeling of RN*?

Definition. The iterative probabilistic relational-neighbor classifier (pRN*) is similar to RN*, except it uses pRN in its inner loop and all initially unknown instances have their probabilities continuously updated. Unknown instances initially are assigned the class priors based on the initially known labels. Unlike RN*, pRN* updates class-probabilities of *all* initially unknown entities at every iteration. Because of the loopy nature of the propagation, there is no guarantee of convergence, though in all our test cases it the probabilities seems to be converging.[10] However, we need to set a maximum number of iterations as well as a convergence stopping criterion (e.g., based on how much the probabilities change from one iteration to the next).[11]

This probabilistic propagation is more satisfying intuitively as it takes into account probabilities, and can further be bootstrapped by assigning priors to unknown examples. These priors can simply be class priors, or could be estimated by other learning algorithms [21]. We ran pRN* on the three case studies, using the same methodology as reported above for RN*, to see how well this probabilistic version would stand up to the more simple class-propagating classifier.

Surprisingly, pRN* generally performed worse or—at best—only comparably to RN*. In virtually all tests, when only a small fraction ($\leq 30\%$)) of data was initially labeled, RN* performed better than pRN*, though they often had similar performance when we label $> 75\%$ of the data. In only two instances did pRN* outperform RN*. In the CoRA domain, while pRN* is initially worse, it does end up outperforming both RN* as well as the PRM. Figure 3 shows their comparative performances.

The other case where pRN* outperformed RN* was in the WebKB study using only directly neighboring pages, where it was able to achieve an AUC of 0.468 (variance of 0.018) whereas RN* only got an AUC of 0.310—though both are worse than random. However, when we went to a path length of 2, the two classifiers had equivalent performance (same mean AUC and variance).

5 pRN on Synthetic Data

Although pRN* did not perform as well as RN* on the previous data sets, there are cases where it should perform better. In particular, it seems likely that if there is a large class skew or if very few instances are known, then pRN would benefit from being able to propagate probabilities rather than class labels.

We test this hypothesis on a data set where there is a large class skew, where we varied the amount of initially labeled instances. We had access to a synthetic data set—created by others, and not with this study in mind—of people and their interactions, where people were categorized as good or bad depending on whether they belonged to any bad group. The data set consisted of 306 people (each belonging to one or more of 12 groups), 8 of whom belonged to the one bad group. We had $63,602$ interaction links between these people ($23,350$ links from $19,251$ 2-way communication events and $40,252$ from 3842 n-way communication events).

[10] For a simple case where there is no convergence, consider a two-class problems with two instances linked to each other, one having a prior of 1 for class A and the other having a prior of 1 for class B. In this case, they would just keep alternating their beliefs between A and B.

[11] In all our test cases, probabilities were still changing slowly even after 100 iterations, which was the maximum number of iterations we set. However, even a few iterations are generally enough to get estimates that seem to be converging.

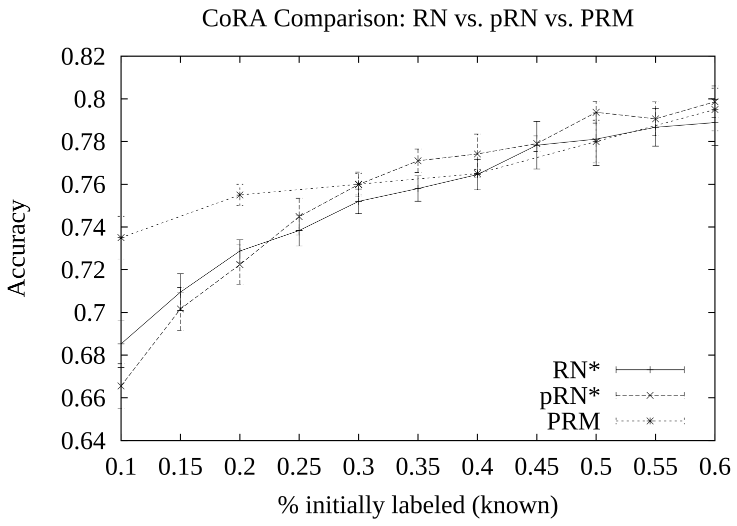


Fig. 3. CoRA Comparison

We have 2 dimensions along which we can test this data set—how many good people are initially known as well as how many bad people are known. We tested our classifiers labeling 0.1%, 1% and 5% of the good people, and labeling $0, .., 7$ of the bad people (in one extreme we do not know *any* bad person, and in the other extreme we know all but one of the bad people)—giving us a total of 24 possible test scenarios. Each test scenario was run 10 times, where we randomly sampled the appropriate number of good and bad people—we pooled the predictions of all runs for a given scenario and created an overall AUC score for that scenario.

Known	0.1% **good**		1% **good**		5% **good**	
# bad	**RN***	**pRN***	**RN***	**pRN***	**RN***	**pRN***
0	0.500	0.836	0.613	0.842	0.506	0.937
1	0.822	0.999	0.873	0.999	0.939	0.999
2	0.746	1.000	0.911	0.995	0.988	1.000
3	0.714	1.000	0.924	0.999	0.952	0.999
4	0.654	0.999	0.885	0.999	0.999	0.999
5	0.660	0.995	0.899	0.998	0.987	0.999
6	0.672	0.990	0.941	0.999	0.987	1.000
7	0.787	0.994	0.947	0.998	0.999	0.998

Table 3. AUCs of RN* and pRN* on a synthetic data set.

Table 3 shows the resulting AUCs for the RN and pRN classifier, where the columns are paired by how many good people are known, and the rows represent the number of bad people initially known. It is clear from these results that pRN* is able to use very little information (e.g., one bad guy and only one good guy—0.1% good) to virtually perfectly label the remaining bad guys. Even when no bad guys are known, pRN* is still able to perform very well. RN* has obvious problems if it doesn't have at least one labeled instance of each class. Further, it needs to have 5% of the good guys labeled before it can perform comparably to pRN*.

6 Final Remarks

We started by observing that although there recently has been much work in the area of relational learning, very little work had been done on creating baseline studies with relational structure in mind. In fact, published results generally compare against non-relational learners as the comparative baseline. We put forth a simple relational neighbor classifier as one potential baseline relational model, which uses no learning and considers nothing but known class labels of related entities.

We performed three comparative studies on reported results for three probabilistic relational learners on three relational data sets (CoRA, IMDb, and WebKB). In all three studies we were able to perform comparably to the relational learners using no learning—though we did use feature selection to improve on upon an already comparable performance for the IMDb data set. If we assume that the experimental designs were comparable, these results provide strong evidence that simple models should receive more attention. Specifically, we feel this makes a strong case for how information can be simply extracted and effectively used from class labels of instances in the immediate neighborhood of an unknown instance. It also makes a strong case for using such simple models for baseline comparisons to assess how well the more complex learners are able to learn from the relational data. One important point was raised in the WebKB study—what to do if the study uses a holdout set that is completely separate from the test set. In this case, we were still able to quantify the performance of the learners by identifying how much of the test set RN needed in order to perform comparably—in the WebKB data set, this turned out to be on the order of 5%.

We also proposed a probabilistic version of RN, though replicating the three case studies showed that it at best performed comparably to the non-probabilistic version, often performing quite worse. While this seemed to indicate that the probabilistic version was not as powerful, a test case on a synthetic data set showed that a probabilistic version does work better under certain conditions, such as a large class skew or having very few labels known initially. Further, a probabilistic version has the added benefit that it can easily take into account estimated probabilities—provided either by the class priors or other learning methods—of unknown instances. The use of learning methods to generate priors warrants further investigation.

Acknowledgments

Thanks to Claudia Perlich for many helpful discussions. David Jensen made many helpful suggestions, including pointing us to the WebKB data set. We thank Ben Taskar and

Andrew McCallum for providing us with versions of the Cora data set. Jennifer Neville was instrumental in our recreating the test environments and setup for the IMDb and WebKB studies. This work is sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-01-2-585. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

1. A. Bernstein, S. Clearwater, and F. Provost. The Relational Vector-space Model and Industry Classification. Working paper CDeR IS-03-02, Stern School of Business, New York University, 2003.
2. P. M. Blau. *Inequality and Heterogeneity: A Primitive Theory of Social Structure*. New York: Free Press, 1977.
3. S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*, 1998.
4. M. Craven, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and C. Y. Quek. Learning to Extract Symbolic Knowledge from the World Wide Web. In *15th Conference of the American Association for Artificial Intelligence*, 1998.
5. P. Domingos and M. Pazzani. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In *Proceedings of the 13th International Conference on Machine Learning*, pages 105–112, 1996.
6. S. Dzeroski and N. Lavrac. *Relational data mining*. Berlin; New York: Springer, 2001.
7. W. Emde and D. Wettschereck. Relational Instance-Based Learning. In L. Saitta, editor, *Proceedings 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.
8. T. Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Technical Report HPL-2003-4, HP Labs, 2003.
9. P. A. Flach and N. Lachiche. 1BC: A First-Order Bayesian Classifier. In S. Dzeroski and P. A. Flach, editors, *Ninth International Workshop on Inductive Logic Programming (ILP'99)*, volume 1634, pages 92–103. Springer-Verlag, June 1999.
10. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.
11. D. Jensen. Quantitative criteria to characterize kd-ml research for c-xnt. 1998.
12. D. Jensen and J. Neville. Data Mining in Social Networks. In *National Academy of Sciences workshop on Dynamic Social Network Modeling and Analysis*, 2002.
13. D. Jensen and J. Neville. Linkage and Autocorrelation Cause Feature Selection Bias in Relational Learning. In *Nineteenth International Conference on Machine Learning (ICML2002)*, 2002.
14. D. Jensen and J. Neville. Schemas and Models. In *Proceedings of the Workshop on Multi-Relational Data Mining (MRDM-2002)*, pages 56–70, 2002.
15. T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In D. H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann.

16. D. Koller and A. Pfeffer. Probabilistic Frame-Based Systems. In *AAAI/IAAI*, pages 580–587, 1998.
17. A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163, 2000.
18. M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27:415–444, 2001.
19. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
20. J. Neville. Personal communication, June 2003.
21. J. Neville and D. Jensen. Iterative Classification in Relational Data. In *AAAI Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.
22. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning Relational Probability Trees. Technical Report 02-55, Department of Computer Science, University of Massachusetts Amherst, 2002. Revised version February 2003.
23. J. Neville, D. Jensen, B. Gallagher, and R. Fairgrieve. Simple Estimators for Relational Bayesian Classifiers. Technical Report 03-04, Department of Computer Science, University of Massachusetts Amherst, 2003.
24. J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1998.
25. F. J. Provost, C. Perlich, and S. A. Macskassy. Relational Learning Problems and Simple Models. In *Proceedings of the Relational Learning Workshop at IJCAI-2003*, 2003.
26. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
27. B. Taskar, E. Segal, and D. Koller. Probabilistic Classification and Clustering in Relational Data. In *17th International Joint Conference on Artificial Intelligence*, pages 870–878, 2001.

Collective Classification with Relational Dependency Networks

Jennifer Neville and David Jensen

Department of Computer Science
140 Governors Drive
University of Massachusetts, Amherst
Amherst, MA 01003
{jneville|jensen}@cs.umass.edu

Collective classification models exploit the dependencies in a network of objects to improve predictions. For example, in a network of web pages, the topic of a page may depend on the topics of hyperlinked pages. A relational model capable of expressing and reasoning with such dependencies should achieve superior performance to relational models that ignore such dependencies. In this paper, we present relational dependency networks (RDNs), extending recent work in dependency networks to a relational setting. RDNs are a collective classification model that offers simple parameter estimation and efficient structure learning. On two real-world data sets, we compare RDNs to ordinary classification with relational probability trees and show that collective classification improves performance.

1 Introduction

In this paper, we show how *autocorrelation* can be used to improve the accuracy of statistical models of relational data. Autocorrelation is a statistical dependency between the values of the same variable on related entities and is a common characteristic of many relational data sets. In previous work, we have shown how autocorrelation can cause bias in learning algorithms (Jensen & Neville 2002), particularly when it appears with *concentrated linkage*, another common characteristic of relational data sets. However, this work explores methods which exploit autocorrelation to improve model accuracy. In particular, we demonstrate this can be accomplished with a relatively simple approach to structure learning in a class of undirected graphical models that we call relational dependency networks (RDNs).

It is relatively easy to understand how autocorrelation could be used to improve the predictions of statistical models. For example, consider the problem of automatically predicting the *topic* of a technical paper (e.g., neural networks, reinforcement learning, genetic algorithms). One simple method for predicting paper topics would look at the position of a paper within a citation graph. It may be possible to predict a given paper's topic with high accuracy based on the topics of neighboring papers in this graph. This is possible only because we expect high autocorrelation in the citation graph—papers tend to cite other papers with the same topic.

However, such a scheme for prediction assumes that all the labels of related entities (e.g., topics of referenced papers) are known. In many cases, the topics of an entire set of papers may need to be inferred simultaneously. This approach, called *collective classification* (Taskar, Abbeel & Koller 2002) requires both models and inference procedures that can use inferences about one entity in a relational data set to influence inferences about related entities. Similar approaches have been used in several recent applications (Neville & Jensen 2000; Chakrabarti, Dom & Indyk 1998).

In this paper, we introduce relational dependency networks (RDNs), an undirected graphical model for relational data. We show how RDNs can be learned and how RDNs and Gibbs sampling can be used for collective classification. Because they are undirected graphical models, RDNs can represent the cyclic dependencies required to express autocorrelation, and they can express a joint probability distribution, rather than only a single conditional distribution. In addition, they are relatively simple to learn and easy to understand.

We show preliminary results indicating that collective inference with RDNs offers improved performance over non-collective inference that we term "individual inference." We also show that RDNs applied collectively can perform near the theoretical ceiling achieved if all labels of neighbors are known with perfect accuracy. These results are very promising, indicating the potential utility of additional exploration of collective inference with RDNs.

2 Classification Models of Relational Data

Many relational models are used for "individual inference," where inferences about one instance are not used to change the inference of related instances. For example, some work in inductive logic programming (ILP) and relational learning uses relational instances that can be represented as disconnected subgraphs (e.g., molecules), thus removing the need (and the opportunity) for collective inference. Such models can cope with complex relational structure *of an instance*, but they do not attempt to model the relational structure *among instances*.

Even learning relational models for individual inference can be difficult because learning in relational data differs substantially from learning in propositional settings. For example, work in ILP has long considered difficult representational issues such as recursion, aggregation, variables, quantification, and other aspects of first-order and higher-order logics. In addition, some of our recent work has examined the unique statistical properties of relational data, and the influence of these characteristics on learning algorithms. We have shown how concentrated linkage and relational autocorrelation can bias relational learners toward particular features (Jensen & Neville 2002). We have also shown how degree disparity can cause misleading correlations in relational data, leading learning algorithms to add excessive structure to learned models (Jensen, Neville & Hay 2003).

To address the challenges of relational learning, new learning algorithms are necessary. For example, probabilistic relational models (PRMs) (Getoor, Friedman, Koller & Pfeffer 2001) are a form of directed graphical model thatextend Bayesian networks to support reasoning in complex relational domains. Unfortunately, PRMs are a directed model in which autocorrelation cannot be represented due to acyclicity constraints.[1]

As another example, we have recently developed relational probability trees (RPTs), a method for learning tree-structured models that encode conditional probability distributions for a class label that depend on both the attributes of related instances and the features of the surrounding structure (Neville, Jensen, Friedland & Hay 2003). Our methods for learning RPTs adjust for the sources of bias mentioned above. As a result, the learned RPTs are a relatively compact and parsimonious representation of conditional probability distributions in relational data. Until recently, we had only applied these models for individual inference. The algorithm for learning RPTs adjusts for statistical biases caused by autocorrelation, but our inference techniques have not made use of autocorrelation to improve inference.

This is unfortunate, because autocorrelation is a nearly ubiquitous phenomenon in relational data. We have observed relatively high levels of autocorrelation in relational data sets. For example, in analysis of the 2001 KDD Cup data we found that the proteins located in the same place in a cell (e.g., mitochondria or cell wall) had highly autocorrelated functions (e.g., transcription or cell growth). Autocorrelation has been identified by other investigators as well. For example, fraud in mobile phone networks has been found to be highly autocorrelated (Cortes, Pregibon & Volinsky 2001). The topics of authoritative web pages are highly autocorrelated when linked through directory pages that serve as "hubs" (Kleinberg 2001).

We also know that exploiting autocorrelation can result in significant increases in predictive accuracy. Several recent developments in relational learning have focused on exploiting autocorrelation. For example, the relational vector-space (RVS) model (Bernstein, Clearwater & Provost 2003) uses weighted adjacency vectors to construct classifiers. The model is extremely simple, but it produces accurate classifiers in data with strong autocorrelation. Other examples include work with web pages (Chakrabarti et al. 1998) that uses the hyperlink structure to produce smoothed estimates of class labels and our own prior work (Neville & Jensen 2000) that uses an iterative classification scheme to improve accuracy by exploiting the inferred class labels of related instances.

Recently, undirected graphical models capable of representing and reasoning with autocorrelation have been explored for the task of modeling relational data. Domingos and Richardson (2001) represent market entities as social networks and develop Markov random field models to model the influence in purchasing patterns throughout the network. Taskar et al. (2002) use relational Markov networks (RMNs), based on conditional random fields for sequence data (Lafferty, McCallum & Pereira 2001), to model the dependencies among web pages to predict page type. Undirected models have proved to be successful for collective classification of relational data. However, research into these models has focused primarily on parameter estimation and inference procedures. Model structure is not learned automatically, it is pre-specified by the user. Also, the models do not automatically identify which features are most relevant to the task. For relational tasks, which are likely to have a large

[1] An exception is where autocorrelation is structured by some additional information such as temporal constraints. (Getoor et al. 2001)

number of features, this lack of selectivity will make the model more difficult to interpret and understand.

3 Relational Dependency Networks

Relational dependency networks extend recent work on dependency networks (Heckerman, Chickering, Meek, Rounthwaite, and Kadie 2000) to a relational setting. Dependency networks (DNs) are graphical models of probabilistic relationships—an alternative to Bayesian networks and Markov random fields. DNs are easy to learn and have been shown to perform well for a number of propositional tasks so we expect them to offer similar advantages when used in a relational setting. We begin by reviewing the details of dependency networks for propositional data and then describe how to extend dependency networks for use with relational data.

3.1 Dependency Networks

Like Bayesian networks and Markov random fields, dependency networks encode probabilistic relationships among a set of variables. Dependency networks are an alternative form of graphical model that approximate the full joint distribution with a set of conditional distributions that are learned independently. DNs combine characteristics of both undirected and directed graphical models. The dependencies among variables are represented with an undirected graph, so conditional independence can be interpreted using graph separation. However, as with directed models, dependencies are quantified using conditional probability distributions (CPDs) of a variable given its parents. The primary distinction between DNs and other graphical models is that DNs are an *approximate* model. Because the CPDs are learned independently, DN models are not guaranteed to specify a coherent probability distribution (see *Learning* section below for details).

DNs offer several advantages over conventional Bayesian networks, but also have several disadvantages (Heckerman et al. 2000). Unlike Bayesian networks, DNs are difficult to construct using a knowledge-based approach and they cannot represent causal relationships. However, DN models can encode predictive relationships (i.e. dependence and independence) and there are simple techniques for parameter estimation and structure learning of DNs.

The characteristics that distinguish DN models from Bayesian networks make them similar to Markov random fields. Both DNs and Markov random fields use undirected graphs to represent dependencies among variables. When the causal relationships among variables are unknown, undirected models are often more interpretable than directed models which require d-separation reasoning to assess conditional independencies. Undirected graphical models also have straightforward techniques for parameter estimation and structure learning (e.g. Della Pietra, Della Pietra and Lafferty 1997). DN conditional probability distributions will generally be easier to inspect and understand than Markov network clique potentials, but DNs approximate the full joint distribution and therefore Markov networks may produce more accurate probabilities.

Representation. The DN model consists of a set of conditional probability distributions (CPDs), one for each variable given its parents. Consider the set of variables $X=(X_1,...,X_n)$ with a joint distribution $p(x)=p(x_1,...,x_n)$. A dependency network for X is represented by a graph G where each node in G corresponds to an X_i□X. The parents of node X_i, denoted pa_i, consist of the nodes in its *Markov blanket*: This specifies that, given its parents, a node is conditionally independent of the rest of the nodes in the graph:

$$p(x_i \mid pa_i) = p(x_i \mid \mathbf{x} \setminus x_i)$$

The undirected edges of the graph connect each node x_i to each of its parents (the nodes in pa_i). Furthermore, each node in the graph contains a local CPD, $p_i=p(x_i|pa_i)$. Together these CPDs specify the joint distribution over $\boldsymbol{X}$.

For example, the DN in figure 1 could be used to model the set of variables $\boldsymbol{X}=(X_1,X_2,X_3,X_4,X_5)$. Each node is conditionally independent of the other nodes in the graph given its parents. For example, X_1 is conditionally independent of X_2 and X_4 given X_3 and X_5. Each node contains a CPD, which specifies a probability distribution over possible values given the values of its parents. The full joint probability is the product of the local CPDs:

$$p(X) = p(X_1 \mid X_3,X_5)p(X_2 \mid X_3,X_4)p(X_3 \mid X_1,X_2)p(X_4 \mid X_2,X_3)p(X_5 \mid X_1)$$

Notice that the model may have cycles. For example, X_2, X_3 and X_4 form a clique in the graph. This necessitates the use of approximate inference techniques (see *Inference* section below for details). Also, notice that X_4 is dependent on X_3, but the reverse is not true. An undirected edge is placed between two nodes if either of the nodes is dependent on the other. In this case, a node's parents will still form a Markov blanket, but they won't be the minimal set (e.g. $p_3=p(x_3|x_1,x_2,x_4)= p(x_3|x_1 ,x_2))$.

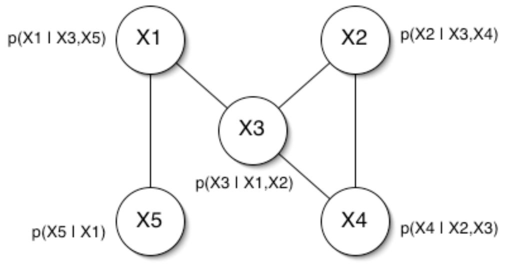


Fig. 1. Sample dependency network.

Learning. As with any graphical model, there are two components to learning a DN: structure learning and parameter estimation. Both structure learning and parameter estimation is accomplished through learning the local CPDs of each variable. The structure of a DN model is defined by the components of the local CPDs, much as feature specifications define the structure of a undirected graphical model. The edges of the graph connect a node to each of the variables in its local CPD. The parameters of the model correspond to the parameters of the local CPDs.

The DN learning algorithm learns a separate CPD for each variable, conditioned on the other variables in the data. For the DN model to be less than fully connected, a selective learning algorithm should be used. Any selective modeling technique can be used, however, to build a parsimonious DN model it is desirable to use a selective learner that will learn small, accurate CPDs.

For example, an algorithm for learning relational probability trees could be used to model X_1 given X_2,X_3,X_4,X_5. The variables included in the tree would then be designated as X_1's parents in the network, the appropriate edges would be added to the graph (e.g. X_1 to X_3 and X_1 to X_5) and the tree itself would be used as the local CPD for X_1. Learning the full DN in figure 1 would require learning five models, one tree for each variable in the graph.

Although the DN approach to structure learning is simple, it can result in an inconsistent network, both structurally and numerically—there may be no joint distribution from which each of the CPDs can be obtained using the rules of probability. Learning each local CPD independently may result in an inconsistent network where there is an edge between two variables but one is not dependent on the other (e.g. X_4 is dependent on X_3 but not vice versa). Independent parameter estimation may also result in inconsistencies where the overall joint distribution does not sum to 1. However, Heckerman et al. (2000) show that DNs will be "nearly" consistent if learned from large data sets. That is, the data serve a coordinating function that ensures some degree of consistency among the CPDs.

Inference. Unlike Bayesian networks, the dependency network graphs are potentially cyclic. This is due to the nature of the structure-learning algorithm where there is no restriction on the form of the CPDs. Cyclic dependencies necessitate the use of approximate inference techniques such as Gibbs sampling (e.g. Neal 1993) or loopy belief propagation (e.g. Murphy, Weiss and Jordan 1999). Heckerman et al. use Gibbs sampling to combine the models to approximate the full joint distribution and extract probabilities of interest. In practice, DNs have produced good approximations to the joint distributions represented by directed graphical models (Heckerman et al. 2000).

3.2 Relational Dependency Networks

Relational dependency networks (RDNs) extend DNs to work with relational data. The extension is similar to the way in which probabilistic relational models (Getoor et al. 2001) extend Bayesian networks for relational data.

Representation. RDNs specify a probability model over a network of instances. Given a set of objects and the links between them, a RDN defines a full joint probability distribution over the attribute values of the objects. Attributes of an object can depend probabilistically on other attributes of the object, as well as on attributes of objects in its relational neighborhood.

Instead of defining the dependency structure over attributes, as in DNs, RDNs define a generic dependency structure at the level of object *types*. Each attribute A_i associated with object type X is linked to a set of parents that influence the value of

A_i. Parents of A_i are either (1) other attributes associated with type X, or (2) attributes associated with objects of type Y where objects Y are linked to objects X. For the latter type of dependency, if the relation between X and Y is one-to-many, the "parent" consists of a set of attribute values. In this situation, RDNs uses aggregated features to map sets of values into single values.

For example, Figure 2 contains an example dataset from the movie domain. There are three types of objects—movies, studios and actors. Each object type has a number of associated attributes that will be each be modeled in the RDN. Consider the attribute *actor.hasAward*—the set of potential parents for this attribute consists of *actor.gender*, *movie.receipts* and *studio.country*. Notice that an actor can star in multiple movies and thus be associated indirectly with multiple studios.

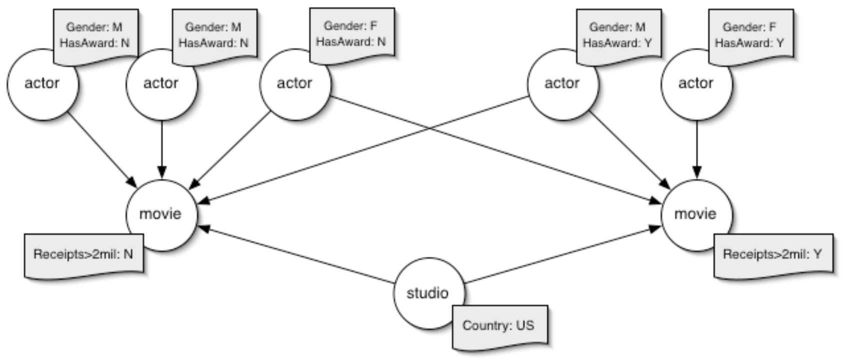


Fig. 2. Sample relational data graph.

The full RDN model is potentially much larger than the original data graph. To model the full joint distribution there must be a separate node (and CPD) for each attribute value in the data graph. Consequently, the total number of nodes in the model graph will be $\sum_T N_T A_T$ where N_T is the number of objects of type T in the data graph and A_T is the number of attributes for objects of that type. To make the models tractable, the structure and parameters of the CPDs are tied—the RDN model contains a single CPD template for each attribute of each type of object. For the example above, the model would consist of four CPDs, one for *actor.gender*, *actor.hasAward*, *movie.receipts* and *studio.country*.

To construct the model graph, the set of template CPDs is *rolled-out* over the entire data graph. Each object-attribute pair gets a separate, local copy of the appropriate CPD. This facilitates generalization across data graphs of varying size. We can learn the CPD templates from one data graph and apply the model to a second data graph with a different number of objects by rolling-out more CPD copies. This approach is analogous to other graphical models that tie distributions across the network (e.g. hidden Markov models, PRMs).

Figure 3 contains a possible RDN model graph for the example discussed above. It shows dependencies between *actor.gender* and *actor.hasAward*, between *actor.hasAward* and *movie.receipts*, and between *movie.receipts* and *studio.country*. Furthermore, there is a dependency between *movie.receipts* of related movies. Notice the hyper-edges between movies and associated actor awards. This indicates that movie receipts is dependent on an aggregated feature of *actor.hasAward* (e.g. EXISTS(*actor.hasAward*=Y)). Aggregation is one approach to ensure the template CPDs are applicable across data graphs of varying structure. Each movie may have a different number of actor award values, so aggregation is used to map these sets of values into single values.

Learning. Learning an RDN model again consists of two tasks: learning the dependency structure, and estimating the parameters of the conditional probability distributions. The RDN learning algorithm is much like the DN learning algorithm, except we use a selective *relational* classification algorithm to learn a set of conditional models. We use relational probability trees (RPTs) for the CPD components of the RDN. RPTs extend standard probability estimation trees to a relational setting in which data instances are heterogeneous and interdependent (Neville et al. 2003). RPT models estimate probability distributions over possible class label values in the same manner as conventional classification trees, but the algorithm looks beyond the attributes of the item for which the class label is defined and considers the effect of the local relational neighborhood on the probability distribution. RPT models represent a series of questions to ask about an item and the objects/links in its relational neighborhood.

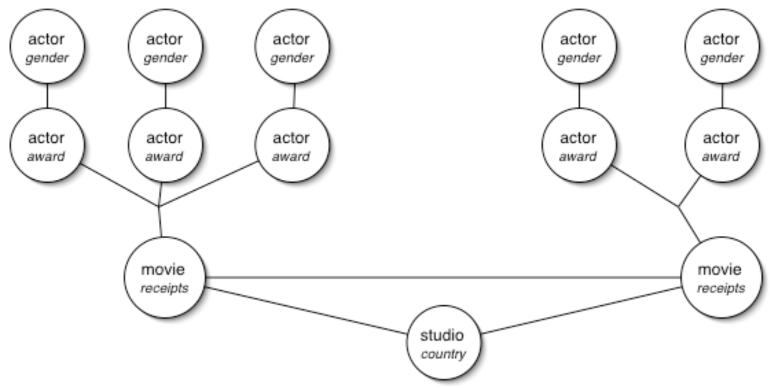


Fig. 3. Sample RDN model graph.

The RPT learning algorithm (Neville et al. 2003) uses a novel form of randomization tests to adjust for biases towards particular features due to the characteristics of the relational data (e.g. degree disparity, autocorrelation). We have shown that RPTs learned with randomization tests, build significantly smaller trees than other models and achieve equivalent, or better, performance. This characteristic of the RPTs is crucial for learning parsimonious RDN models—the collection of RPT models will be used during Gibbs sampling so the size of the models will have a direct impact on inference efficiency.

Given a data graph, an object type and a target attribute—an RPT is learned to predict the attribute given (1) the other attributes of the object, and (2) the attributes of other objects and links in the relational neighborhood. In our current approach, the user specifies the size of relational neighborhood to be considered by the RPT algorithm in (2). For efficiency reasons, we've limited the algorithm to consider attributes of objects one or two links away in the data graph. However, it is possible for the RPT models to consider attributes of much more "distant" objects (in the sense of a graph neighborhood).

Figure 4 shows an example RPT learned on data from the IMDb to predict whether a movie's opening-weekend receipts are more than $2million, given the attributes of movies and everything up to two links away—actors, directors, producers and studios, as well as movies associated with those objects (see Section 4 for experimental details). The tree indicates that movie *receipts* depend on the receipts of other movies made by the same studio, as well as actor age and movie genre. The root node of this tree asks whether more than 60% of the other movies made by the studio (e.g. *Studio Movie* objects) have *receipts=Y*. If this is not the case, the model moves to the next node on the right branch and asks whether the movie has more than 5 actors born after 1958. If the answer to this question is also no, a prediction of *P(Y)*=0.01 is returned.

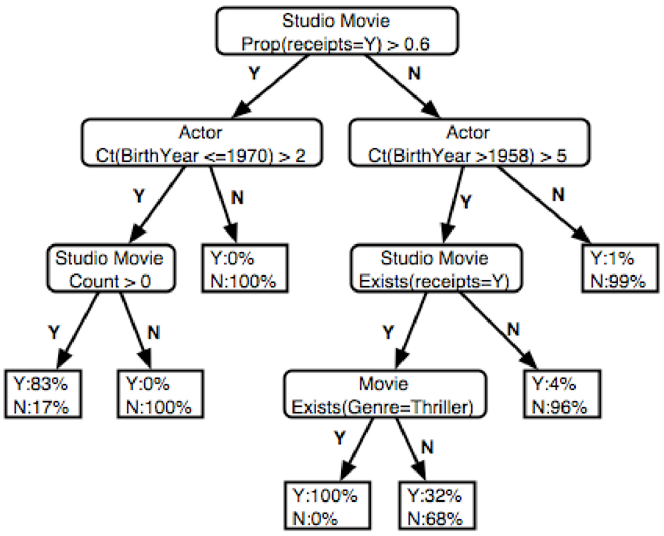


Fig. 4. Example RPT learned for the IMDb dataset to predict movie *receipts* given the labels of related movies.

Inference. As in the case of DNs, we use Gibbs sampling for inference in RDN models. For classification tasks, we want to estimate the full joint distribution over the target attributes in the graph. During inference, the model graph consists of a rolled-out network with both observed and unobserved variables. For example, we may want to use the network in Figure 3 to predict *movie.receipts* given *studio.country*, *actor.gender* and *actor.hasAward*. In this case, all attribute values are observed except *movie.receipts*. The values of the target variable are initialized to values drawn from the prior distribution (e.g. default distribution of *movie.receipts* in the training set). Gibbs sampling then proceeds iteratively, estimating the full joint distribution in cycles. For each target variable, the RPT model is used to return a probability distribution given the current attribute values in the rest of the graph. A new value is drawn from the distribution, assigned to the variable and recorded. This process is repeated for each unobserved variable in the graph. After a sufficient number of iterations, the values will be drawn from a stationary distribution and we can use the samples to estimate the full joint distribution. There are many implementation issues that could improve the estimates obtained from a Gibbs sampling chain, such as length of "burn in" and number of samples. We have not yet investigated the effects of these decisions on RDN performance. For the experiments reported in this paper we do not use a burn-in period and we use a fixed length chain of 2000 samples.

4 Experiments

This paper focuses on evaluation of RDNs in a classification context, where only a single attribute is unobserved in the test set—others are assumed to be observed and are not modeled with CPDs.

The experiments reported below are intended to evaluate two assertions. The first claim is that dependencies among instances can be used to improve model accuracy. We evaluate this claim by comparing the performance of two models. The first model is a conventional RPT model—an individual classification model that does not use labels of related instances, reasoning about each instance independently from other instances. We call this model *RPT-indiv*. The second model is a collective classification RDN model that exploits additional information available in labels of related instances and reasons about networks of instances collectively.

The second claim is that the RDN models, using Gibbs sampling, can effectively infer labels for a network of instances. To evaluate this claim, we include two more models for comparison. The third model is a conventional RPT model in which we allow the true labels of related instances to be used during both learning and inference. We call this model *RPT-ceiling*. This model is included as a ceiling comparison for the RDN model. It shows the level of performance possible if the model knew the *true* labels of related instances. The fourth model is intended to assess the need for a collective inference procedure. We call this model *RPT-default*. It reports the performance achieved on the first round of Gibbs sampling. This is equivalent to learning a conventional RPT model with the true labels of related instances, then randomly assigning labels according to the prior distribution of labels to use for inference. Since the test sets have connections to labeled instances in the training set (see next section for details), it is possible that these labels could provide enough information for accurate inferences, making Gibbs sampling unnecessary.

4.1 Tasks

The first data set is drawn from the Internet Movie Database (IMDb) (www.imdb.com). We gathered a sample of all movies released in the United States from 1996 through 2000, with opening weekend receipt information. The resulting collection contains 1383 movies. In addition to movies, the data set contains associated actors, directors, producers, and studios. In total, the data set contains 46,000 objects and 68,000 links. The learning task was to predict movie opening-weekend box office receipts. We discretized the attribute so that a positive label indicates a movie that garnered more than $2 million in opening-weekend box office receipts (*receipts*) (*P(Y)=0.45*).

We created training set/test set splits by temporal sampling into five sets, one for each year. Links to the future were removed from each sample. For example, the sample for 1997 contains links to movies from 1996, but not vice versa. We trained on the movies from one year (e.g. 1996) and tested on movies from the following year (e.g. 1997). Notice that during inference, there are links from the test set to fully labeled instances in the training set. This approach to sampling is intended to reproduce the expected domain of application for these models.

The RPT learning algorithm was applied to subgraphs centered on movies. The subgraphs consisted of movies and everything up to two links away—actors, directors, producers and studios, as well as movies associated with those objects. Nine attributes were supplied to the models, including studio country, actor birth-year and the class label of related movies two links away. Figure 4 shows an example RPT learned with the class labels of related movies. For a given movie x, the objects tagged "Studio Movie" refer to the other movies made by the primary studio associated with x.

The second data set is drawn from Cora, a database of computer science research papers extracted automatically from the web using machine learning techniques (McCallum, Nigam, Rennie and Seymore 1999). We selected the set of 1478 machine-learning papers published from 1994 through 1998, along with associated authors, journals, books, publishers, institutions and references. The resulting collection contains 11,500 objects and 26,000 links. The prediction task was to identify paper topic. Machine learning papers are divided into seven topics {Reinforcement Learning, Case-Based Reasoning, Probabilistic Methods, Theory, Genetic Algorithms, Neural Networks, and Rule Learning}.

As with the IMDb, we created training set/test set splits by temporal sampling into five sets, one for each year. The RPT learning algorithm used subgraphs centered on papers. The subgraphs consisted of papers, authors, journals, books, publishers, institutions and references, as well as papers associated with the authors. Twelve attributes were available to the models, including the journal affiliation, paper venue, and the topic of papers one link away (references) and two links away (through authors). Figure 5 shows an example RPT learned with the topics of related papers.

The RPT learning algorithm used randomization tests for feature selection and a Bonferroni-adjusted p-value growth cutoff of □=0.05. The RDN algorithm used a fixed number of Gibbs sampling iterations (2000).

4.2 Results and Discussion

Table 1 shows accuracy results for each of the four models on the IMDb classification tasks. On the IMDb, the RDNs models perform comparably to the *RPT-ceiling* models. This indicates that the RDN model realized its full potential, reaching the same level of performance as if it had access to the *true* labels of related movies.

In addition, the performance of the RDN models is superior to both the *RPT-indiv* models (RPT learned without labels) and the *RPT-default* models (RPT learned with labels and tested with a default labeling for the class values of related instances). In the example RPT in Figure 4 we can see that two of the five features refer to the target label on related movies. This indicates that autocorrelation is both present in the data and identified by the RPT models. The performance improvement over *RPT-indiv* is due to successful exploitation of this autocorrelation.

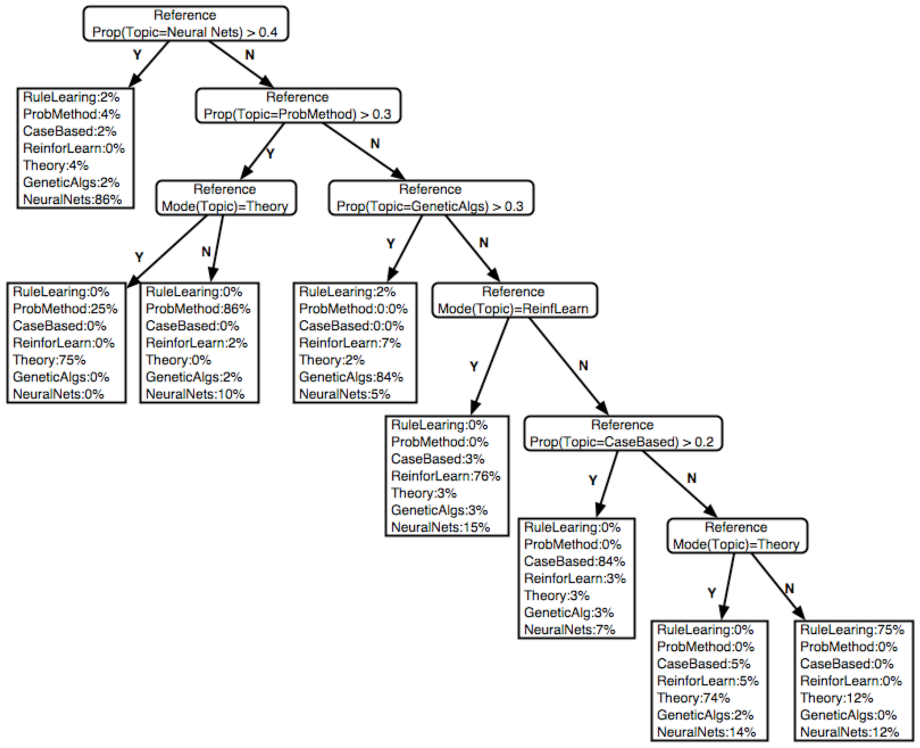


Fig. 5. Example RPT learned for the Cora dataset to predict paper topic given the topics of related papers.

We used two-tailed, paired t-tests to assess the significance of the accuracy results obtained from the four trials. The t-tests compare the RDN results to each of the three other models. The null hypothesis is that there is no difference in the accuracies of the two models; the alternative is that there is a difference. The resulting p-values are reported in the bottom row of the table, in the column of the model being compared to the RDN. The results support our conclusions above that the RDN results are significantly better than both *RPT-indiv* and *RPT-default*. Although the average performance of the RDNs is slightly lower than the *RPT-ceiling* models, the t-test indicates that this difference is not significant.

Table 1: Accuracy results for IMDb task

	RPT-indiv	*RPT-ceiling*	*RPT-default*	*RDN*
1	0.7148	0.8303	0.7473	0.8628
2	0.7500	0.8052	0.7370	0.8084
3	0.6893	0.8357	0.7429	0.7857
4	0.7103	0.8318	0.7383	0.8224
Avg	0.7161	0.8258	0.7414	0.8198
t-Test	0.0113	0.7529	0.0137	

On the Cora classification task, shown in Table 2, the RDN models show significant gains over the *RPT-indiv* and *RPT-default* models. This indicates that most of the predictive information lies in the topics of related pages. Nearly 90% of the features selected for the trees involved the topic of referenced papers. (Recall that reference topic was one of twelve attributes available to the model to form features.)

In this experiment, RDN models did not achieve the level of performance possible if the true label of related papers were known. However, the improvement from *RPT-indiv* and *RPT-default* models is notable. This is due to the paucity of predictive attributes other than the target label, clearly showing how autocorrelation can be exploited to improve model accuracy.

Table 2: Accuracy results for Cora task

	RPT-indiv	*RPT-ceiling*	*RPT-default*	*RDN*
1	0.2813	0.7437	0.4429	0.6852
2	0.2456	0.7646	0.4429	0.7013
3	0.2619	0.8027	0.5374	0.7313
4	0.2689	0.8571	0.5630	0.7983
Avg	0.2644	0.7920	0.4966	0.7290
t-Test	0.0004	0.0002	0.0004	

The difference in accuracy between the RDN and *RPT-ceiling* models may indicate that Gibbs sampling had not converged within the 2000 trials. To investigate this possibility we tracked accuracy throughout the Gibbs sampling procedure. Figure 6 shows curves for each of the tasks based on number of Gibbs iterations. The four lines represent each of the trials. Although we used 2000 iterations, we limit the graph because the accuracy plateaus within the first 150 iterations. Accuracy improves very quickly, leveling within the first 50 iterations. This shows that the approximate inference techniques employed by the RDN may be quite efficient to use in practice. We are currently experimenting with longer Gibbs chains, random restarts and convergence metrics to fully assess this aspect of the model.

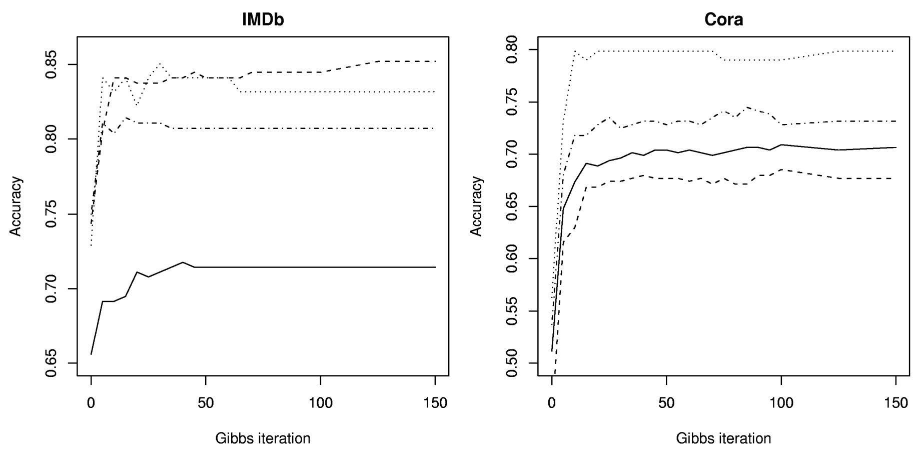


Fig. 6. Accuracy vs. number of Gibbs iterations. Each curve represents a separate trial.

If we can conclude from the learning curves that the Gibbs chain had converged, why didn't the RDN model perform as well as the *RPT-ceiling* model on Cora? One possible explanation is the lack of predictive attributes other than *topic*. The Gibbs chain may mix slowly, making the procedure unlikely to jump to distant portion of the labeling space. The inference procedure will suffer when there are no predictive attributes known with certainty to drive the mixing process in the right direction.

5 Conclusions and Future Work

These results indicate that collective classification with RDNs can offer significant improvement over non-collective approaches to classification when autocorrelation is present in the data. The performance of RDNs can approach the performance that would be possible if all the class labels of related instances were known. In addition, RDNs offer a relatively simple method for learning the structure and parameters of a graphical model, and they allow us to exploit existing techniques for learning conditional probability distributions. Here we have chosen to exploit our prior work on RPTs, which construct particularly parsimonious models of relational data, but we expect that the general properties of RDNs would be retained if other approaches to learning conditional probability distributions were used, given that those approaches are both selective and accurate.

Acknowledgments

Andrew McCallum provided valuable comments and suggestions on this work. We thank Michael Hay and Pippin Wolfe for comments. This research is supported under a National Science Foundation Graduate Research Fellowship and by DARPA, NSF and AFRL under contract numbers F30602-00-2-0597, EIA9983215 and F30602-01-2-0566. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the DARPA, the AFRL or the U.S. Government.

References

Bernstein, A., S. Clearwater, and F. Provost (2003). The relational vector-space model and industry classification. CDeR working paper #IS-03-02, Stern School of Business, New York University.

Chakrabarti, S., B. Dom and P. Indyk (1998). Enhanced hypertext categorization using hyperlinks. *Proc of ACM SIGMOD98*, pp 307-318.

Cortes, C., D. Pregibon, and C. Volinsky (2001). Communities of interest. *Proceedings Intelligent Data Analysis 2001.*

Della Pietra, S. V. Della Pietra and J. Lafferty (1997). Inducing features of random fields. I*EEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4): 380-393.

Domingos, P., M. Richardson. Mining the Network Value of Customers (2001). *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, pp. 57-66.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (2001). Learning probabilistic relational models. In *Relational Data Mining*, Dzeroski and Lavrac, Eds., Springer-Verlag.

Heckerman, D., D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie (2000). Dependency networks for inference, collaborative filtering and data visualization. *JMLR*, 1:49--75.

Jensen, D. and J. Neville (2002). Linkage and autocorrelation cause bias in relational feature selection. *Machine Learning: Proceedings of the Nineteenth International Conference*. Morgan Kaufmann.

Jensen, D., J. Neville and M. Hay (2003). Avoiding bias when aggregating relational data with degree disparity. *Proc. of the 20th Intl Joint Conf. on Machine Learning*, to appear.

Kleinberg, J. (1999). Authoritative sources in a hyper-linked environment. *Journal of the ACM* 46:604-632.

Lafferty, J., A. McCallum and F. Pereira (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of ICML2001.*

McCallum, A., K. Nigam, J. Rennie and K. Seymore (1999). A Machine Learning Approach to Building Domain-specific Search Engines. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 662-667.

Murphy, K., Y. Weiss, and M. Jordan (1999). Loopy belief propagation for approximate inference: an empirical study. *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*.

Neal, R. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Tech report CRG-TR-93-1, Dept of Computer Science, University of Toronto.

Neville, J. and D. Jensen (2000). Iterative Classification in Relational Data. *AAAI Workshop on Learning Statistical Models from Relational Data*, 42-49.

Neville, J., D. Jensen, L. Friedland, M. Hay (2003). Learning relational probability trees. *Proceedings of the 9th International Conference on Knowledge Discovery & Data Mining*, to appear.

Taskar, B., P. Abbeel and D. Koller (2002). Discriminative probabilistic models for relational data. *Proceedings of UAI2002.*

Structural Logistic Regression for Link Analysis*

Alexandrin Popescul and Lyle H. Ungar

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{popescul,ungar}@cis.upenn.edu

Abstract. We present Structural Logistic Regression, an extension of logistic regression to modeling relational data. It is an integrated approach to building regression models from data stored in relational databases in which potential predictors, both boolean and real-valued, are generated by structured search in the space of queries to the database, and then tested with statistical information criteria for inclusion in a logistic regression. Using statistics and relational representation allows modeling in noisy domains with complex structure. Link prediction is a task of high interest with exactly such characteristics. Be it in the domain of scientific citations, social networks or hypertext, the underlying data are extremely noisy and the features useful for prediction are not readily available in a "flat" file format. We propose the application of Structural Logistic Regression to building link prediction models, and present experimental results for the task of predicting citations made in scientific literature using relational data taken from the CiteSeer search engine. This data includes the citation graph, authorship and publication venues of papers, as well as their word content.

1 Introduction

A growing number of machine learning applications involves the analysis of noisy data with complex relational structure. This dictates a natural choice in such domains: the use of relational rather than propositional representation and the use of statistical rather than deterministic modeling. Classical statistical learners provide powerful modeling but are generally limited to a "flat" file propositional domain representation where potential features are fixed-size attribute vectors. The manual process of preparing such attributes is often costly and not obvious when complex regularities are involved.

We present Structural Logistic Regression, an extension of logistic regression to modeling relational data that combines the strengths of classical statistical models with the higher expressivity of features automatically generated from a relational database. Structural Logistic Regression is an "upgrade" method in a sense used in inductive logic programming (ILP) to refer to the methods which extend existing propositional learners to handle relational representation. An "upgrade" implies that modeling and relational structure search are tightly coupled into a single process dynamically driven by the assumptions and model selection criteria of a propositional learner used. This contrasts with "propositionalization" which generally implies a decoupling of relational

* This paper partially overlaps with [29].

feature generation and modeling. Propositionalization has its disadvantages compared to upgrading, as it is difficult to decide a priori what features will be useful for a given propositional learner. Upgrading techniques let their learning algorithms select their own features with their own criteria. In large problems it is impossible to "exhaustively" propositionalize; feature generation should be driven dynamically at the time of learning. An extreme form of propositionalization is generating the full join of a database. This is both impractical and incorrect—the size of the resulting table is prohibitive, and the notion of an object corresponding to an observation is lost, being represented by multiple rows. Moreover, the entries in the full join table will be "atomic" attribute values, rather than values resulting from complex queries, what we desire for our features. The dynamic approach of coupled feature generation and modeling is more time and space efficient than its static alternative: it avoids making premature decisions about the depth of the feature space to explore; it generates only the features which will be looked at by the model selection, thus avoiding over-generation or under-generation; it avoids storing the entire table containing all feature candidates; it allows for flexible search strategies, and invites a number of statistical optimizations, for example sampling from subspaces of features before or instead of fully evaluating them.

Structural Logistic Regression integrates classical logistic regression with feature generation from relational data. We formulate the feature generation process as search in the space of relational database queries, based on the top-down search of refinement graphs commonly used in ILP [9], and extend it to include aggregate operators. Statistical information criteria are used dynamically during the search to determine which features are to be included into the model.

We propose the application of Structural Logistic Regression to link prediction and argue that the characteristics of the method and the task form a good match. Link analysis is an important problem arising in many domains. Web pages, computers, scientific publications, organizations, people and biological molecules are interconnected and interact in one way or another. Being able to predict the presence of links or connections between entities in a domain is both important and difficult to do well. We emphasize three key characteristics of such domains: i) their nature is inherently multi-relational, making the standard "flat" file domain representation inadequate, ii) such data is often very noisy or partially observed, and iii) the data are often extremely sparse. For example, in the domain of scientific publications, documents are cited based on many criteria, including their topic, conference or journal, and authorship, as well as the extremely sparse citation structure. In one example given below only one link exists in more than 7,000 potential link possibilities. All attributes contribute, some in fairly complex ways. The characteristics of the task suggest: i) using relational data model as a natural way to represent and store such data, ii) using statistical learning for building robust models from noisy data, and iii) using focused feature generation to produce complex, possibly deep, but local regularities, to be combined in a single discriminative model instead of trying to produce a full probabilistic model of the entire domain.

We present experimental results for citation prediction task in the domain of scientific publications. Link prediction models in this domain can be used to recommend citations to users who provide the abstract, names of the authors and possibly a partial reference list of a paper in progress. In addition to prediction, the learned features have an explanatory power, providing insights into the nature of the citation graph structure. We use data from CiteSeer (a.k.a. ResearchIndex), an online digital library of computer science papers [22] (`http://citeseer.org/`). CiteSeer contains a rich set of relational data, including the text of titles, abstracts and documents, citation information, author names and affiliations, conference or journal names.[1]

2 Methodology

Our method couples two main components: generation of feature candidates from relational data and their selection with statistical model selection criteria (Figure 1). Relational feature generation is a search problem. It requires formulation of the search in the space of queries to a relational database. We structure the search space based on the formulation widely used in inductive logic programming for learning logic descriptions, and extend it to include other types of queries as the use of statistics relaxes the necessity of limiting the search space to only boolean values.

The language of non-recursive first-order logic formulas has a direct mapping into SQL and relational algebra (e.g. [5]), which can be used as well for the purposes of our discussion, e.g. as we do in [29]. Our implementation uses SQL for efficiency reasons providing connectivity with relational database engines.

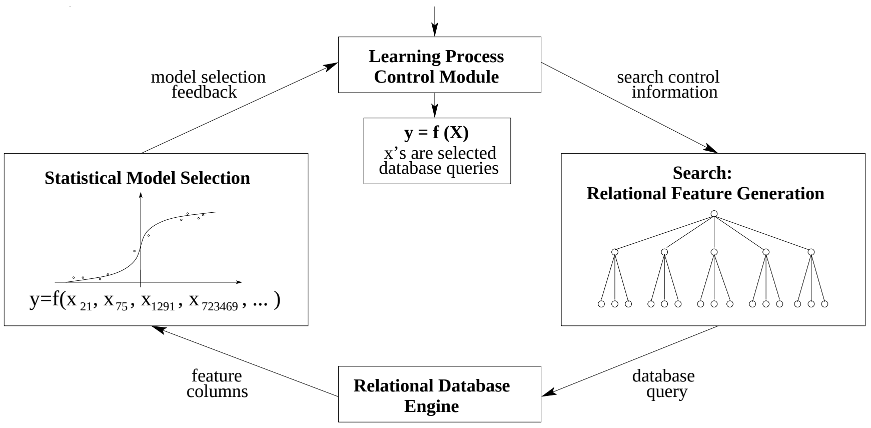

Learning Process Control Module
model selection feedback
search control information
y = f (X)
x's are selected database queries
Statistical Model Selection
y=f(x 21, x75, x1291, x723469, ...)
Search: Relational Feature Generation
feature columns
Relational Database Engine
database query

Fig. 1. The search in the space of database queries involving one or more relations produces feature candidates one at a time to be considered by the statistical model selection component. The process results in a statistical model where each selected feature is the evaluation of a database query encoding a predictive data pattern in a given domain

Logistic regression is used for classification problems. It is a discriminative model, i.e. it models conditional class probabilities without attempting to model the marginal

[1] Publication venues are extracted by matching information with the DBLP database: `http://dblp.uni-trier.de/`

distribution of features. Model parameters/regression coefficients are learned by maximizing conditional likelihood function. More complex models result in higher likelihood values, but at some point will likely overfit the data, resulting in poor generalization. A number of criteria aiming at striking the balance between likelihood optimization and model complexity have been proposed. Among the more widely used is the Bayesian Information Criterion (BIC) [33], which works by penalizing the likelihood by a term that depends on model complexity. We use sequential model selection to find a model which generalizes well by adding predictors resulting in improved BIC one at a time until the search space exploration is completed.

We introduce the notation first. Throughout this section we use the following relations: $cites(FromDoc, ToDoc)$, $author(Doc, Auth)$, $published\_in(Doc, Venue)$, and $word\_count(Doc, Word, Int)$. For compactness we use extended logic-based notation. First-order expressions are treated as database queries resulting in a table of all satisfying variable bindings, rather than a boolean value. The extended notation is necessary to introduce aggregations over entire query results or over their individual columns. Aggregate operators are subscripted with the corresponding variable name if applied to an individual column, or are used without subscripts if applied to the entire table. For example, an average count of the word "learning" in documents cited from a learning example document d, a potentially useful type of feature in document classification, is denoted as:

$$class(d) \;\sim\; ave_C\,[cites(d,\, D),\, word\_count(D,\, learning,\, C)],$$

where $\sim$ denotes "modeled using", i.e. the right hand side of the expression is one of the features in a statistical model of $class$. The duplicates in the column of satisfying bindings of C are not eliminated, unless an explicit projection of that column is performed before aggregation takes place. When necessary, we will borrow the projection operator notation (π) from the relational algebra.

The following is an example of a feature useful in link prediction; here the target concept is binary and the feature is a database query about both target documents $d1$ and $d2$:

$$cites(d1,\, d2) \;\sim\; count\,[cites(d1,\, D),\, cites(d2,\, D)]$$

is the number of common documents that both $d1$ and $d2$ cite.[2] Queries may be just about one of the documents in a target pair. For example:

$$cites(d1,\, d2) \;\sim\; count\,[cites(D,\, d2)]$$

is the number of times document $d2$ is cited. Larger values of this feature increase the prior probability of $d1$ citing $d2$, regardless of what $d1$ is.

2.1 Feature Generation

We define the search space based on the concept of "refinement graphs" [34] and expand it to include aggregate operators. Top-down search of refinement graphs is widely

[2] The database query is the right hand side of the expression.

used in inductive logic programming to search the space of first-order logic clauses. The search of refinement graphs starts with most general clauses and progresses by refining them into more specialized ones. Refinement graphs are directed acyclic graphs specifying the search space of the first-order logic queries. The space is constrained by specifying legal clauses (e.g. disallowing recursion and negation), and then structured by partial ordering of clauses, using a syntactic notion of generality (θ-subsumption [28]). Typically, a search node is expanded via a "refinement operator" to produce its most general specializations. Inductive logic programming systems using refinement graph search, usually apply two refinement operators: i) adding a predicate to the body of a clause, involving one or more variables already present, and possibly introducing one or more new variables, or ii) a single variable substitution (see e.g. [9]). We use a single refinement operator which combines the two: it adds (joins) one relation to a query expanding it into the nodes accounting for *all* possible configurations of equality conditions of new attributes with either a new or an old attribute[3], such that i) each refinement contains at least one equality condition with an old attribute, and ii) the types are taken into account to avoid adding equalities between attributes of different types. This refinement operator is complete. Not all refinements it produces are the most general refinements of a given query, however, we find that this definition simplifies pruning of equivalent subspaces; it has to account only for the type and the number of relations joined in a query.

In contrast to learning logic programs, we are not limited to searching in the space of boolean-valued expressions when building statistical models. At a high level we use refinement graphs to structure the search space. Each node of the graph is a query evaluating into a table of all satisfying variable binding. Within each node we perform a number of aggregations to produce both boolean and real-valued features. Thus, each node of the refinement graph can produce multiple feature candidates. Although there is no limit to the number of aggregate operators one may try, e.g. square root of the sum of column values, logarithm of their product etc., we find a few of them to be particularly useful. We use the following typical to SQL aggregate operators: $count$, ave, max, min and $empty$. Aggregations can be applied to a whole table or to individual columns, as appropriate given type restrictions, e.g. ave cannot be applied to a column of a categorical type. Adding aggregate operators results in a much richer search space. Binary logic-based features are also included through the aggregate operator $empty$. The situations when an aggregation is not defined, e.g. the average of an empty set, are resolved by introducing an interaction term with a 1/0 (not-defined/defined) feature.

The second aspect of defining search once the search space is structured is choosing search strategy. The present implementation performs the breadth-first search. In this setting, it is not necessary to specify the depth of the search prior to learning: intermediate models at any point of the search are usable. The decision to continue the exploration of deeper subspaces will depend on the available resources as well the expectation of how likely a model is to improve significantly if the search continues.

Search space potentially can be made arbitrarily complex. Richer queries, not necessarily involving only conjuncts and equality conditions, can also be made part of the

[3] In the experiments reported in this paper, we do not use conditions of equality with a constant.

search space. A key question for the future is how best to define such search spaces and how to control the search space complexity and search space bias.

The use of aggregate operators in feature generation makes pruning of the search space more involved. Currently, we use a hash function of partially evaluated feature columns to avoid fully recomputing equivalent features. In general, determining equivalence among relational expressions is known to be NP-complete. Polynomial algorithms exist for restricted classes of expressions, e.g. [1] (without aggregates) and [25] (with aggregates). However, deciding the equivalence of two arbitrary queries is different from avoiding duplicates when we have control over the way we structure the search space. The latter is simpler and should be the subject of future improvements.

Top-down search of refinement graphs allows a number of optimizations, e.g. i) the results of queries (prior to applying the aggregations) at a parent node can be reused at the children nodes; certainly, this needs to be balanced with the space requirements needed to store the views, and ii) a node which query results are empty for each observation should not be refined further as its refinements will also be empty.

3 Tasks and Data

Learning link prediction from relational data differs in several important aspects from other learning settings. Relational learning, in general, requires a quite different paradigm from "flat" file learning. The assumption that the examples are independent is violated in the presence of relational structure; this can be addressed explicitly [13], [14], or implicitly, as we do here, by generating more complex features which capture relational dependencies. When the right features are used, the observations are conditionally independent given the features, eliminating the independence violation.

In our link prediction setting, a class label of a learning example indicating the presence of a link between two documents is information of the same type as the rest of the link structure. Target links are not included in the background knowledge. This setting combines modeling-based and memory-based learning. We build a formal statistical model, but prediction of future data points requires database access, as each selected feature is a database query. Thus, an important aspect, more so than in attribute-value learning, is what information about new examples will be available at the time of prediction and how missing or changing background information would affect the results.

Consider the following two link prediction scenarios:

- The identity of all objects is known. Only *some* of the link structure is known. The goal is to predict unobserved links, from existing link structure alone or also using information about other available object attributes.
- New objects arrive and we want to predict their links to other existing objects. What do we know about new objects? Perhaps, we know some of their links, and want to predict the other. Alternatively, we might not know any of the links, but know some other attributes of the new objects.

In the latter case, when none of the new objects' links is known, and prediction is based solely on other attributes, e.g. only authorship and word content, feature generation would have to be controlled to not produce features based on immediate links, but use them when referring to the links in already existing background knowledge.

In this paper, we perform experiments for the first scenario. The data for our experiments was taken from CiteSeer [22]. CiteSeer catalogs scientific publications available in full-text on the web in PostScript and PDF formats. It extracts and matches citations to produce a browsable citation graph. The data we used contains 271,343 documents and 1,092,200 citations.[4] Additional information includes authorship and publication relations.[5] We use the following schema:

$cites(Document,\ Document)$,
$author(Document,\ Person)$,
$published\_in(Document,\ Venue)$.

The training and test sets are formed by sampling citations (or absent citations for negative examples) from the citation graph. We perform learning on five datasets. Four of the datasets include links among all documents containing a certain query phrase, and the fifth data set covers the entire collection. Note that the background knowledge in the first four datasets also includes all other links in the full collection; only training and test links are sampled from the subgraph induced by document subsets. Table 1 contains the summary of the datasets.

The detailed learning setting is as follows:

– Populate three relations $cites$, $author$ and $published\_in$ initially with *all* data.
– Create training and test sets of 5,000 examples each by i) randomly sampling 2,500 citations for training and 2,500 citations for testing from those in column `# Links` of the Table 1; and ii) creating negative examples by sampling from the same subgraph also 2,500/2,500 train/test of "empty" citations, i.e. pairs of documents not citing each other.
– *Remove* test set citations from the $cites$ relation; but not the other information about the documents involved in the test set citations. For example, other citations of those documents are not removed.
– *Remove* training set citations from the $cites$ relation, so as not to include the actual answer in the background knowledge.
– Learning is performed i) using $cites$ relation only, or ii) using all three relations $cites$, $author$ and $published\_in$.

The positive and negative classes in this task are extremely unbalanced. We ignore the lack of balance at the training phase; at the testing phase we perform additional *precision-recall* curve analysis for larger negative class priors. The next section reports experimental results.

[4] This data is part of CiteSeer as of August 2002. Documents considered are only non-singleton documents out of the total of 387,703. Singletons are documents which both citation indegree and outdegree registered in CiteSeer are zero.

[5] The authorship information is known for 218,313 papers, and includes 58,342 authors. Publication venues are known for 60,646 documents. The set of venues consists of 1,560 conferences and journals.

Table 1. Number of documents, number of citations and citation graph density in each dataset. Density is the percentage of existing citations out of the total number of possibilities, (# Docs)^2

Dataset	# Docs	# Links	Density (10^{-2}%)
"artificial intelligence"	11,144	16,654	1.3
"data mining"	3,424	6,790	5.8
"information retrieval"	5,156	8,858	3.3
"machine learning"	6,009	11,531	3.2
entire collection	271,343	1,092,200	0.1

4 Results

We start by presenting the results for the balanced class priors test scenario, and continue with the analysis of the unbalanced class settings. Two sets of models are learned for each dataset: i) using only $cites$ relation, and ii) using all three relations $cites$, $author$ and $published\_in$.

When only $cites$ is used the average test set accuracy in five datasets is 88.73% and when all three relations are used the average increases to 90.90%.[6] In both cases the search explored features involving joins of up to three relations. It is not unreasonable to expect that even better models can be built if we allow the search to progress further. Table 2 details the performance in each dataset. The largest accuracy of 93.22% is achieved for the entire CiteSeer dataset. Even though this is the largest and the most sparse dataset, this is not surprising because, since the features are not domain specific and rely on the surrounding citation structure, this dataset retains more useful "supporting link structure" after some of them are removed to serve as training and testing examples (Section 3).

Table 2. Training and test set accuracy (%) of the models learned from the $cites$ alone, and from the $cites$, $author$ and $published\_in$. 5,000 train/test examples; balanced priors

Dataset	with $cites$		with all data	
	Train	Test	Train	Test
"artificial intelligence"	90.24	89.68	92.60	92.14
"data mining"	87.40	87.20	89.70	89.18
"information retrieval"	85.98	85.34	88.88	88.82
"machine learning"	89.40	89.14	91.42	91.14
entire collection	92.80	92.28	93.66	93.22

In the experiments using only the $cites$ relation the average number of features selected is 32; 13 of the selected features are the same across all five datasets. When

[6] The predicted probability of 0.5 was used as the decision cut-off in logistic regression.

all three relations $cites$, $author$ and $published\_in$ are used the average number of selected features is 40, with 14 features common to all five datasets. In addition to more obvious features, such as $d1$ is more likely to cite $d2$ if $d2$ is frequently cited, or if the same person co-authored both documents, or if $d1$ and $d2$ are co-cited, or cite the same papers[7], we learned some more interesting features. For example, a document is more likely to be cited if it is cited by frequently cited documents. Locally, this effectively learns the concept of an authoritative document [17], [26]. Or, the following feature, selected in all models:

$$cites(d1,\ d2) \ \sim \ count\,[\pi_{D2}\ (cites(D1,\ d2),\ cites(D1,\ D2))]$$

increases the probability of a citation if $d2$ is co-cited with many documents. Since this feature is selected in addition to the simple citation count feature, it could mean that either $d2$ appears more often in reviews, which tend to have longer lists of references, or it is cited from documents having smaller overlap among their references, which is more probable if they belong to different communities.

We compare the above results to the models trained on only binary features, i.e. when using only the $empty$ aggregate operator on the entire table. Such features are the logic-based features from the original formulation of refinement graph search. The binary features resulted in models with lower out-of-sample accuracies in all datasets. On average the accuracy with only binary features is 2.52 percentage points lower in models using $cites$ relation, and 2.20 percentage points lower in models using all three relations. The decrease of accuracy is significant at the 99% confidence level in both cases according to the t-test. We are currently unable to make experimental comparisons with a representative of classical ILP, FOIL; it runs out of memory on the system on which we have performed the rest of our experiments.

The class priors in our data are extremely unbalanced, due to the sparsity of the citation structure. The citation graph of the “artificial intelligence” dataset, for example is only 1.34×10^{-4} dense; that means that for one citation between two documents there are more than 7,000 non-existing citations, thus there are more than 7,000 times as many negative examples as there are positive. We perform the precision-recall curve analysis of our models trained with balanced class priors for testing situations with increased negative class proportions.

We vary the ratio k of the number of negative to the number positive examples used at testing. The ratio of one corresponds to the initial balance. We use for illustration the “artificial intelligence” dataset and the model trained using all three relations. New larger sets of negative examples are sampled with replacement from all “non-existing” links between documents in this dataset. Figure 2 presents precision-recall curves for k = 1, 10 and 100. As k increases the precision falls for the same levels of recall. Reducing the negative class prior should be performed when possible by filtering out obviously negative examples, for example by using a text based similarity, or other measures appropriate for a given task.

In application to citation recommendation, when only a few citations need to be recommended, we should care only about the high precision mode performance. In the case

[7] These two features correspond to the concepts of co-citation and bibliographic coupling used in bibliometrics.

of the precision recall curve for k = 100, for example, 10% of citations can be recalled for recommendation with 91% precision. This is an overall measure of performance—some users can receive more than enough recommendations, and others none. When we want to recommend a fixed number of citations to *every* user, the CROC performance metric should be used [32].

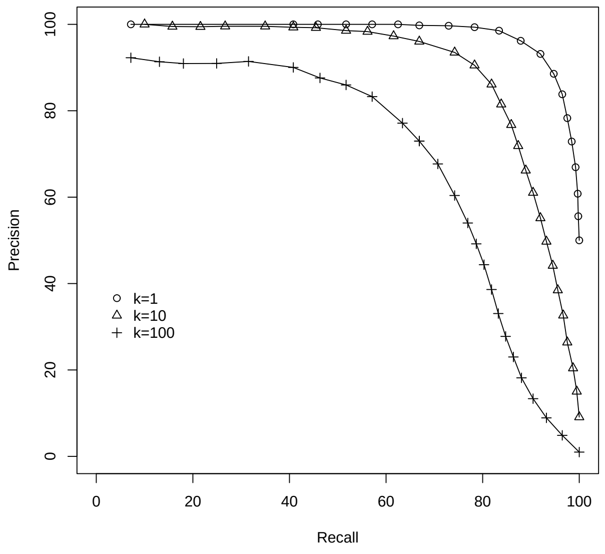


Fig. 2. Precision-recall curves for the “artificial intelligence” dataset with different class priors. k is the ratio of the number of negative to the number of positive examples used at testing

5 Related Work and Discussion

A number of approaches “upgrading” propositional learners to multi-relational domains have been proposed in the inductive logic programming community [21]. Often, these approaches upgrade learners most suitable to binary attributes. TILDE [4] and WARMR [8], for example, upgrade decision trees and association rules, respectively. S-CART [20] upgrades CART, a propositional algorithm for learning classification and regression trees. Upgrading implies that generation of relational features and their modeling are coupled into a single loop. Structural Logistic Regression presented in this paper can be categorized as an “upgrade”.

Another approach is “propositionalization” (see e.g. [19]). In a broader sense “propositionalization” refers to any process of transforming a first-order representation into a propositional representation to make it suitable for a propositional learning algorithm.

In a narrower sense, as used more often in the literature, it is defined as "decoupling" of feature generation from modeling, where the features are first constructed from relational representation and then presented to a propositional algorithm. While in the former case we can view "upgrading" also as a form of "propositionalization", the latter emphasizes that the native propositional algorithm's model selection criteria do no take part in feature construction.

One form of "decoupled propositionalization" is to learn a logic theory with an ILP rule learner and then use the bodies of learned clauses as binary features in a propositional learner. For example, Srinivasan and King [35] use linear regression to model features constructed from the clauses returned by Progol. Decoupling feature construction from modeling, however, retains the inductive bias of the technique used to construct features, and better models can potentially be built if we allow a propositional learner itself to select its own features based on its own criteria. First Order Regression System (FORS) [15] more closely integrates feature construction into regression modeling, but does so using a FOIL-like covering approach for feature construction. Additive, or cumulative, models, such as linear or logistic regression, have different criteria for feature usefulness; fusing of feature construction and model selection into a single process is advocated in this context in [3] and [30].

Aggregate operators present an attractive way of extending feature spaces. For example, Knobbe et al. [18] use aggregates in propositionalization by first constructing a single table involving aggregate summaries of relational data and then using a standard propositional learner on this table. Perlich and Provost present a detailed discussion of aggregation in relational learning in [27].

An approach explicitly addressing numerical reasoning limitations of classical ILP is proposed in [36]. This approach augments the search space within P-Progol with clausal definitions of numerical, including statistical, functions which are lazily evaluated during the search. Our framework allows for the implementation of this extension to supplement the numerical reasoning capability we achieve via aggregate operators. The choice of numerical functions to be included into the search formulation on a par with the original relations should certainly be driven by the user insights into the application domain, as including too many augmenting functions makes the size of the search space prohibitive in large problems.

A number of learning models have been proposed which combine the expressivity of first-order logic with probabilistic semantics to address uncertainty. For example, "Stochastic Logic Programs" [23] model uncertainly from within the ILP framework by providing logic theories with a probability distribution; "Probabilistic Relational Models" (PRMs) [11] are a relational "upgrade" of Bayesian networks. Other examples include "Bayesian Logic Programs" [16], PRISM [31], "Relational Markov Networks" [37] and "Relational Markov Models" [2]. The marriage of richer representations and probability theory makes resulting formalisms extremely powerful, and inevitably a number of equivalences among them can be observed. In addition to a fundamental question of semantic and representational equivalence, it is useful to also consider the differences in how models are built, i.e. what objective function is optimized, what training algorithm is used to optimize that function, what is done to avoid over-fitting, what simplifying assumptions are made. We believe that answering these questions in a single study will greatly enhance the discussion of these models.

A conflict exists between the two goals: i) probabilistically characterizing the whole domain at hand and ii) building a model that would address a specific question only, such as classification or regression modeling of a single response variable. This distinction typically leads to two philosophies in probabilistic/statistical machine learning: "generative" modeling and "discriminative" modeling. Generative models would attempt to model the distribution of their features, while discriminative models, like ours, do not do that, accepting them as they are and solely focusing on modeling the response variable distribution given these features, thus making it easier to learn by reducing the degrees of freedom that need to be estimated. For this reason, our method allows inclusion into the model of arbitrarily complex features without trying to do the impossible in large and sparse environments—estimating their distribution.

PRMs, for example, are generative models of joint probability distribution of entities and their attributes in a relational domain. Being a joint probability model of the entire domain, PRMs can provide answers to a large number of questions, including class labels, latent groupings, changing beliefs given new observations. An important limitation, however, of generative modeling is that in reality there is rarely enough data to reliably estimate the entire model. Generative modeling does not allow searching for complex features arbitrarily deep. One can achieve superior performance when focusing only on a particular question, e.g. class label prediction, and training models discriminatively to answer that question. Taskar et al. [37] propose "Relational Markov Networks" (RMNs)—a relational extension of discriminatively trained Markov networks. In RMNs, however, the structure of a learning domain, determining which direct interactions are explored, is prespecified by a relational template; this precludes the discovery of deeper and more complex regularities made possible by more focused feature construction advocated in this paper. Certainly, classification and regression do not exhaust potential applications. Generative modeling can prove useful in other problems, e.g. a formulation similar to PRMs, but semantically different, called a "Statistical Relational Model"—a statistical model of a particular database instantiation—was proposed for optimizing relational database queries [12].

Link analysis plays an important role in the hypertext domains, a notable example being Google, which uses the link structure of the Web by employing a link based concept of page authority in ranking search results [26]. In addition to knowing the authoritative documents, it is often useful to know the web pages which point to authorities on a topic, the so called called "hub" pages [17], which correspond to the concept of review papers in the scientific literature domain. A technique called "Statistical Predicate Invention" [7] was proposed for learning in hypertext domains, including learning certain types of relations between pages. It combines statistical and relational learning by using classifications produced by Naive Bayes as predicates in FOIL. Statistical Predicate Invention preserves FOIL as the central modeling component and calls Naive Bayes to supply new predicates. Neville and Jensen [24] propose an iterative technique based on a Bayesian classifier that uses high confidence inferences to improve class inferences for linked objects at later iterations. Cohn and Hofmann propose a joint probability model of document content and connectivity in [6].

6 Conclusions and Future Work

We presented Structural Logistic Regression, an “upgrade” of standard logistic regression to handle relational data representation. We demonstrate the advantages of using richer first-order representation with classical statistical modeling by applying the method to link prediction in the domain of scientific literature citations. The link prediction task is inherently relational, noisy and extremely sparse, thus suggesting relational representation and the use of discriminative statistical modeling of complex features obtained by a tightly coupled search in the space of database queries and selected with model’s native information criteria. Focused search allows generation of complex features and avoids their manual “packaging” into a single table, a process that can be expensive and difficult. Discriminative modeling, such as in logistic regression, does not require modeling the distribution of individual features, an impossible task in sparse domains with many potential predictors; it focuses instead solely on modeling the response variable’s distribution given these features. Our method extends beyond classical ILP because statistics allows generation of richer features, better control of search of the feature space, and more accurate modeling in the presence of noise. On the other hand, our method differs from relational probabilistic network models, such as PRMs and RMNs, because these network models, while being good at handling uncertainly, do not attempt to learn and model new complex relationships. Other regression models, such as linear regression for modeling continuous outcomes or Poisson regression for modeling count data can be used within our framework provided a common package is used—they all fall into the category of generalized linear models.

In the citation prediction task explored here, the learned models have explanatory as well as predictive power. Selected features provide insights into the nature of citations; some features “re-discovered” common concepts in link analysis and bibliometrics, such as bibliographic coupling, co-citation, authoritative and “hub” documents. Other linked environments, such as the Web, social networks, and biological interactions, e.g. protein interactions, we believe, can be explored with this methodology.

We are extending this work in three main directions: better feature selection, better search, and incorporation of relations derived from clustering into the search space.

Learning takes place with an exponential number of potential feature candidates, only relatively few of which are expected to be useful. Instead of avoiding large feature spaces because of the dangers of overfitting, we should rather learn to deal with them with more sophisticated model selection criteria. Feature selection methods recently derived by statisticians give promising results for handling this potentially infinite stream of features with only a finite set of observations.

Our formulation supports the introduction of sophisticated procedures for determining which subspaces of the query space to explore. Intelligent search techniques which combine feedback from the feature selection algorithms, other information such as sampling from feature subspaces to determine their promise and using database meta-information will help scale to truly large problems.

We advocate the use of clustering to extend the set of relations used in feature generation. Clusters improve modeling of sparse data, improve scalability, and produce richer representations [10]. New cluster relations can be derived using attributes in other relations. For example, one can cluster documents based on words, giving “topics”, or authors based on co-authorship, giving "communities". Once clusters are formed, they represent new relationships which can be added to the relational database schema, and then used interchangeably with the original relations.

Acknowledgments

We thank Steve Lawrence, David Pennock and the anonymous reviewers for their help.

References

1. A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal of Computing*, 8(2):218–246, 1979.
2. Corin Anderson, Pedro Domingos, and Dan Weld. Relational Markov models and their application to adaptive web navigation. In *Proc. of KDD-2002*, pages 143–152, 2002.
3. Hendrik Blockeel and Luc Dehaspe. Cumulativity as inductive bias. In P. Brazdil and A. Jorge, editors, *Workshops: Data Mining, Decision Support, Meta-Learning and ILP at PKDD-2000*, pages 61–70, 2000.
4. Hendrik Blockeel and Luc De Raedt. Top-down induction of logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
5. Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin, 1990.
6. David Cohn and Thomas Hofmann. The missing link - A probabilistic model of document content and hypertext connectivity. In *NIPS*, volume 13. MIT Press, 2001.
7. M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1/2):97–119, 2001.
8. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
9. Saso Dzeroski and Nada Lavrac. An introduction to inductive logic programming. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 48–73. Springer-Verlag, 2001.
10. Dean Foster and Lyle Ungar. A proposal for learning by ontological leaps. In *Proc. of Snowbird Learning Conference*, Snowbird, Utah, 2002.
11. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 307–338. Springer-Verlag, 2001.
12. Lise Getoor, Daphne Koller, and Benjamin Taskar. Statistical models for relational data. In *Workshop on Multi-Relational Data Mining at KDD-2002*, pages 36–55, 2002.
13. P.D. Hoff. Random effects models for network data. In *Proc. of the National Academy of Sciences: Symposium on Social Network Analysis for National Security*, 2003.
14. D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proc. of ICML*, pages 259–266. Morgan Kaufmann, 2002.
15. Aram Karalic and Ivan Bratko. First order regression. *Machine Learning*, 26:147–176, 1997.
16. K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In *Proc. of ILP-2001*, 2001.
17. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
18. Arno J. Knobbe, Marc De Haas, and Arno Siebes. Propositionalisation and aggregates. In De Raedt and A. Siebes, editors, *PKDD, LNAI 2168*, pages 277–288. Springer-Verlag, 2001.

19. S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.
20. Stefan Kramer and Gerhard Widmer. Inducing classification and regression trees in first order logic. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 140–159. Springer-Verlag, 2001.
21. W. Van Laer and L. De Raedt. How to upgrade propositional learners to first order logic: A case study. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 235–261. Springer-Verlag, 2001.
22. Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.
23. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *the 5th Int'l Workshop on Inductive Logic Programming*, 1995.
24. J. Neville and D. Jensen. Iterative classification in relational data. In *Workshop on Learning Statistical Models from Relational Data, AAAI*, pages 13–20. AAAI Press, 2000.
25. Werner Nutt, Yehoshua Sagiv, and Sara Shurin. Deciding equivalences among aggregate queries. In *Proc. of PODS-98*, pages 214–223, 1998.
26. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Computer Science Department, Stanford University, 1998.
27. Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *Proc. of KDD-2003*, 2003.
28. G. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1969.
29. Alexandrin Popescul and Lyle H. Ungar. Statistical relational learning for link prediction. In *Workshop on Learning Statistical Models from Relational Data at IJCAI-2003*, 2003.
30. Alexandrin Popescul, Lyle H. Ungar, Steve Lawrence, and David M. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *Workshop on Multi-Relational Data Mining at KDD-2002*, pages 130–141, 2002.
31. Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th Int'l Conference on Logic Programming (ICLP95)*, pages 715–729, 1995.
32. Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proc. of the 25'th Conference on Research and Development in Information Retreival (SIGIR 2002)*, 2002.
33. Gideon Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1979.
34. E. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.
35. A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.
36. Ashwin Srinivasan and Rui C. Camacho. Numerical reasoning with an ILP system capable of lazy evaluation and customized search. *J. of Logic Programming*, 40(2-3):185–213, 1999.
37. Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proc. of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-2002)*, 2002.

Scaling Up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism

Lappoon R. Tang, Raymond J. Mooney, and Prem Melville

Department of Computer Sciences,
University of Texas at Austin,
Austin, TX 78712, U.S.A.
{rupert, mooney, melville}@cs.utexas.edu
http://www.cs.utexas.edu/users/ml/

Abstract. Inductive Logic Programming (ILP) has been shown to be a viable approach to many problems in multi-relational data mining (e.g. bioinformatics). Link discovery (LD) is an important task in data mining for counter-terrorism and is the focus of DARPA's program on Evidence Extraction and Link Discovery (EELD). Learning patterns for LD is a novel problem in relational data mining that is characterized by having an unprecedented number of background facts. As a result of the explosion in background facts, the efficiency of existing ILP algorithms becomes a serious limitation. This paper presents a new ILP algorithm that integrates top-down and bottom-up search in order to reduce search when processing large examples. Experimental results on EELD data confirm that it significantly improves efficiency over existing ILP methods.

1 Introduction

The terrible events of September 11, 2001 have sparked increased development of information technology that can aid intelligence agencies in detecting and preventing terrorism. The Evidence Extraction and Link Discovery (EELD) program of the Defense Advanced Research Projects Agency (DARPA) is one attempt to develop new computational methods for addressing this problem. More precisely, *Link Discovery* (LD) is the task of identifying known, complex, multi-relational patterns that indicate potentially threatening activities in large amounts of relational data. Some of the input data for LD comes from *Evidence Extraction* (EE), which is the task of obtaining structured evidence data from unstructured, natural-language documents (e.g. news reports), other input data comes from existing relational databases (e.g. financial and other transaction data). Finally, *Pattern Learning* (PL) concerns the automated discovery of new relational patterns for detecting potentially threatening activities in large amounts of multi-relational data.

Scaling to large datasets in data mining typically refers to increasing the *number* of training examples that can be processed. Another measure of complexity that is particularly relevant in multi-relational data mining is the *size* of examples, by which we mean the number of ground facts used to describe the examples. To our knowledge, the challenge problems developed for the EELD program are the largest ILP problems attempted to date in terms of the number of ground facts in the background knowledge. Relational data mining in bioinformatics [5] was probably the previously largest ILP problem in this sense. Table 1 shows a comparison between link discovery and, to our knowledge, the largest problem in bioinformatics.

Domain	*# Bg. preds.*	*Avg. Arity*	*# Bg. facts*
Link Discovery	52	2	$\approx$ 568k
Bioinformatics	36	4.9	$\approx$ 24k

Table 1. Link Discovery versus Bioinformatics (e.g. carcinogenesis). *# Bg. preds.* is the number of different predicate names in the background knowledge, *Avg. Arity* is the average arity of the background predicates, and *# Bg facts* is the total number of ground background facts.

Scaling up ILP to efficiently process large examples like those encountered in EELD is a significant problem. Section 2 discusses the problems existing ILP algorithms have scaling to large examples and presents our general approach to controlling the search for multi-relational patterns by integrating top-down and bottom-up search. Section 3 presents the details of our new algorithm, BETH. Section 4 presents some theoretical results on our approach. Experimental results are presented and discussed in Section 5, followed by concluding remarks in Section 6.

2 Combining Top-down and Bottom-up Approaches in BETH

The two standard approaches to ILP are bottom-up and top-down [6]. Bottom-up methods start with a very specific clause generated from an individual positive example and generalize it as far as possible without covering negative examples. Top-down methods start with the most general (empty) clause and repeatedly specialize it until it no longer covers negative examples. Both approaches have problems scaling to large examples.

The state-of-the-art ILP approach, originated from bottom-up methods, is based on inverse entailment [2]. The most popular approach to implementing inverse entailment is a two-stage process: 1) *saturation* which builds up the most specific clause (a.k.a. *bottom clause*) describing a positive example, and 2) *truncation* which finds solutions that generalize the bottom clause [3]. This approach is implemented in PROGOL [2] and its successor ALEPH.[1]

[1] The Aleph Manual can be accessed via `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html`.

Given a positive example and background knowledge, the bottom clause can be infinite, and practically one has to bound it. In PROGOL, it is bounded by five parameters: i, r, $\mathcal{M}$, j^-, and j^+ (please refer to [2] for more details). Unfortunately, the complexity of PROGOL's bottom clause is exponential w.r.t. the variable depth i, which results in a hypothesis space that is doubly exponential! (The size of the subsumption lattice is two to the power of the size of the bottom clause.)

In problems with large examples like EELD, the background knowledge contains many facts using numerous predicates that describe each complex object or event. Typically, many of these facts are irrelevant to the task. However, PROGOL's bottom clause includes every piece of background knowledge (within the recall bound r) in its body. This leads to intractably large bottom-clauses which generates an exponentially larger hypothesis space when learning a clause. This leads one to wonder if it is possible to bound the bottom-clause differently so that it contains only a relevant subset of background facts.

A strength of the top-down approach is that the generation of literals is inherently directed by the heuristic search process itself: only the set of literals that make refinements to clauses in the search beam are generated. Clauses with insufficient heuristic value are discarded, saving the need to generate literals for them. So, there is a tangible link between the entire set of literals that could be included in a bottom-clause and the heuristic search for a good clause. Therefore, perhaps it is possible to employ the heuristic search as a guide to selecting a relevant subset of background facts for inclusion in an alternative bottom-clause.

A major weakness of the top-down approach (as far as literal generation is concerned) is that the enumeration of all possible combination of variables generates many more literals than necessary; some literals generated by the algorithm are not even guaranteed to cover one positive example. The complexity of enumerating all such combinations in FOIL [7] (and mFOIL [6]) is exponential w.r.t. the arity of the predicates [8]. [2] A corresponding strength of the bottom-up approach is that a literal is created using a ground atom describing a known positive example. The advantages are: 1) specializing using this literal results in a clause that is guaranteed to cover at least the seed example, and 2) the set of literals generated are constrained to those that satisfy 1).

Given the strengths and weaknesses of typical top-down and bottom-up approaches, it seems that one can take advantage of the strength of each approach by combining them into one coherent approach. More precisely, we no longer build the bottom clause using a random seed example before we start searching for a good clause. Instead, after a seed example is chosen, one generates literals in a top-down fashion (i.e. guided by heuristic search) except that the literals generated are constrained to those that cover the seed example. Based on this idea, we have developed a new system called **B**ottom-clause **E**xploration

[2] Enforcing argument type restrictions can help lower the complexity but cannot completely solve the problem.

Through **H**euristic-search (BETH) in which the bottom clause is not constructed in advance but "discovered" during the search for a good clause.[3]

3 The Algorithm

BETH's bottom clause is virtual in the sense that the algorithm does not have to construct it to work, unlike PROGOL/ALEPH; it is, nonetheless, constructed to facilitate collection of statistics. However, the virtual bottom clause is a real bound on the subsumption lattice (see Section 4).

Let us begin with some basic definitions. A *predicate specification* takes the form *PredName/Arity* where *PredName* is the name of the predicate in concern and *Arity* its arity. A list of predicate specifications for the background knowledge is given to the ILP system before learning starts. The function $predname(P)$ returns the predicate name of the predicate specification of the background predicates P and $arity(P)$ returns its arity. Likewise, $predname(L)$ returns the predicate name of the literal L and $arity(L)$ returns its arity.

3.1 Constructing a Clause

The outermost loop of BETH is a simple set covering algorithm like that of any typical ILP algorithm: 1) find a good clause which covers a non-empty subset of positive examples, 2) remove the positive examples covered by the clause from the entire set of positive examples, 3) add the clause found to the set of clauses being built (a.k.a. theory) which was initially empty, 4) repeat step 1) to step 3) until the entire set of positive examples are covered by the theory, 5) return the entire set of clauses found.

The way a clause is constructed in BETH is very similar to a traditional top down ILP algorithm like FOIL; the search for a good clause goes from general to specific. It starts with the most general clause $\square$ which is specialized by adding a literal to its body. The most general clause $\square = T \leftarrow true$ where T is a literal such that $predname(T) = predname(e)$ and $arity(T) = arity(e)$, where e is a randomly chosen seed example from the set of positive examples. A beam of potentially good clauses is kept while searching for refinements of each clause in the beam. The construction of a clause terminates when there is a clause in the beam which is sufficiently accurate (i.e. its m-estimate is greater than or equal to a certain threshold).

In addition, we also compute the bottom clause which bounds the search space. The initial bottom clause is set to the smallest (i.e. $e \leftarrow true$ which has an empty set of literals in the body of this initial bottom clause and e is from the set of positive examples) in which case the search space contains only the most general clause (a.k.a. the empty clause). The bound is expanded incrementally during the search for a good clause. The bound is fixed when a *sufficiently good*

[3] PROGOL and ALEPH are really, more precisely, "Subsumption lattice exploration through heuristic-search". Here, we explore the bottom clause and the subsumption lattice simultaneously.

clause is found, at which point both the clause and the bound are returned as solutions to the search. The algorithm which constructs a clause is outlined in Figure 1.

```
1. Given a set of predicate specifications $\mathcal{P}$ of the background predicates, a beam width $b$, a
   bound on the clause length $n$, variable depth bound $i$, recall bound $r$, a non-empty set of
   positive examples $Pxs$ and a set (possibly empty) of negative examples $Nxs$.
2. Randomly choose a seed example $e \in Pxs$.
3. $\bot_0 := e \leftarrow true$.
4. $Q_0 := \{\Box\}$.
5. $Q := Q_0$.
6. $\bot := \bot_0$.
7. REPEAT
   $generate\_refinements(Q, \mathcal{P}, b, n, i, r, Pxs, Nxs, Q', \bot, \bot')$,
   $Q := Q'$,
   $\bot := \bot'$
   UNTIL
   there is a clause $C \in Q$ which is sufficiently accurate. $Q'$ and $\bot'$ are output variables and the
   rest in generate_refinements are input variables.
8. Return $C$ and $\bot$.
```

Fig. 1. The construction of a clause in BETH

3.2 Generating Refinements for a Clause

To find all the refinements of a given clause C_i, first find a substitution θ, which satisfies the body of the clause (a "successful proof" of the clause, given the background facts); then construct a literal (with dummy variables) R_j for a predicate specification in $\mathcal{P}$, and find a substitution β that makes $R_j\beta$ a ground atom such that $C_i\theta$ and $R_j\beta$ satisfy the following conditions we call *refinement constraints*: 1) the link constraint: one of the arguments of $R_j\beta$ has to appear in $C_i\theta$ (this is to make sure that the resulting clause is still a linked clause), 2) the unique-literal constraint: $R_j\beta \notin C_i\theta$ (to avoid making two identical literals). We try to find pairs of θ and β satisfying the refinement constraints, but at most r distinct ground atoms $R_j\beta$ will be used. For example, suppose $C_i\theta = f(a,b) \leftarrow g(a,c), h(c,d), g(b,e)$ and $R_j\beta_1 = g(e,f)$ and $R_j\beta_2 = g(a,c)$, then only $R_j\beta_1$ satisfies all the refinement constraints, as $R_j\beta_2$ fails the unique-literal constraint. So, only $R_j\beta_1$ will be used to make literals for the clause C_i.

To avoid repeatedly finding a successful proof of a given clause by theorem proving, we make a set of "cached proofs" for each clause in the beam (similar to the way variable bindings are stored in extensional FOIL) by starting with the initial proof $e \leftarrow true$, where e is a randomly chosen seed example, and we incrementally update the cache of proofs of each clause by adding to the end of each proof a ground atom satisfying all the refinement constraints. A bound is also given to the cache size. When finding a satisfying substitution θ for a clause C_i in the beam, we will simply unify C_i with a proof in its cache. If there is no $R_j\beta$ satisfying the refinement constraints, which can happen if the first chosen example e was a "bad" one, a new example $e' \neq e$ will be randomly chosen from the remaining set of positive examples to be covered. The clause C_i will be replaced (in the beam) by the most general clause such that its cache of proofs will contain only $e' \leftarrow true$. The idea is that if a clause cannot be refined, then we will just restart with a different seed example.

One can also take advantage of type declarations (if available) to further restrict the number of predicate specifications needed to be considered for a given clause — one needs only to consider those which contain at least one argument type which is the same as at least one of the types of all the variables in the current clause (so that a linked clause that satisfies the type constraints is possible).

One can also make use of mode declarations (if available) by substituting arguments with "input" mode for constants which appear in the clause, provided that the argument type and the constant type are the same (similar to the way the bottom clause is built in PROGOL). One needs to find satisfying substitutions for R_j, for each unique way of substituting arguments with input mode for constants in the clause. The algorithm for generating refinements to a clause is outlined in Figure 2.

1. Given a set of predicate specifications $\mathcal{P}$ of the background predicates, a beam width b, a bound on the clause length n, variable depth bound i, recall bound r, a non-empty set of positive examples Pxs and a set (possibly empty) of negative examples Nxs, the current bottom clause $\perp$ (i.e. the current bound on the search space).
2. For each clause $C_i \in Q$ and for each $P_j \in \mathcal{P}$, make a literal R_j with dummy variables such that $predname(R_j) = predname(P_j)$ and $arity(R_j) = arity(P_j)$.
3. Find substitutions θ, β such that 1) θ satisfies C_i, 2) β satisfies R_j, and 3) $C_i\theta$ and $R_j\beta$ satisfy all the *refinement constraints.*
4. Collect at most r such ground atoms $R_j\beta$ for different θ and β.
5. For each pair of $C_i\theta$ and $R_j\beta$ satisfying all the refinement constraints, $make\_literals(C_i\theta, R_j\beta, Lits)$ and add $R_j\beta$ to the body of $\perp$.
6. For each $L \in Lits$, add L to the body of C_i to make C_i' and let the set of all C_i''s be Q_i.
7. Evaluate each clause in $\bigcup_{C_i \in Q} C_i$ by a heuristic (e.g. m-estimate) given Pxs and Nxs.
8. Put only the best b clauses into Q'.
9. Let $\perp'$ be the resulting bottom clause after adding all the ground atoms $R_j\beta$'s to the body of $\perp$ for each C_i and R_j (such that there exists θ and β satisfying all the refinement constraints).
10. Return Q' and $\perp'$.

Fig. 2. Generate Refinements

3.3 Making Literals

To make literals given a clause C, a satisfying substitution θ of C, and a ground atom $R_j\beta$, we replace arguments of $R_j\beta$ by variables in C instantiated, in θ, to these arguments in $R_j\beta$, *only if* $R_j\beta$ is not in $C\theta$ and the resulted literal observes the variable depth bound. θ^{-1} replaces all occurrences of a term by the same variable. For example, consider a clause $C : f(A, B) \leftarrow g(B, D), h(A, E), l(D, E)$, $\theta = \{A/a, B/b, D/b, E/e\}$ (thus, $\theta^{-1} = \{a/A, b/B, b/D, e/E\}$) and two ground atoms $a_1 = p(b, e)$ and $a_2 = p(e, f)$. We can make two literals using a_1: 1) $p(B, E)$ (since $b/B, e/E \in \theta^{-1}$), and 2) $p(D, E)$ (since $b/D, e/E \in \theta^{-1}$). We can make one literal using a_2: $p(E,F)$ (since $e/E \in \theta^{-1}$, but the constant f is not bound to any variable in θ^{-1}). However, if the variable depth bound is one, then the literal $p(E,F)$ will be rejected because the depth of F is two. The variable depth $d(V)$ of variable V is defined in LINUS [6]. The algorithm for making literals is outlined in Figure 3.

1. Given a clause $C\theta$ of the form $e \leftarrow a_1, \ldots, a_n$ (where C is the current clause being refined, i.e. specialized, and θ is a substitution that satisfies C and $e \in Pxs$ and background knowledge $BK \models a_i$ for each ground atom a_i in the body of $C\theta$) and a ground atom a_{n+1} such that $BK \models a_{n+1}$.
2. Make a set of literals *Lits* such that each literal $L \in Lits$ satisfies: 1) $predname(L) = predname(a_{n+1})$, 2) $arity(L) = arity(a_{n+1})$, 3) suppose the constant c_i is the ith argument of a_{n+1} and the variable V_i is the ith argument of L. If c_i appears in $C\theta$, then $c_i/V_i \in \theta^{-1}$; otherwise, V_i is a new variable not appearing in C, 4) there is no variable V in L such that $d(V) > i$ where i is the variable depth bound.
3. Return *Lits*.

Fig. 3. Make Literals

3.4 A Concrete Example

We can see how the algorithm works through a simple example from the family-relation domain. Suppose we want to learn the concept $uncle(X,Y)$, which is true iff X is an uncle of Y (blood uncle).

Suppose we have the following set of background facts (Figure 4):

1. $male(Bob)$, $male(Tom)$, $male(Tim)$
2. $female(Ann)$, $female(Mary)$, $female(Susan)$, $female(Betty)$, $female(Joyce)$
3. $parent(Tom, Mary)$, $parent(Tom, Betty)$, $parent(Tom, Bob)$, $parent(Mary, Ann)$,$parent(Joyce, Susan)$,$parent(Tom, Tim)$
4. $friend(Mary, Susan)$, $friend(Susan, Mary)$, $friend(Joyce, Betty)$, $friend(Betty, Joyce)$

and $\mathcal{P} = \{male/1, female/1, friend/2, parent/2\}$ (exactly in this order from left to right) is our set of predicate specifications. We will use $'+'$ to denote the output mode and $'-'$ the input mode here. The following is the set of mode specifications for each predicate specification:

$$male(-), female(-), parent(+,-), parent(-,+), friend(+,-), friend(-,+)$$

and the following is the set of type specifications for each predicate specification:

$$male(person), female(person), parent(person, person), friend(person, person)$$

Suppose we have this set of training examples:

1. Positive: $uncle(Bob, Ann)$
2. Negative:
   $uncle(Bob, Susan)$, $uncle(Betty, Ann)$, $uncle(Tim, Susan)$, $uncle(Tom, Betty)$
   $uncle(Susan, Betty)$, $uncle(Joyce, Ann)$, $uncle(Tim, Joyce)$, $uncle(Tom, Mary)$

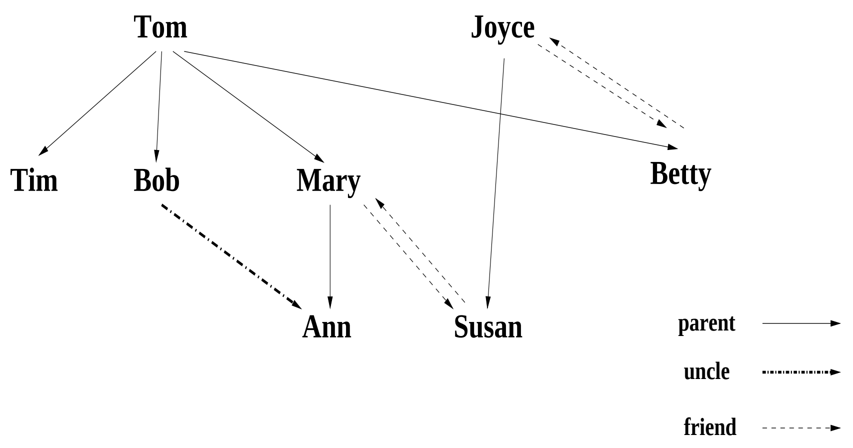


Fig. 4. A simple family relation domain

We present a trace of how our algorithm discovers a good clause, given a beam size and a recall bound of one, and a clause length of four. It starts by choosing a random seed example from the set of positive examples. This has to be *uncle*(*Bob*, *Ann*) since there is only one positive example. When generating refinements to a clause, it considers each predicate specification in $\mathcal{P}$ (from left to right). We will show the specialized clause before its set of cached proofs. The literal added to the clause currently being built is generated from the *new* ground atom added to the body of the cached proof of the current clause.

The algorithm starts with:

1. The most general clause which covers every pair of people: `uncle(X,Y) :- true`
2. The set of cached proofs for this clause: {`uncle(bob,ann) :- true`}
3. The empty bottom clause: `uncle(bob,ann) :- true`

It considers *male*/1 and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X)`
   `(m-est = 0.153)`
2. The set of cached proofs for this clause: {`uncle(bob,ann) :- male(bob)`}
3. The updated bottom clause: `uncle(bob,ann) :- male(bob)`

Next, the algorithm considers *female*/1, and the literal `female(Y)` is generated (in the same way as *male*/1), the new ground atom `female(ann)` is added to the current bottom clause. The specialized clause `uncle(X,Y) :- female(Y)` has an m-estimate of 0.111. Next, it considers *parent*/2 (using $parent(+,-)$) and generates the following:

1. The specialized clause: `uncle(X,Y) :- parent(Z,X)`
   `(m-est = 0.136)`

2. The set of cached proof of this clause: {`uncle(bob,ann) :- parent(tom,bob)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob)`

Similarly *parent*/2 (using *parent*$(+,-)$) is used to generate another specialized clause `uncle(X,Y) :- parent(W,Y)` (m-estimate $= 0.122$) using the ground atom `parent(mary,ann)`.

The predicate specification *friend*/2 was considered but no ground atom was found to satisfy all the refinement constraints; the link constraint could not be satisfied, because neither *Bob* nor *Ann* has a friend. There are totally four different refinements to the most general clause. The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X)$$

Since the beam size is just one, only this clause is retained in the beam. This clause is still covering negative examples: *uncle*(*Bob*, *Susan*), *uncle*(*Tom*, *Betty*), *uncle*(*Tim*, *Susan*), *uncle*(*Tim*, *Joyce*), and *uncle*(*Tom*, *Mary*). So, it still needs to be refined. Next, *male*/1 is considered but no ground atom is found to satisfy all the refinement constraints; the unique-literal constraint could not be satisfied (*male*(*Bob*) is already in the cached proof of the clause). The current bottom clause is `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob)`.

Next, it considers *female*/1 and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X),female(Y)`
   `(m-est = 0.153)`
2. The set of cached proof of this clause:
   {`uncle(bob,ann) :- male(bob),female(ann)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),`
   `parent(mary,ann)`

Next, it considers *parent*/2 (using *parent*$(+,-)$) and generates the following:

1. The specialized clause: `uncle(X,Y) :- male(X),parent(Z,X)`
   `(m-est = 0.204)`
2. The set of cached proof of this clause:
   {`uncle(bob,ann) :- male(bob),parent(tom,bob)`}
3. The updated bottom clause:
   `uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),`
   `parent(mary,ann)`

parent/2 (using *parent*$(+,-)$) can be used to generate another specialized clause `uncle(X,Y) :- male(X),parent(W,Y)` (m-estimate $= 0.175$) using the ground atom `parent(mary,ann)`.

The predicate specification *friend*/2 was considered but no ground atom was found to satisfy all the refinement constraints (the link constraint cannot be satisfied). There are totally three different refinements to $uncle(X,Y) \leftarrow male(X)$. The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X), parent(Z,X)$$

This clause still covers a non-empty set of negative examples:

$$uncle(Bob, Susan), uncle(Tim, Susan), uncle(Tim, Joyce).$$

The algorithm continues in exactly the same manner for the last two steps (omitted to save space). The clause $uncle(X,Y) \leftarrow male(X), parent(Z,X)$ has four different refinements. The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X), parent(Z,X), parent(W,Y)$$

which is still covering a non-empty set of negative examples: $uncle(Bob, Susan)$ and $uncle(Tim, Susan)$.

There are totally eight different refinements to

$$uncle(X,Y) \leftarrow male(X), parent(Z,X), parent(W,Y).$$

The clause with the best m-estimate is:

$$uncle(X,Y) \leftarrow male(X), parent(Z,X), parent(W,Y), parent(Z,W)$$

which covers all the positive examples and no negative examples. At this point, the algorithm has found the target concept. Both the bottom clause discovered and the consistent clause found are returned. Notice that the bottom clause found by BETH is:

```
uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),parent(mary,ann),
male(tom),female(mary),parent(tom,mary),friend(mary,susan),
friend(susan,mary)
```

whereas, PROGOL's bottom clause is:

```
uncle(bob,ann) :- male(bob),female(ann),parent(tom,bob),parent(mary,ann),
male(tom),female(mary),parent(tom,mary),friend(mary,susan),
friend(susan,mary),female(susan)
```

which is bigger than BETH's bottom clause.

4 Analysis

Let $\perp(b, n, \mathcal{P}, r, i)$ be the bottom clause constructed by BETH (Section 3) given the parameters b, n, $\mathcal{P}$, r, and i which are the beam width, the maximum clause length, the set of predicate specifications, the recall bound, and the variable depth bound respectively.

Theorem 1. Suppose B is a beam of clauses produced by BETH, for any clause $C \in B$, $C \preceq \perp(b, n, \mathcal{P}, r, i)$.

Proof. Suppose C_j is a clause in B such that $C_j = H \leftarrow L_1, \ldots, L_m$ where $m \leq n$. Each L_k is produced from a ground atom a_k and H from a particular seed example e. Obviously, $e \in \perp(b, n, \mathcal{P}, r, i)$. For any $k : 1 \leq k \leq m$, $a_k \in \perp(b, n, \mathcal{P}, r, i)$, since each ground atom satisfying all the refinement constraints is added to the current bottom clause and only ground atoms satisfying all the refinement constraints are used to make literals for any clause.

Thus, there's a substitution θ which satisfies C_j s.t. $C_j\theta = e \leftarrow a_1, \dots, a_m$. So, $C_j\theta \subseteq \perp(b, n, \mathcal{P}, r, i)$. And, we have $C_j \preceq \perp(b, n, \mathcal{P}, r, i)$. Hence we have $C \preceq \perp(b, n, \mathcal{P}, r, i)$ for any clause $C \in B$. $\Box$

Theorem 2. The worst case length of $\perp(b, n, \mathcal{P}, r, i)$ is $\mathcal{O}(bn|\mathcal{P}|r)$.
Proof. The maximum number of ground atoms that 1) satisfy the refinement constraints and 2) make literals observing the variable depth bound i for any clause in the search beam at the point the clause is being refined are $|\mathcal{P}|r$. Therefore, the maximum number of ground atoms satisfying the refinement constraints after adding n literals to the body of the most general clause are $n|\mathcal{P}|r$. Since there are at most b clauses in the search beam at any time, the maximum number of ground atoms satisfying the refinement constraints are $bn|\mathcal{P}|r$. Thus, the worst case complexity of the bottom clause $\perp(b, n, \mathcal{P}, r, i)$ is $\mathcal{O}(bn|\mathcal{P}|r)$. $\Box$

The length of PROGOL's bottom clause is $\mathcal{O}((r|\mathcal{M}|j^+j^-)^{ij^+})$; where $|\mathcal{M}|$ is the number of mode declarations, and $j^{+/-}$ are bounds on the number of $(+/-)$ types in a mode declaration [2] — which makes a hypothesis space doubly exponential w.r.t. i. Whereas the length of BETH's bottom clause is only linear w.r.t. n (which gives rise to a much smaller hypothesis space).

5 Experimental Evaluation

We compared our system, BETH, with two other leading ILP systems — ALEPH and mFOIL.

5.1 Domain

After the events of 9/11, the EELD project has been working on several Challenge Problems that are related to counter-terrorism. The problem that we choose to tackle is the detection of Murder-For-Hires (contract killings) in the domain of Russian Organized Crime. The data used in all EELD Challenge Problems include representations of people, organizations, objects, and actions and many types of relations between them. One can picture this data as a large graph of entities connected by a variety of relations. For our purposes, we represent these relational databases as facts in Prolog.

For the ease of generating large quantities of data, and to avoid violating privacy, the program currently only uses synthetic data generated by a simulator. The data for the Murder-For-Hire problem was generated using a Task-Based (TB) simulator developed by Information Extraction and Transport Incorporated (IET). The TB simulator outputs case files, which contain complete and unadulterated descriptions of murder cases. These case files are then filtered for observability, so that facts that would not be accessible to an investigator are eliminated. To make the task more realistic, this data is also corrupted, e.g., by misidentifying role players or incorrectly reporting group memberships. This filtered and corrupted data form the evidence files. In the evidence files, facts about each event are represented as ground facts, such as:

```
murder(Murder714)
```

```
perpetrator(Murder714, Killer186)
crimeVictim(Murder714, MurderVictim996)
deviceTypeUsed(Murder714, PistolCzech)
```

The synthetic dataset that we used consists of 632 murder events. Each murder event has been labeled as either a positive or negative example of a murder-for-hire. There are 133 positive and 499 negative examples in the dataset. Our task was to learn a theory to correctly classify an unlabeled event as either a positive or negative instance of murder-for-hire. The amount of background knowledge for this dataset is extremely large; consisting of 52 distinct predicate names, and 681,039 background facts in all.

5.2 Results

The performance of each of the ILP systems was evaluated using 6-fold cross-validation. The total number of Prolog atoms in the data is so large that running more than six folds is not feasible.[4] The data for each fold was generated by separate runs of the TB simulator. The facts produced by one run of the simulator, only pertain to the entities and relations generated in that run; hence the facts of each fold are unrelated to the others. For each trial, one fold is set aside for testing, while the remaining data is *combined* for training. To test performance on varying amounts of training data, learning curves were generated by testing the system after training on increasing subsets of the overall training data. Note that, for different points on the learning curve, the background knowledge remains the same; only the number of positive and negative training examples given to the system varies.

We compared the three systems with respect to accuracy and training time. Accuracy is defined as the number of correctly classified test cases divided by the total number of test cases. The training time is measured as the CPU time consumed during the training phase. All the experiments were performed on a 1.1 GHz Pentinum with dual processors and 2 GB of RAM. BETH and mFOIL were implemented in Sicstus Prolog version 3.8.5 and ALEPH was implemented in Yap version 4.3.22. Although different Prolog compilers were used, the Yap Prolog compiler has been demonstrated to outperform the Sicstus Prolog compiler, particularly in ILP applications [4].

In our experiments, we used a beam width of 4 for BETH and mFOIL; and limited the number of search nodes in ALEPH to 5000. We used m-estimate ($m = 2$) as a search heuristic for all ILP algorithms. The clause length was limited to 10 and the variable depth bound to 5 for all systems. The recall bound was limited to 1 for BETH and ALEPH (except for some mode declarations it was set to '*'). We modified mFOIL to be constrained by the maximum clause length and the variable depth bound, to ensure that it terminates. We refer to this version of mFOIL as *Bounded* mFOIL. All the systems were given 1 second of CPU time to compute the set of examples covered by a clause. If a specialized

[4] The maximum number of atoms that the Sicstus Prolog compiler can handle is approximately a quarter million.

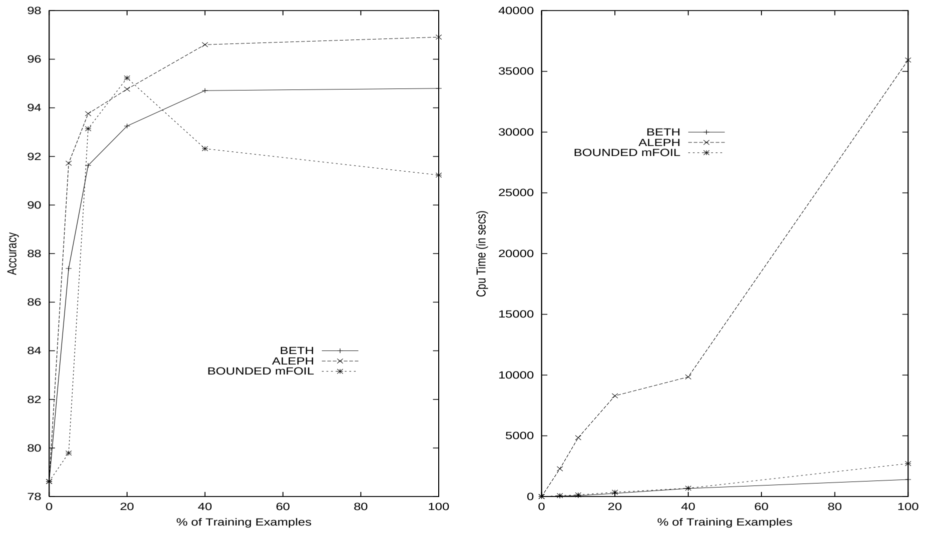


Fig. 5. Performance of the systems versus the percentage of training examples given

System	*Accuracy*	*CPU Time (mins)*	*# of Clauses*	*Bottom Clause Size*
Beth	94.80% (+/- 2.3%)	23.39 (+/- 4.26)	4483	34
Aleph	96.91% (+/- 2.8%)	598.92 (+/- 250.00)	63334	4061
mFoil	91.23% (+/- 4.8%)	45.28 (+/- 5.40)	112904	n/a

Table 2. Results on classifying *murder-for-hire* events given all the training data. *# of Clauses* is the total number of clauses tested; and *Bottom Clause Size* is the average number of literals in the bottom clause constructed for each clause in the learned theory. The 90% confidence intervals are given for test *Accuracy* and *CPU time.*

clause took more time than allotted, the clause was ignored; although the time it took to create the clause is still recorded.

The results of our experiments are summarized in Figure 5. A snapshot of the performance of the three ILP systems given 100% of the training examples is shown in Table 2. The following is a sample rule learned by Beth:

```
murder_for_hire(A):- murder(A), eventOccursAt(A,H),
  geographicalSubRegions(I,H), perpetrator(A,B),
  recipientOfinfo(C,B), senderOfinfo(C,D), socialParticipants(F,D),
  socialParticipants(F,G), payer(E,G), toPossessor(E,D).
```

This rule covered 9 positive examples and 3 negative examples. The rule can be interpreted as: *A* is a murder-for-hire, if *A* is a murder event, which occurs in a city in a subregion of Russia, and in which *B* is the perpetrator, who received information from *D*, who had a meeting with and received some money from *G*.

5.3 Discussion of Results

On the full training set, BETH trains 25 times faster than ALEPH while losing only 2 percentage points in accuracy and it trains twice as fast as mFOIL while gaining 3 percentage points in accuracy. Therefore, we believe that its integration of top-down and bottom-up search is an effective approach to dealing with the problem of scaling ILP to large examples. The learning curves further illustrate that the training time of BETH grows slightly slower than that of mFOIL, and considerably slower than that of ALEPH.

The large speedup over ALEPH is explained by the theoretical analysis on the complexity of the bounds on the search space, i.e. the different sizes of the bottom clauses they construct. The size of the bottom clause for BETH is only linear w.r.t. n compared to that of ALEPH which is exponential w.r.t. to i ($i \leq n$) even for small i. As a result, ALEPH's search space is much larger than BETH's. ALEPH's bottom clause was on average 119x larger than BETH's and the total number of clauses it constructed was 14x larger, although a theory of similar accuracy was learned.

Systems like BETH and ALEPH construct literals based on actual ground atoms in the background knowledge, guaranteeing that the specialized clause covers at least the seed example. On the other hand, mFOIL generates more literals than necessary by enumerating all possible combination of variables. Some such combinations make useless literals; adding any of them to the body of the current clause makes specialized clauses that do not cover any positive examples. Thus, mFOIL wastes CPU time constructing and testing these literals. Since the average predicate arity in the EELD data was small (2), the speedup over mFOIL was not as great, although much larger gains would be expected for data that contains predicates with higher arity.

Nevertheless, searching a smaller space comes at the cost of spending more time generating each literal for refining a clause. In ALEPH, all the necessary ground literals are generated before the search starts, while BETH must spend time computing a set of ground atoms satisfying the refinement constraints on literal generation, resulting in fewer clauses tested per unit time compared to both ALEPH and mFOIL.

From the experimental results obtained, we can conclude that 1) an approach like BETH, which emphasizes searching a much smaller space over testing hypotheses at a higher rate, can outperform (in terms of efficiency) an approach like PROGOL/ALEPH, which trades off the two factors the other way around, and 2) using ground atoms directly avoids testing useless literals, improving training time over a purely top-down approach like mFOIL.

6 Conclusions

An important under-studied aspect of scaling to large databases in multi-relational data mining concerns the size of examples rather than their number. For ILP methods, this issue involves scaling to large numbers of connected background facts associated with each example or set of examples. We have developed a new ILP algorithm that integrates top-down and bottom-up search in order to more efficiently learn in the presence of large sets of background facts. Challenge problems constructed for DARPA's program on Evidence Extraction and Link Discovery concern identifying potential threatening activities in large amounts of heterogeneous, multi-relational data. These problems contain relatively modest numbers of examples but involve very large sets of background facts. Experimental results on these problems demonstrate that our new hybrid approach substantially decreases training time compared to existing ILP methods.

7 Acknowledgments

This research is sponsored by DARPA and managed by Rome Laboratory under contract F30602-01-2-0571. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency, Rome Laboratory, or the United States Government.

References

1. R. J. Mooney, P. Melville, L. R. Tang, J. Shavlik, I. de Castro Dutra, D. Page, and V. S. Costa. Relational data mining with inductive logic programming for link discovery. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, 2002.
2. S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.
3. C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*, pages 63–92. Academic Press, London, 1992.
4. V. Santos Costa. Optimising bytecode emulation for Prolog. In *LNCS 1702, Proceedings of PPDP'99*, pages 261–267. Springer-Verlag, September 1999.
5. F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, 2002.
6. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, Chichester, 1994.
7. R. J. Quinlan. *Learning Logical Definitions from Relations. Machine Learning*, 5(3):239–266, 1990.
8. M. J. Pazzani and D. F. Kibler. *The Utility of Background Knowledge in Inductive Learning. Machine Learning*, 9:57–94, 1992.

Efficient Multi-relational Classification by Tuple ID Propagation

Xiaoxin Yin, Jiawei Han and Jiong Yang

Department of Computer Science
University of Illinois at Urbana-Champaign
{xyin1, hanj, jioyang}@uiuc.edu

Abstract. Most of today's structured data is stored in relational databases. In contrast, most classification approaches only apply on single "flat" data relations. And it is usually difficult to convert multiple relations into a single flat relation without losing essential information. Inductive Logic Programming approaches have proven effective with high accuracy in multi-relational classification. Unfortunately, they usually suffer from poor scalability with respect to the number of relations and the number of attributes in the database. In this paper we propose *CrossMine*, an efficient and scalable approach for multi-relational classification. It uses a novel method *tuple ID propagation* to perform virtual joins, so as to achieve high classification accuracy and high efficiency on databases with complex schemas. We present experimental results on two real datasets to show the performance of our approach.

1 Introduction

Most of today's structured data is stored in relational databases. Many important classification approaches, such as neural networks [6] and support vector machines [2], can only be applied to data represented by single flat data relations. And it is usually difficult to convert a relational database into a single flat relation without losing essential information.

Another category of approaches to multi-relational classification is Inductive Logic Programming (ILP) [1, 7, 8, 10]. The ILP classification approaches aim at finding hypotheses of certain format that can predict class labels of examples, based on background knowledge. They achieve good classification accuracy in multi-relational classification. However, most ILP approaches are not scalable with respect to the number of relations and attributes in the database. We use FOIL [10] as an example. FOIL constructs conjunctive rules that distinguish positive examples from negative ones. It adds predicates one by one when building rules and many predicates are evaluated at each step. To evaluate a predicate, it generates a new rule by appending the predicate to the current rule, and evaluates the new rule, by finding out the number of positive and negative examples satisfying the rule. The whole procedure is very time-consuming when the number of relations is large or the number of possible predicates is large.

In a database for multi-relational classification, there is one target relation R_t, whose tuples are called target tuples. Each target tuple is associated with a class label. Other relations are non-target relations and may contain relevant information for classification. When building a classifier for a database with complex schema, the search space is very large. Therefore, it is crucial to prune the search space to achieve both high accuracy and high efficiency. Until now, there is not any accurate, efficient and scalable approach for multi-relational classification.

In this paper we propose *CrossMine*, a scalable and accurate approach for multi-relational classification. The basic idea of CrossMine is to virtually join relations to find good predicates. For any non-target relation R that can be joined to R_t, we can evaluate every predicate on R based on the joined relation of R and R_t. CrossMine utilizes a novel method called *tuple ID propagation*, which enables virtually joining the relations and avoids the high cost of physical joins.

CrossMine uses rules for classification. It uses a sequential covering algorithm that is similar to FOIL, which repeatedly constructs rules and removes positive target tuples covered by each rule. To construct a rule, it repeatedly searches for the best predicate and appends it to the current rule.

This paper is organized as follows. In section 2 we introduce the related works. Section 3 introduces the idea of tuple ID propagation. We describe the implementation of our approach in section 4. Experimental results are presented in section 5 and this paper is concluded in section 6.

2 Related Works

ILP is the most important category of approaches in multi-relational classification. The well known ILP systems include FOIL [10], Golem [8], and Progol [7]. FOIL is a top-down learner, and is considered to be an efficient ILP approach. It builds rules that cover many positive examples and few negative ones. Golem is a bottom-up learner, which performs generalization from most specific rules. Progol uses a combined search strategy. A more recent approach is TILDE [1], which uses the idea of C4.5 [11] and inductively constructs decision trees. TILDE is more efficient than traditional ILP approaches due to the divide-and-conquer nature of decision tree algorithm. Another approach for constructing multi-relational decision tree is presented in [5].

Besides ILP, probabilistic approaches [12, 4, 9] are also very important for multi-relational classification and modeling. The most famous one is probabilistic relational models (PRM's) [12, 9], which is an extension of Bayesian networks for handling relational data. PRM's can integrate the advantages of both logical and probabilistic approaches to knowledge representation and reasoning. In [9] the authors propose an approach that integrates ILP and statistical modeling for document classification and retrieval.

Another approach for modeling or classifying relational data is through frequent pattern mining or association rule mining. In [13] an approach is presented for frequent pattern mining in graphs, which can be applied to relational data. In [3] the authors propose an approach for association rule mining in relational databases.

Here we take FOIL as a typical example and show its working procedure. It is a sequential covering algorithm that builds rules one by one. After building each rule, all positive target tuples satisfying that rule are eliminated. To build a rule, predicates are added one by one. At every step, every possible predicate is evaluated and the best one is added to the current rule. To evaluate a predicate *p*, *p* is appended to the current rule *r* to get a new rule *r'*. Then it constructs a new dataset which contains all positive and negative target tuples satisfying *r'*, together with the relevant non-target tuples. Then *p* is evaluated based on the number of positive and negative target tuples satisfying *r'*.

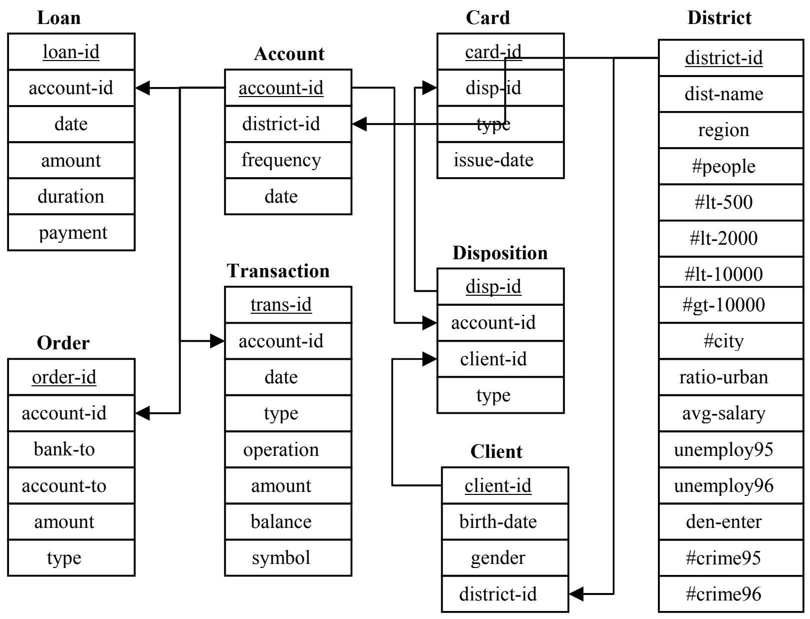


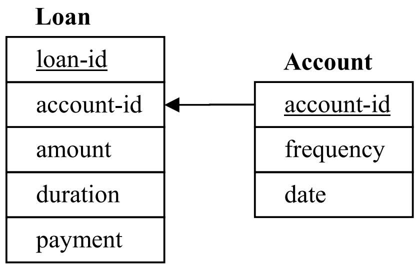
Fig. 1. A sample database from PKDD CUP 99.

We illustrate the FOIL algorithm with the database shown in figure 1. The arrows go from primary keys to the corresponding foreign-keys. The target relation is Loan. Each tuple in Loan is either positive or negative, indicating whether the loan is paid in time. Initially FOIL uses an empty rule as the current rule. Then it evaluates all predicates that are directly related to the target, such as "Loan(*L*, ?, ?, ?, 12, ?)" (the duration of loan *L* is 12 months), and "Loan(*L*, *A*, ?, ?, ?, ?)" (loan *L* is associated with some account *A*). After the *foil gain* of each of these predicates are computed, the best predicate is added to the current rule. After adding this predicate, FOIL searches for further predicates. It will stop and output the current rule when the stop condition is met. Then it will start constructing the second rule, and so on.

FOIL is not efficient because it has to evaluate too many predicates in the whole procedure, and it is time-consuming to evaluate each predicate because a new dataset needs to be built. Therefore, FOIL is not scalable with respect to the number of relations and the number of attributes in databases

3 Tuple ID Propagation

In this section we explain the idea of tuple ID propagation and the method of finding good predicates based on propagated IDs. Tuple ID propagation is a method for virtually joining non-target relations with the target relation. It is a flexible and efficient method and it avoids the high cost of physical join.



Account		
account-id	frequency	date
124	monthly	960227
108	weekly	950923
45	monthly	941209
67	weekly	950101

Loan					
loan-id	account-id	amount	duration	payment	
1	124	1000	12	120	+
2	124	4000	12	350	+
3	108	10000	24	500	-
4	45	12000	36	400	-
5	45	2000	24	90	+

Fig. 2. An example database.

3.1 Basic Definitions

Let us take the sample database in figure 2 as an example. Suppose the current rule is empty and we want to find the best predicate. We first define the measure for the goodness of a predicate. We use the measure *foil gain*, which is used by FOIL.

For a certain rule *r*, we use *P*(*r*) and *N*(*r*) to denote the number of positive and negative target tuples satisfying *r*. Suppose the current rule is *r*. We use *r*+*p* to denote the rule constructed by appending predicate *p* to *r*. The foil gain of predicate *p* is,

$$foil\_gain(p) = P(r+p) \times \left[-\log \frac{P(r)}{P(r)+N(r)} + \log \frac{P(r+p)}{P(r+p)+N(r+p)} \right].$$

Intuitively *foil_gain*(*p*) represents the total number of bits saved in representing positive target tuples by appending *p* to the current rule.

By foil gain we can measure the goodness of each predicate in Loan relation. For a predicate in Account relation, such as “Account(*A*, ?, monthly, ?)”, we need to define what is the meaning of “a target tuple satisfies a rule containing this predicate”. Suppose rule *r* = “Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, ?, monthly, ?)”. We say that a tuple *t* in Loan relation satisfies *r*, if and only if **any** tuple in Account relation that is joinable with *t* has value “monthly” on the attribute of frequency. In this example, there are two tuples {124, 45} in Account relation that satisfy the predicate “Account(*A*, ?, monthly, ?)”. And there are four tuples {1, 2, 4, 5} in Loan relation that satisfy this rule.

In the above example a tuple in Loan can only be associated with one tuple in Account. In fact it may be associated with more than one tuple in other relations such as Order and Client (see figure 1). This depends on whether the joined attributes (account-id in this example) are primary keys or foreign-keys in the joined relations.

3.2 Search for Predicates by Joins

Consider the sample database in figure 2. For predicates in Loan relation, we can compute their foil gain directly. For predicates in Account relation, we can find their foil gain in the following way. Set the current rule *r* to “Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?)”. For each predicate *p* in Account relation, we add it to *r*, and find out all positive and negative target tuples satisfying *r*. We take *p* = “Account(*A*, ?, monthly, ?)” as an example. The rule *r*+*p* is “Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, ?, monthly, ?)”, which can be converted into a SQL query “SELECT L.loan-id FROM Loan L, Account A WHERE L.account-id = A.account-id AND A.frequency = ‘monthly’”. If the database is stored in main memory as arrays, we can simulate the process of executing a SQL query to find out *P*(*r*+*p*) and *N*(*r*+*p*).

The disadvantage of this approach is that, it needs to do much computation for each predicate. If the current rule is “Loan(*L*, +) :- Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, ?, monthly, ?)”, then for each predicate in a relation joinable to Account, we need to execute a SQL query with two selection and two join operations, or do similar things in main memory. This is time consuming and unscalable.

One approach to solving this problem is to do the join once and compute the foil gain of all predicates. For the database in figure 2, we can join the two relations, as in table 1.

Table 1. The join of Loan relation and Account relation.

Loan ∞ Account							
loan-id	account-id	amount	duration	payment	frequency	date	
1	124	1000	12	120	monthly	960227	+
2	124	4000	12	350	monthly	960227	+
3	108	10000	24	500	weekly	950923	-
4	45	12000	36	400	monthly	941209	-
5	45	2000	24	90	monthly	941209	+

With the joined relation, we can compute the foil gain of every predicate in both relations. To compute the foil gain of all predicates on a certain attribute (such as "Account(*A*, ?, monthly, ?)"), we only need to scan the corresponding column in the joined relation once. Another advantage of this approach is that, it can handle continuous attribute very well. Suppose we want to find the best predicate on Account.date. We can first sort that column. Then we iterate from the smallest value to the largest value. For each value *d*, we compute the foil gain of two predicates "date ≤ *d*" and "date ≥ *d*".

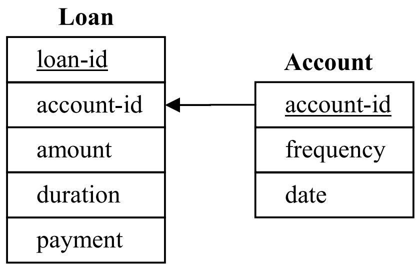
The advantage of this approach is that, we need to do the join only once. The disadvantage is that, it is almost impossible to join all relations because the joined relation is too large. There are usually tens of relations, each having a few keys or foreign-keys. There are usually many join paths from the target relation to each non-target relation. And the final joined relation might be thousands of times larger than the original database.

3.3 Tuple ID Propagation

In this section we describe how to simulate the procedure in section 3.2 by propagating tuple IDs of the target relation. Suppose the primary key of the target relation is an attribute of integers, which represent the IDS of the target tuples. We use the ID of each target tuple to represent that tuple. Consider the sample database in figure 3.

We propagate the tuple IDs from Loan relation to Account relation. Or to say, for each tuple *t* in the Account relation, we store the IDs of the target tuples associated with *t* by natural join.

Definition 1: **ID propagation**. Suppose we have relation R_1 and R_2, which can be joined by attributes R_1.A and R_2.A. Each tuple in R_1 is associated with some IDs in the target relation. For each tuple *t* in R_2, we set *t*'s IDs to be the union of {*u*'s ID | *u*єR_1, *u*.A=*t*.A}.



Loan					
loan-id	account-id	amount	duration	payment	
1	124	1000	12	120	+
2	124	4000	12	350	+
3	108	10000	24	500	-
4	45	12000	36	400	-
5	45	2000	24	90	+

Account				
account-id	frequency	date	IDs	Class labels
124	monthly	960227	1, 2	2+, 0−
108	weekly	950923	3	0+, 1−
45	monthly	941209	4, 5	1+, 1−
67	weekly	950101	--	0+, 0−

Fig. 3. An example for ID propagation.

By the propagated IDs on Account relation, we can compute the foil gain of every predicate in this relation, according to the following theorem.

Theorem 1: Suppose we have relations R_1 and R_2, which can be joined by attribute R_1.A and R_2.A. R_1 is the target relation. All tuples in R_1 satisfy the current rule (all others have been eliminated). If R_1's IDs are propagated to R_2, then using these propagated IDs, the foil gain of every predicate in R_2 can be computed.
Proof:

Given the current rule *r*, for a predicate *p* in R_2, such as R_2.B=*b*, its foil gain can be computed based on *P*(*r*), *N*(*r*), *P*(*r*+*p*) and *N*(*r*+*p*). *P*(*r*) and *N*(*r*) should have been computed during the process of building the current rule. *P*(*r*+*p*) and *N*(*r*+*p*) can be acquired in the following way: (1) find all tuples *t'* in R_2 that *t'*.B=*b*; (2) find all tuples in R_1 that can be joined with any tuple found in (1); (3) count the number positive and negative tuples found in (2). ■

For example, suppose the current rule is "Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?)". For predicate "Account(*A*, ?, monthly, ?)", we can first find out tuples in Account relation that satisfy this predicate, which are {128, 45}. Then we can find out tuples in Loan relation that can be joined to these two tuples, which are {1, 2, 4, 5}. We maintain a global table of the class labels of each target tuple. From this table, we can know that {1, 2, 4, 5} contains three positive and one negative target tuples. By this information we can easily compute the foil gain of predicate "Account(*A*, ?, monthly, ?)".

Please notice that we cannot compute the foil gain only from the class labels of each tuple in Account relation (see figure 3). Suppose some tuple *t* in Loan relation can be joined with two tuples in Account relation. If we only use the number of positive and negative target tuples associated with every tuple in Account relation to compute the foil gain, the tuple *t* will be counted twice. And we cannot get the exact number of positive and negative target tuples satisfying a rule.

In fact tuple ID propagation is a way to do virtual join. Instead of physically joining the two relations, we virtually join them by attaching the tuple IDs of the target relation to the tuples of another relation. In this way we can compute the foil gain as we are doing physical join.

We have seen that we can propagate IDs from the target relation to relations directly joinable with it. We can also propagate IDs from one non-target relation to another non-target relation, according to the following theorem.

Theorem 2: Suppose we have relations R_2 and R_3, which can be joined by attribute R_2.A and R_3.A. All tuples in R_2 satisfy the current rule (all others have been eliminated). For each tuple *t* in R_2, there is a set of IDs representing the tuples in the target relation that can be joined with *t* (using the join path specified in the current rule). By propagating the IDs from R_2 to R_3 through the join R_2.A=R_3.A, for each tuple *t'* in R_3, we can get the set of IDs representing tuples in the target relation that can be joined with *t'* (using the join path in the current rule, plus the join R_2.A=R_3.A).
Proof:

Suppose the current rule is *r*. Suppose a tuple *t* in R_2 has the set of IDs $s(t) = \{i_1, i_2, \ldots, i_k\}$. For a tuple *t'* in R_3, suppose it can be joined with $t_1, t_2, \ldots, t_m$ in R_2. For some predicate *p'* in R_3, if *t'* satisfies *p'*, then $t_1, t_2, \ldots, t_m$ will satisfy the rule (*r*+*p'*). Then all target tuples associated with any one of $t_1, t_2, \ldots, t_m$ will satisfy the rule (*r*+*p'*), and they are all target tuples satisfying this rule. Therefore, if we propagate the IDs from R_2 to R_3, we will have the correct IDs associated with each tuple in R_3, with which we can compute the foil gain of every predicate in R_3. ■

Please refer to the example database in figure 4. Suppose the current rule is "Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, *D*, monthly, ?)". And we want to evaluate the foil gain of predicate "District(*D*, Urbana, ?)". We can propagate the IDs from Account relation to District relation. It means that for each tuple *t'* in District relation, suppose it can be joined with $t_1, t_2, \ldots, t_m$ in Account relation, then we set the IDs of *t'* to be the union of the IDs of $t_1, t_2, \ldots, t_m$. For each tuple t_i in $t_1, t_2, \ldots, t_m$, the associated IDs show the tuples in target relation that can be joined to t_i, based on the join path specified in the current rule. To evaluate the predicate "District(*D*, Urbana, ?)", we need to append it to the current rule to get the new rule "Loan(*L*, +) :- Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, ?, monthly, ?), District(*D*, Urbana, ?)". If we propagate IDs from Account relation to District relation, then for each tuple *t'* in District relation, the IDs on *t'* are actually the IDs of tuples in the target relation that are joinable to *t'*. And by these IDs we can compute the foil gain of this predicate.

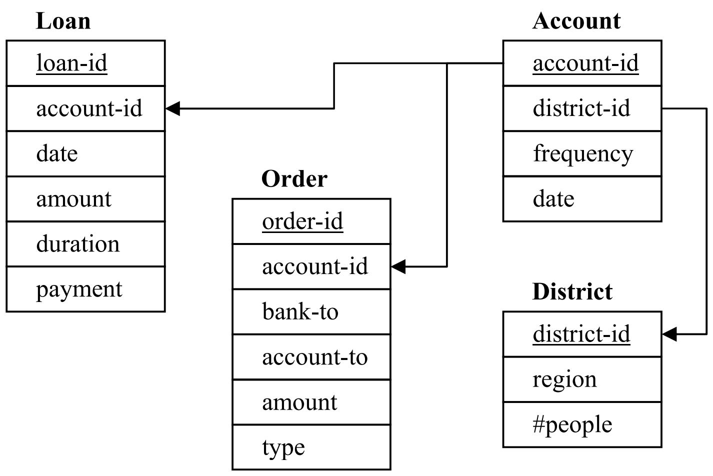


Fig. 4. Another sample database.

Sometimes too many IDs may be propagated to each tuple in a relation, which makes it hard to limit the time/space complexity of the algorithm. In our algorithm we give the following constraint for ID propagation. The total number of IDs associated with tuples in a relation must be smaller than a certain number of times the length of that relation, or a certain number of times the length of the target relation. Or to say, if too many IDs are to be associated with a relation, we will stop exploring along that relation.

4 Implementation

4.1 Data Representation

The database contains a set of relations. One of them is the target relation, with class labels on its tuples. The other relations have no class labels. But we may propagate class labels to their tuples. Each relation may have one primary key and several foreign keys, each pointing to the primary key of some other relation.

We only use joins between keys and foreign-keys (including joins between two foreign-keys which point to the same primary key). We ignore other joins because they do not represent strong semantic relationships between entities in the database.

We want to find rules that can distinguish the positive target tuples from negative ones. Each rule is a list of predicates, with a class label associated. There are two types of predicates. The first type of predicates specify how to do join or tuple ID propagation. For example, "Loan(*L*, *A*, ?, ?, ?, ?)" specifies we should join the Account relation with Loan relation (in our case, propagate the tuple IDs of Loan relation to Account relation). The second type is some constraint on a certain attribute of a certain relation. For example, "Account(*A*, ?, monthly, ?)" specifies that the tuples should have value "monthly" on attribute frequency in Account relation.

We define a *predicate pair* to be the combination of a first-type predicate p_1 and a second-type predicate p_2. In the predicate pair, the first predicate specifies how we can propagate IDs to a relation, and the second predicate specifies the constraint on that relation. For example, suppose the current rule is empty, and the predicate pair "Loan(*L*, *A*, ?, ?, ?, ?), Account(*A*, ?, monthly, ?)" specifies that we propagate the tuple IDs from Loan relation to Account relation, and in Account relation the frequency attribute must have value "monthly".

4.2 Learning Algorithm

Given a dataset which contains one target relation and some other relations, CrossMine builds a classifier containing a set of rules, each of which is the conjunction of a set of predicates associated with a class label. The overall idea is to repeatedly build rules. After each rule is built, remove all positive target tuples satisfying it. The algorithm is shown in figure 5.

```
Algorithm Find-Rules
Input: a relational database.
Output: a set of rules that distinguish the positive target tuples.
Procedure:
    Rule set $R = \Phi$;
    do
        Rule $r$ = Find-A-Rule( );
        Add $r$ to the $R$;
        Remove all positive target tuples that satisfy $r$;
    while(there are more than 10% positive target tuples left)
    return $R$;
```

Fig. 5. Algorithm *Find-Rules*.

To build a rule, we repeatedly search for the best predicate or predicate pair, and append it to the current rule. We define a relation to be active if it appears in the current rule, or it is the target relation. We require that every active relation has the correct propagated IDs on every tuple. Before building a rule, only the target relation is active. The algorithm is shown in figure 6.

```
Algorithm Find-A-Rule
Input: a relational database.
Output: a rule that distinguishes the positive target tuples.
Procedure:
    Rule r = empty rule;
    set target relation to be active
    do
        Predicate pair p = Find-Best-Predicate( );
        if foil gain of p is less than MIN_FOIL_GAIN
        then break;
        else
            r = r + p;
            remove all positive and negative tuples not satisfying r;
            if p is on an inactive relation
            then set that relation to be active;
            update IDs on every active relation;
    while( r.length < MAX_RULE_LENGTH )
    return r;
```

Fig. 6. Algorithm *Find-A-Rule*.

In algorithm *Find-A-Rule* we use procedure *Find-Best-Predicate*() to find the best predicate or predicate pair. Its procedure is described as follows.

(1) Find the best predicate in every active relation, by computing the foil gain of every predicate in every active relation.
(2) For every relation *R* that can be joined with some active relation, propagate IDs to *R* from some active relation, and find the best predicate on *R*.
(3) From the predicates and predicate pairs found in the previous two steps, find the best one and append it to the current rule.

After we have found the best predicate or predicate pair, if its foil gain is greater than the threshold, we append it to the current rule; otherwise we stop and output this rule.

5 Experimental Results

We test our approach on two datasets. The first one is the financial dataset used in PKDD CUP 1999. The schema is described in figure 1. The number of attributes and tuples of each relation is shown in table 2.

Table 2. Specification of financial dataset.

Relation name	Account	Client	Disposition	Order	Loan	Card	Dis-trict	Trans
# tuples	4500	5369	5369	6471	400	892	77	52904
# attr	4	4	4	6	6	4	16	8

We modified the dataset a little by shrinking the Trans relation which is extremely large, and removing some positive target tuples in the Loan relation to make the number of positive and negative target tuples more balanced.

Loan is the target relation. The class label indicates whether each loan is paid on time. Among the 400 target tuples, there are 324 positive ones and 76 negative ones.

The second dataset is Mutagenesis dataset, whose specification is shown in table 3.

Table 3. Specification of Mutagenesis dataset.

Relation name	Molecule	Atom	Molecule-Atom	Bond
# tuples	188	4893	4893	5244
# attributes	5	4	2	3

We compare our approach with FOIL and TILDE. We got the source code from the authors of the two approaches. CrossMine and FOIL run on a PC with P4 1.7GHz CPU running Windows 2000. TILDE runs on a Sun Blade 1000 running Solaris. Ten-fold experiment is used and the average running time of each fold is reported. The performances of the approaches are shown in table 4.

Table 4. Performances of CrossMine, FOIL and TILDE.

	Financial dataset		Mutagenesis dataset	
	Accuracy	Training Time	Accuracy	Training Time
CrossMine	90.7%	15.3 sec	87.7%	0.83 sec
FOIL	74.0%	3338 sec	79.7%	1.65 sec
TILDE	81.3%	2429 sec	89.4%	25.6 sec

6 Conclusions

Multi-relational classification is a very important research area because of the popularity of relational database. Unfortunately most existing approaches are not scalable with respect to the number of relations and the complexity of the database schema. In this paper we propose CrossMine, an efficient approach that will promisingly solve the scalability problem in multi-relational classification. It uses tuple ID propagation to do virtual join, so as to find optimal predicates for building rules efficiently. In our experiments it is shown that CrossMine is much more efficient than two existing approaches.

Besides classification, there are many other important tasks in relational databases, such as object matching, schema matching, and data cleaning. It is quite possible that we can adapt our approach to these tasks, using the idea of tuple ID propagation.

References

1. H. Blockeel, L. De Raedt and J. Ramon. Top-down induction of logical decision trees. In *Proc. of the Fifteenth Int. Conf. of Machine Learning*, Madison, WI, 1998.
2. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 1998.
3. L. Dehaspe and H. Toivonen. Discovery of Relational Association Rules. In *Relational Data Mining*, Springer-Verlag, 2000.
4. L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proc. 18th International Conf. on Machine Learning*, Williamtown, MA, 2001.
5. H. A. Leiva. MRDTL: a multi-relational decision tree learning algorithm. *M.S. thesis*, Iowa State Univ., 2002.
6. T. Mitchell. Machine Learning. McGraw Hill, 1996.
7. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 1995.
8. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. of First Conf. on Algorithmic Learning Theory*, Tokyo, Japan, 1990.
9. A. Popescul, L. Ungar, S. Lawrence, and M. Pennock. Towards Structural Logistic Regression: Combining Relational and Statistical Learning. In *Proc. of Multi-Relational Data Mining Workshop*, Alberta, Canada, 2002.
10. J. R. Quinlan. FOIL: A midterm report. In *Proc. of the sixth European Conf. on Machine Learning*, Springer-Verlag, 1993.
11. J. R. Quilan. C4.5: Programs for Machine Learning. In *Morgan Kaufmann series in machine learning*, Morgan Kaufmann, 1993.
12. B. Taskar, E. Segal, and D. Koller. Probabilistic Classification and Clustering in Relational Data. in *Proc. of 17th Int. Joint Conf. on Artificial Intelligence*, Seattle, WA, 2001.
13. X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *Proc. of 2002 Int. Conf. on Data Mining*, Maebashi, Japan, 2002.

Constraint-Based Relational Subgroup Discovery

Filip Železný[1], Nada Lavrač[2], and Sašo Džeroski[2]

[1] Czech Technical University, Prague, Czech Republic
zelezny@fel.cvut.cz
University of Wisconsin, Madison, USA
zelezny@biostat.wisc.edu
[2] Department of Intelligent Systems, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
nada.lavrac@ijs.si
saso.dzeroski@ijs.si

Abstract. Relational rule learning is typically used in solving classification and prediction tasks. However, it can also be adapted to the description task of subgroup discovery. This paper takes a propositionalization approach to relational subgroup discovery (RSD), based on adapting rule learning and first-order feature construction, applicable in individual-centered domains. It focuses on the use of constraints in RSD, both in feature construction and rule learning. We apply the proposed RSD approach to the Mutagenesis benchmark known from relational learning and a real-life telecommunications dataset.

1 Introduction

Developments in *descriptive induction* [24, 33] have recently gained much attention of researchers developing rule learning algorithms. These involve mining of association rules (e.g., the APRIORI association rule learning algorithm [1]), clausal discovery (e.g., the CLAUDIEN system [24, 25]), subgroup discovery (e.g., the MIDOS [31, 32] and EXPLORA [15] subgroup discovery systems) and other approaches to non-classificatory induction aimed at finding interesting patterns in data.

In this paper, we consider the task of subgroup discovery: given a population of individuals and a specific property of those individuals that we are interested in, find population subgroups that are statistically 'most interesting', e.g., are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest. We restrict ourselves to class-labeled data in our approach, with the class attribute being the property of interest. While the goal of standard rule learning is to generate models, one for each class, inducing class characteristics in terms of properties occurring in the descriptions of training examples, in contrast, subgroup discovery aims at discovering individual 'patterns' of interest.

Our approach adapts classification rule learning to relational subgroup discovery, which is achieved by (a) propositionalization through first-order feature construction, (b) incorporation of example weights into the covering algorithm, (c) incorporation of example weights into the weighted relative accuracy search heuristic, (d) probabilistic classification based on the class distribution of covered examples by individual rules, and (e) area under the ROC curve rule set evaluation. The main advantage of the proposed approach is that each induced rule with a high weighted relative accuracy represents a 'chunk' of knowledge about the problem, due to the appropriate tradeoff between accuracy and coverage, achieved through the use of the weighted relative accuracy heuristic.

We apply language constraints to define the language of possible subgroup descriptions. These are applied both in feature generation and rule induction. We apply evaluation constraints during rule induction to select the (most) interesting rules/subgroups.

We believe that the combination of the above mentioned strategies controlled by constraints is an original approach to relational subgroup discovery, altough previous work exists incorporating some of the techniques mainly in classification rule discovery; eg. rule induction with constraints in relational domains including propositionalization [2, 3], or using rule sets to maximize ROC performance [10].

This paper is organized as follows. Section 2 presents the underlying principles. It describes the proposed relational subgroup discovery algorithm and provides background about constraint exploitation in inductive databases and data mining. Section 3 then takes a closer look at how constraints are used in our approach to relational subgrup discovery. Section 4 describes the application of our approach to a relational learning benchmark concerning the mutagenecity of chemicals and to a real-life telecommunications dataset. Section 5 concludes by summarizing the results and presenting plans for further work.

2 Background

2.1 Relational Subgroup Discovery

The input to the RSD algorithm consists of **a relational database** containing one main table (relation), where each row corresponds to a unique *individual* and one attribute of the main table is specified as the *class* attribute - this table defines the *training examples*, and other tables (relations) defining the *background knowledge*. In addition, **a mode-language definition** is given, which is used to construct first-order features.

The output of RSD is a set of subgroups whose class distributions differ substantially from the class distribution in the complete data set. The subgroups are defined by conjunctions of (automatically generated/defined) first-order features. The RSD algorithm proceeds in two stages: first-order feature construction and rule-based subgroup discovery.

RSD First-order Feature Construction In our approach to first-order feature construction, based on [11, 16, 20], local variables referring to parts of individuals are introduced by so-called *structural predicates*. In a given 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 [17] (called *properties* in [11]) that 'consume' all new variables by assigning properties of individuals or their parts, represented by variables introduced so far. Utility predicates do not introduce new variables. (Examples of both types of predicates will be given below.)

The design of an algorithm for constructing first-order features can be split into two relatively independent problems:

Step 1: Identify features. This step results in identifying all first-order literal conjunctions that form a feature in the sense explained above, 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.

Step 2: Employ constants. This step results in extending the feature set by variable instantiations. Certain features are copied several times with some variables substituted to constants 'carefully' chosen from the input data. During this process, some irrelevant features are detected and removed, based on several constraints.

Both steps can be viewed as an exploitation of the combination of pre-set and user-defined sets of constraints of both syntactic (language-related) and semantic (data-oriented) character. From this viewpoint, they will be explained in detail in the devoted Section 3.

RSD Rule Induction Algorithm The core of RSD is a subgroup discovery algorithm which can accept data propositionalized by the feature constructor described above. The algorithm inherits some basic principles of the CN2 rule learner [7], which are adapted in several substantial ways to meet the needs of subgroup discovery. The principal improvements, making it appropriate for subgroup discovery, involve the implementation of the weighted covering algorithm, incorporation of example weights into the weighted relative accuracy heuristic, probabilistic classification, and the area under the ROC curve rule set evaluation.

The Weighted Covering Algorithm. In the classical covering algorithm of rule set induction, only the first few induced rules may be of interest as subgroup descriptors with sufficient coverage, since subsequently induced rules are induced from biased example subsets, i.e., subsets including only positive examples not covered by previously induced rules. This bias constrains the population for subgroup discovery in a way that is unnatural for the subgroup discovery process which is, in general, aimed at discovering interesting properties of subgroups of the entire population. In contrast, the subsequent rules induced by the weighted covering algorithm allow for discovering interesting subgroup properties of the entire population.

The weighted covering algorithm modifies the classical covering algorithm in such a way that covered positive examples are not deleted from the current training set. Instead, in each run of the covering loop, the algorithm stores with each example a count that shows how many times (with how many rules induced so far) the example has been covered so far. Weights derived from these example counts then appear in the computation of *WRAcc*. Initial weights of all positive examples e_j equal 1, $w(e_j, 0) = 1$. The initial example weight $w(e_j, 0) = 1$ means that the example hasn't been covered by any rule, meaning 'please cover this example, it hasn't been covered before', while lower weights mean 'don't try too hard on this example'.

Weights of positive examples covered by the constructed rule decrease according to the formula $w(e_j, i) = \frac{1}{i+1}$. In the first iteration all target class examples contribute the same weight $w(e_j, 0) = 1$, while in the following iterations the contributions of examples are inverse proportional to their coverage by previously selected rules. In this way the examples already covered by one or more constructed rules decrease their weights while rules covering many yet uncovered target class examples whose weights have not been decreased will have a greater chance to be covered in the following iterations.

Modified WRAcc Heuristic with Example Weights. The modification of CN2 reported in [28] affected only the heuristic function: weighted relative accuracy (WRAcc) was used as search heuristic, instead of the accuracy heuristic of the original CN2, while everything else stayed the same. In RSD and its propositional counterpart CN2-SD [19], the heuristic function was further modified to enable handling example weights, which provide the means to consider different parts of the instance space in each iteration of the weighted covering algorithm.

In the *WRAcc* computation all probabilities are computed by relative frequencies. Alternatively, the Laplace [6] and the m-estimate [5, 9] could be used to estimate the probabilities.

In order to include example weights into WRAcc, the following notation is used. Let $n(B)$ stand for the number of instances covered by a rule $H \leftarrow B$, $n(H)$ for the number of examples of class H, and $n(H.B)$ for the number of examples correctly classified by a rule (true positives). We use $p(H.B)$ etc. for the corresponding probabilities. We then have that rule accuracy can be expressed as $Acc(H \leftarrow B) = p(H|B) = \frac{p(H.B)}{p(B)} = \frac{n(H.B)}{n(B)}$.

The modified *WRAcc* measure is then defined as

$$WRAcc(H \leftarrow B) = \frac{n'(B)}{N'} \left(\frac{n'(H.B)}{n'(B)} - \frac{n'(H)}{N'} \right) \tag{1}$$

where N' is the sum of the weights of all examples, $n'(B)$ is the sum of the weights of all covered examples, and $n'(H.B)$ is the sum of the weights of all correctly covered examples.

To add a rule to the generated rule set, the rule with the maximum *WRAcc* measure is chosen out of those rules in the search space, which are not yet present in the rule set produced so far. All rules in the final rule set are thus distinct.

Probabilistic Classification and Area Under the ROC Curve Subgroup Evaluation. As opposed to model induction, subgroup discovery aims at finding patterns in the data, described in the form of individual rules. Rules of the following form are induced: $H \leftarrow B[TPr, FPr]$, where TPr is the *sensitivity* or *true positive rate* computed as $TPr = \frac{TP}{TP+FN}$, and FPr is the *false alarm* or *false positive rate* computed as $FPr = \frac{FP}{TN+FP}$, where TP, TN, FP and FN are true positives, true negatives, false positives and false negatives, respectively.

Each probabilistic rule can be viewed as an individual probabilistic classifier, and plotted as a point in the *ROC space* (ROC: Receiver Operating Characteristic) [23], which is used to show classifier performance in terms of false positive rate FPr (plotted on the X-axis) that should be as low as possible, and true positive rate TPr (plotted on the Y-axis) that should be as high as possible.

The area under the ROC curve (AUC) can be used as a quality measure for comparing different subgroups. To compare individual subgroups, each rule / subgroup is plotted in the ROC space with its true and false positive rates. To compare sets of rules, we calculate the convex hull of the corresponding set of points in ROC space, selecting the subgroups which perform optimally under a particular range of operating characteristics. The area under this ROC convex hull (AUC) indicates the combined quality of the optimal subgroups.

2.2 Constraints in inductive databases

Inductive databases [14] embody a database perspective on knowledge discovery, where knowledge discovery processes are considered as query processes. In addition to normal data, inductive databases contain patterns (either materialized or defined as views). Data mining operations looking for patterns are viewed as queries posed to the inductive database. In addition to patterns (which are of local nature), models (which are of global nature) can also be considered.

A general formulation of data mining [21] involves the specification of a language of patterns and a set of constraints that a pattern has to satisfy with respect to a given database. The constraints that a pattern has to satisfy can be divided in two parts: language constraints and evaluation constraints. The first only concern the pattern itself, the second concern the validity of the pattern with respect to a database.

Inductive queries consist of constraints and the primitives of an inductive query language include language constraints (e.g., find association rules with item A in the head) and evaluation primitives. Evaluation primitives are functions that express the validity of a pattern on a given dataset. We can use these to form evaluation constraints (e.g., find all item sets with support above a threshold) or optimization constraints (e.g., find the 10 association rules with highest confidence).

Constraints thus play a central role in data mining and constraint-based data mining is now a recognized research topic [4]. The use of constraints enables more efficient induction as well as focussing the search for patterns on patterns likely to be of interest to the end user. While many different types of patterns have been considered in data mining, constraints have been mostly considered in mining frequent itemsets and association rules, as well as some related tasks, such as mining frequent episodes, Datalog queries, molecular fragments, etc. Few approaches exist that use constraints for other types of patterns/models, such as size and accuracy constraints in decision trees [13] or in classification rule discovery as mentioned in the introduction.

In this paper, we consider the use of constraints in the context of relational subgroup discovery (RSD). Our approach to RSD first performs feature generation, then applies a propositional approach to subgroup discovery (the RSD implementation in the Yap Prolog with a user's manual and sample problems can be obtained from `http://labe.felk.cvut.cz/~zelezny/rsd`). Constraints are used in both phases, as discussed in detail in the following section.

3 Using constraints in RSD

The curse of combinatorial dimensionality is present in the principles underlying both procedural phases of RSD. Therefore RSD makes heavy use of both syntactic and semantic constraints exploited by search-space pruning mechanisms. On one hand, some of the constraints (such as *feature undecomposability*) are deliberately enforced by the system and pruning based on these constraints is guaranteed not to cause the omission of any solution. On the other hand, additional contraints (e.g. maximum *variable depth*) may be tuned by the user. These are designed with the intention to most naturally reflect possible user's heuristic expectations or minimum requirements on quantitative evaluations of search results.

3.1 Constraints in feature construction

Motivated by language-bias declarations used in ILP systems, RSD accepts language declarations very similar to those used by the systems Aleph [26] and Progol [22], including variable types, modes, setting a *recall* parameter etc, used to syntactically constrain the set of possible features. The use of the language bias declarations are best explained on a simple example. For this purpose we use the famous East-West trains domain.

Structural predicates. By the mode declaration `:-modeb(1, hasCar(+train, -car))` the user tells the system that the binary background relation `hasCar` may be employed in the body of constructed features, so as to provide the identification of some car of a specified train. The number 1 ("recall") determines that a feature can address at most one car of a given train. Input variables are labeled by the + sign, and output variables by the - sign.

Property predicates. Defined as above, but have no output variables.

Head predicate. Its declaration always contains exactly one variable of the input mode (e.g., `:-modeh(1, train(+train))`. The declaration serves merely to identify the key of the main individual.[1]

[1] The head declaration thus may seem overly complicated but contributes to compatability with declarations used with the widely used ILP systems mentioned earlier.

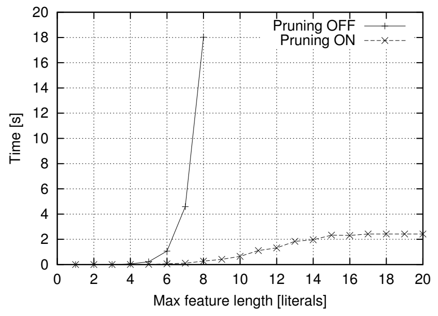


Fig. 1. The effect of pruning in the syntactic feature construction on efficiency in the East-West Trains domain. The diagram shows the amount of time needed to produce the exhaustive set of features for a given maximum feature length when pruning is off or on.

RSD produces all features satisfying the mode and setting declarations. The features produced by RSD have to satisfy an important constraint: a feature may not be decomposable into a conjunction of two features.

For example, the feature set based on the modes

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

will contain a feature

`f(A):- hasCar(A,B),hasCar(A,C),long(C),long(B),notSame(B,C).`

but it will not contain a feature with the body

`hasCar(A,B),hasCar(A,C),long(B),long(C)`

as such an expression would clearly be decomposable into two separate features. We do not construct such decomposable expressions, as these are redundant for the purpose of the subsequent search for rules with conjunctive antecedents.

The language constraint of undecomposability plays a major role: it enables pruning the search for possible features without losing any solutions. As an example, Figure 1 illustrates the speedup gained by the pruning on the East-West Trains domain (evaluation on real-life data will be shown in the experimental section).

In addition, other language constraints can be specified. These are: the maximum length of a feature (number of contained literals), maximum *variable depth* [22] and maximum number of occurrences of a given predicate symbol. If constraints are not specified by the user, the first two acquire a default value while the last is unlimited.

Unlike Aleph and Progol declarations, RSD does not use the # sign to denote a constant-value argument. In the mentioned systems, constants are provided by a single saturated example, while RSD extracts constants from all the input data (examples). The user can instead utilize the special reserved property predicate `instantiate/1`, which does not occur in the background knowledge, to specify a variable that should be substituted with a constant during feature construction. For example, from the modes

```
:-modeh(1, train(+train)).
:-modeb(1, hasCar(+train, -car)).
:-modeb(1, hasLoad(+car, -load)).
:-modeb(1, hasShape(+load, -shape)).
:-modeb(*, instantiate(+shape)).
```

exactly one feature is generated:

```
f1(A) :- hasCar(A,B), hasLoad(B,C), hasShape(C,D), instantiate(D).
```

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 with 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`:

```
f11(A) :- hasCar(A,B), hasLoad(B,C), hasShape(C,rectangle).
```

A similar principle applies for features with multiple occurences of the `instantiate/1` literal. Arguments of this literal within the 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.

However, not all such features will appear in the resulting set. For the sake of efficiency, we do not perform feature filtering by a separate postprocessing procedure, but rather discard certain features already during the feature construction process described above. The following constraints are used: (a) no feature should have the same value for all examples and (b) no two features should have the same values for all examples. For the latter case, only the syntactically shortest feature is chosen to represent the class of semantically equivalent features. In addition, a minimum number of examples for which a feature has to be true can be prescribed. This constraint is similar to the minimum support constraints in mining frequent item sets.

3.2 Constraints in subgroup discovery

In the subgroup discovery phase, a language constraint employed is the prescription of a maximal number of conditions/features in the description of a subgroup.

Several evaluation functions are considered. These include accuracy, weighted relative accuracy (WRAcc), significance, and area under the ROC curve. Accuracy and WRAcc are used in optimization constraints, i.e., RSD looks for rules with high accuracy/WRAcc. In fact, they are used as heuristic functions in RSD. Significance is used in evaluation constraints, i.e., one can prescribe a significance threshold that rules have to satisfy (expressed as significance at e.g., 99% level). WRAcc may be used in a similar fashion.

For lack of space we do not provide here a tabular summary of all employed constraints and the ways of their setting, this can be however found in the RSD user's manual available from the above mentioned RSD download page.

4 Experiments

We have experimented with the well-known relational learning benchmark concerning the Mutagenicity of chemicals and we have also applied RSD to the analysis of a real-life telecommunications dataset. The Mutagenesis data have been described in detail in many sources, see e.g. [27]. The Telecommunication application has been described by Železný et al. in [30, 29]; next we give a brief overview of the data.

4.1 Telecommunications

The data represent incoming calls (1995 items thereof) to an enterprize. Each such call is answered by a human operator and in the usual case further transferred to an attendant distinguished by his/her line number. Further re-transfers may also occur. Each sequence of such transfers is tracked by a computerized exchange and related data are stored in a logging file. By a suitable transformation thereof, one may obtain a relation `incoming/5`, represented by ground facts of the form `incoming(`*date, time, caller, operator, result*`)`. The argument *result* either takes a constant value or is a recursively defined function, so that *result* $\in$ {`talk, unavailable, transfer([`$ln_1, ln_2, ..., ln_n$`],` *result*`)`}, where $ln_1...ln_{n-1}$ denote line numbers to which unsuccessful attempts to transfer have been made, and ln_n the result of the last transfer attempt.

For example, the ground fact

```
incoming(date(10,18), time(13,37,29), [0,6,4,8,2,5,6,8,4,9], 32,
transfer([16,12],transfer([26],talk))).
```

describes a call from the number 0648256849 at 13:37:29 on 10/18 received by the operator on line 32. The operator first tried to transfer the caller to line 16 without success, and then transferred him/her successfully to line 12. The person on line 12 further redirected the caller to line 26. After a talk with line 26, the call was terminated.

We divide all instances of incoming transferred calls into classes determined by the line to which the operator tried to transfer the caller first. We thus obtain 25 classes. Attributes of examples (the main table records) then consist of the first four arguments of `incoming/5` and the class attribute. Finding subgroups interesting with respect to this class attribute may contribute to purposes of decision support of the operator. Further, if the subgroup set has sufficient predictive power, it may partially or completely substitute the operator.

Let us now comment on two of the available background relations. The predicate `prefix(Number,Prefix)` is true whenever the second (output) argument is the prefix (of any length) of the first (input) argument. For instance, regarding the example given above, `prefix([0,6,4,8,2,5,6,8,4,9],[0,6,4])` is true.

This background predicate proved useful in previously published results, since it is able to bind callers from the same area, city, company, office etc. The predicate gives multiple possible outputs for a given input. When used as part of a feature definition, it will be the job of the feature constructor to decide which prefixes should be used (possibly in conjunction with other literals) to

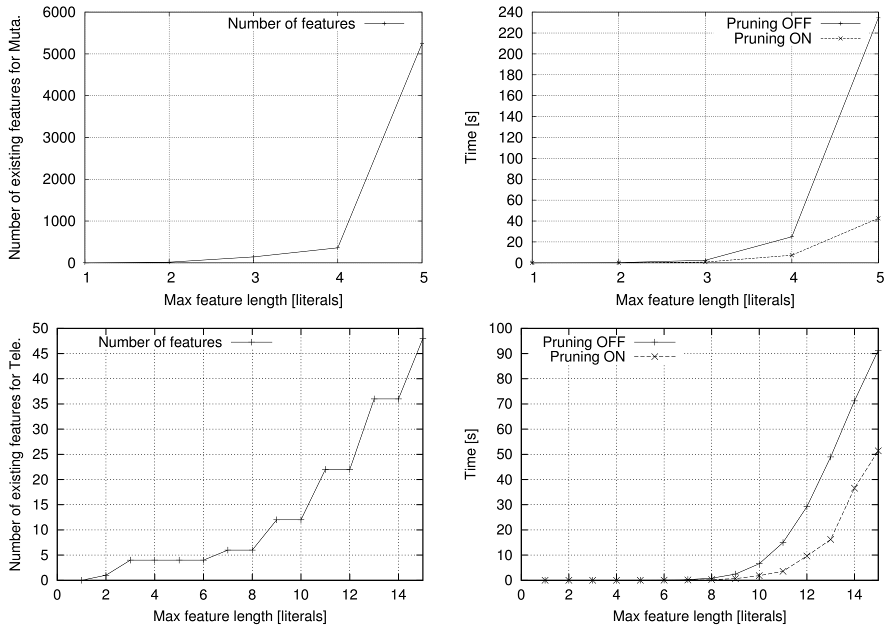


Fig. 2. The number of existing features (left) and the effect of undecomposability enabled pruning in the syntactic feature construction on efficiency (right) in the Mutagenesis (top) and Telecommunication (bottom) domain.

generate features with acceptable coverage measures. Out of the prefixes kept, the rule inducer chooses those that help identify interesting subgroups.

Another background predicate `prev_attempt/6` reflects the fact that a line desired by the caller may often be determined by looking at the caller's recent attempts to reach a person, i.e., by inspecting past records (w.r.t. the time-label of the current example) in the `incoming/4` relation. This problem setting is thus not far from what is known as *multi-instance learning* [8], where relevant attribute values describing an instance extend in multiple rows of a single table.

For example, the goal

```
prev_attempt(date(10,18),time(13,37,29),
[0,6,4,8,2,5,6,8,4,9], Line, When, Result).
```

will succeed with the result

```
Line=10, When=today, Result = unavailable,
```

provided the caller 0648256849 failed to reach line 10 on 10/18 before 13:37:29. Again, the `prev_attempt/6` may obviously yield multiple outputs for a given instantiations of the input arguments.

4.2 Effects of constraints on feature generation

The use of the undecomposability constraint and the pruning enabled thereby greatly reduces the time necessary to generate the features. This reduction increases with the maximum feature length, as illustrated in Figure 2.

The use of the other feature constraints, i.e., the minimum coverage, unique coverage and incomplete coverage (the latter two are referred to as filtering) reduces the number of features generated. In Mutagenesis, the maximum feature length was set to 5 and the minimum feature coverage to 20 instances, obtaining 42 different features. In the Telecom domain, we set the maximum feature length to 8. In this case, using a minimum coverage of 20 instances yields 138 features.

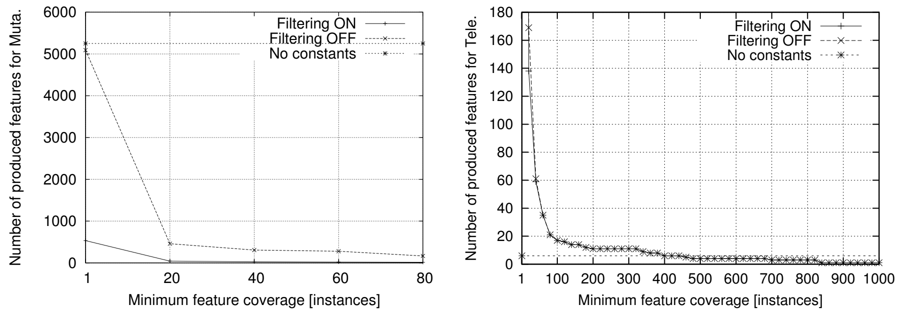


Fig. 3. The effect of using the constraint of unique and incomplete coverage ("Filtering ON" line) vs. ignoring this constraint ("Filtering OFF" line) and the user-adjustable minimum-coverage constrain (left-to-right decay) for Mutagenesis (left) and Telecommunication (right). The "No constants" curve (independent of the horizontal axis) corresponds to the number of features before instantiations to constants (before Step 2 of feature construction), this number is high in Mutagenesis due to many features inprovable with any instantiation to constants.

4.3 Results of subgroup discovery

An example feature in the Mutagenesis domain is `f12(A):-atm(A,B),atm_chr (B,C),lteq_c(C,0.142)` expressing that a drug contains an atom with charge less or equal to 0.142, or `f31(A):-benzene(A,B),benzene(A,C), connected( C,B)`, expressing the presence of two connected benzene rings in the chemical. In telecommunications, an example feature is `f99(A):-ext_number(A,B),prefix( B,[0,4,0,7])`, meaning that the caller's number starts with 0407. Another feature is `f115(A):- call_date(A,B),call _time(A,C),ext_number(A,D), prev _attempt(B,C,D,[3,1], today,unavailable)`, meaning that the caller (of the current call) has today tried to reach line 31, which was unavailable.

With these features, we use the RSD rule induction algorithm with altered covering strategy and heuristic function to produce sets of subgroup-describing rules. The characteristics of the discovered rules are shown in Table 1. We also present descriptions of some of the discovered subgroup in Telecommunication, with comments from the domain expert on the descriptions (Table 2 in Appendix) and distributional characteristics of the subgroups.

Table 1. Characteristics of subgroup-describing rules obtained by the RSD rule induction algorithm in the Mutagenesis and Telecom domains. Algo refers to the combination of search heuristic (A-accuracy, W-WRacc (weighted relative accuracy)) and covering algorithm (C-covering, W-WeCov (weighted covering with $\gamma = 1$)). S = significance, C = coverage, A = area under ROC curve. R : F = average number of rules/class : average number of features per rule. R′ : F′ = same as above, only rules on covex hull are considered. The rule generation for a given class is terminated if the search space has been completely explored or 10 subgroup rules have been generated for that class in Telecommunication (5 in Mutagenesis). Reported results are averages and standard deviations from a 10-fold stratified cross-validation procedure.

Mutagenesis

	Performance			**Complexity**	
Algo	S	C	A	R : F	R′ : F′
AC	1.99	11.33%	0.69	10.00 : 2.16	10.00 : 2.16
	(0.92)	(3.74)	(0.07)	(0.00 : 0.07)	(0.00 : 0.07)
AW	1.33	7.62%	0.58	10.00 : 2.50	10.00 : 2.50
	(1.05)	(4.88)	(0.06)	(0.00 : 0.11)	(0.00 : 0.11)
WC	4.22	35.81%	0.86	3.70 : 1.73	2.30 : 1.62
	(1.22)	(6.44)	(0.06)	(0.82 : 0.33)	(0.48 : 0.22)
WW	7.48	40.58%	0.90	10.00 : 2.63	6.50 : 2.43
	(1.28)	(4.74)	(0.04)	(0.00 : 0.07)	(0.97 : 0.11)

Telecommunication

	Performance			**Complexity**	
Algo	S	C	A	R : F	R′ : F′
AC	2.90	0.37%	0.55	7.36 : 2.39	6.88 : 2.47
	(0.38)	(0.05)	(0.02)	(0.12 : 0.04)	(0.19 : 0.04)
AW	2.25	0.25%	0.55	9.96 : 2.56	9.60 : 2.61
	(0.52)	(0.04)	(0.02)	(0.07 : 0.03)	(0.07 : 0.03)
WC	11.29	4.98%	0.67	6.12 : 2.17	5.20 : 2.28
	(1.71)	(0.54)	(0.02)	(0.16 : 0.04)	(0.16 : 0.04)
WW	11.99	4.02%	0.70	9.64 : 2.06	6.68 : 2.29
	(1.05)	(0.41)	(0.01)	(0.12 : 0.01)	(0.20 : 0.03)

The most significant observation about the results in Table 1 is that the *WRAcc* heuristic very significantly improves the performance with respect to the other accuracy heuristics, in terms od all three quality aspects.

Overall, the combination of *WRAcc* with the strategy of example weighting yields the best performance. This agrees with the findings in [18], where a more extensive empirical evaluation was conducted on a collection of (non-relational) subgroup-discovery problems, comparing the CN2 algorithm with CN2 incorporating the WRAcc heuristic, and further the CN2-SD system (which incorporates the *WRAcc* heuristic and the example weights). These three algorithms roughly correspond to the methods we denote above (in Table 1) as AC, WC, and WW, respectively. The combination of the accuracy heuristic with example weighting (AW) seems not to perform well in the domains considered.

5 Conclusions

This paper presents an approach to relational subgroup discovery, whose origins are based on the recent developments in subgroup discovery [32, 12] and propositionalization through first-order feature construction [11, 16, 20]. It presents the algorithm RSD which transforms a relational subgroup discovery problem to a propositional one, through efficiency-conscious first-order feature construction. Efficiency is boosted through the use of mode declarations and constraints used for pruning the search in the space of possible features.

Four variants of the RSD algorithm have been tested, by combining the standard accuracy (*Acc*) and the weighted accuracy (*WRAcc*) search heuristic used in the construction of individual rules, with the standard covering (*Cov*) and weighted covering (*WeCov*) algorithm used in the construction of a set of rules. The *WRAcc* heuristic combined with the *WeCov* algorithm is the preferred combination (due to an appropriate tradeoff between rule significance, coverage and complexity).

We have successfully applied the RSD algorithm in the Mutagenesis benchmark and the Telecom domain, a real-life dataset from a telecommunications company. These results have been evaluated as meaningful by the domain expert. Both the description of subgroups and their distributional characteristics make sense in many cases.

In further work, we are planning to use the RSD system in other subgroup discovery applications, including the analysis of traffic accidents, for which we have already performed the initial experiments using a relational database of 20 years (1979-1999) of accidents in the UK. We are also planning to investigate more closely an inductive database approach to relational subgroup discovery, along the directions of using constraints outlined in this paper. Finally, an idea suggested by a reviewer of this paper, of incrementally extending the feature set in dependence on the quality of the discovered subgroups, seems very much worth investigating.

Acknowledgments

We are grateful to Peter Flach for the collaboration in the development of the CN2-SD algorithm and in the genesis phase of the RSD algorithm. We are grateful also to Jiří Zídek, Atlantis Telecom s.r.o. for the comments on the Telecom subgroups discovered. F.Ž. is supported by the DARPA EELD grant F30602-01-2-0571 and the Czech ministry of education grant MSM 21230013. S.D. acknowledges the support of the cInQ (Consortium on discovering knowledge with Inductive Queries) project, funded by the European Commission.

References

1. Rakesh Agrawal, Hiekki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, pages 307–328, 1996.
2. John M. Aronis and Foster J. Provost. Efficiently constructing relational features from background knowledge for inductive machine learning. In *KDD Workshop*, 1994.
3. John M. Aronis, Foster J. Provost, and Bruce G. Buchanan. Eploiting background knowledge in automated discovery. In *KDD Workshop*, 1996.
4. R. Bayardo. Constraints in data mining. *SIGKDD Explorations*, 4(1), 2002.
5. B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 147–149. Pitman, 1990.
6. P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proc. Fifth European Working Session on Learning*, pages 151–163, Berlin, 1991. Springer.
7. Peter Clark and Tim Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
8. L. De Raedt. Attribute value learning versus inductive logic programming: The missing links (extended abstract). In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 1–8. Springer-Verlag, 1998.
9. S. Džeroski, B. Cestnik, and I. Petrovski. Using the m-estimate in rule induction. *Journal of Computing and Information Technology*, 1(1):37–46, 1993.
10. Tom Fawcett. Using rule sets to maximize ROC performance. In *ICDM*, pages 131–138, 2001.
11. P. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer-Verlag, 1999.
12. D. Gamberger and N. Lavrač. Expert guided subgroup discovery: Methodology and application. *JAIR*, 17:501–527, 2002.
13. M. Garofalakis and R. Rastogi. Scalable data mining with model constraints. *SIGKDD Explorations*, 2(2):39–48, 2000.
14. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
15. W. Kloesgen. Explora: A multipattern and multistrategy discovery assistant, 1996.
16. Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In Saso Džeroski and Nada Lavrač, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, September 2001.
17. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.
18. Nada Lavrač, Peter Flach, Branko Kavsek, and Ljupco Todorovski. Rule induction for subgroup discovery with cn2-sd. In Marko Bohanec, Dunja Mladenič, and Nada Lavrač, editors, *2nd International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, August 2002.

19. Nada Lavrač, Peter Flach, Brank Kavšek, and Ljupčo Todorovski. Adapting classification rule induction to subgroup discovery. In V. Kumar et al., editor, *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 266–273. IEEE Computer Society, December 2002.
20. Nada Lavrač and Peter A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, October 2001.
21. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
22. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
23. Foster J. Provost and Tom Fawcett. Robust classification systems for imprecise environments. In *AAAI/IAAI*, pages 706–713, 1998.
24. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
25. Luc De Raedt, Hendrik Blockeel, Luc Dehaspe, and Wim Van Laer. Three companions for data mining in first order logic. In Saso Džeroski and Nada Lavrač, editors, *Relational Data Mining*, pages 105–139. Springer-Verlag, 2001.
26. 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 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 89–104. Springer-Verlag, 1996.
27. A. Srinivasan, S. 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), 1996.
28. Ljupčo Todorovski, Peter Flach, and Nada Lavrač. Predictive performance of weighted relative accuracy. In Djamel A. Zighed, Jan Komorowski, and Jan Zytkow, editors, *4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000)*, pages 255–264. Springer-Verlag, September 2000.
29. F. Železný, P. Miksovsky, O. Stepankova, and J. Zidek. ILP for automated telephony. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 276–286, 2000.
30. F. Železný, J. Zidek, and O. Stepankova. A learning system for decision support in telecommunications. In *Proc. of the 1st Int. Conf. on Computing in an Imperfect World, Belfast 4/2002*. Springer-Verlag, 2002.
31. Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In Jan Komorowski and Jan Zytkow, editors, *Proc. First European Symposion on Principles of Data Mining and Knowledge Discovery (PKDD-97)*, pages 78–87, Berlin, 1997. Springer Verlag.
32. Stefan Wrobel. Inductive logic programming for knowledge discovery in databases. In Saso Džeroski and Nada Lavrač, editors, *Relational Data Mining*, pages 74–101. Springer-Verlag, September 2001.
33. Stefan Wrobel and Saso Džeroski. The ILP description learning problem: Towards a general model-level definition of data mining in ILP. In K. Morik and J. Herrmann, editors, *Proc. Fachgruppentreffen Maschinelles Lernen (FGML-95)*, 44221 Dortmund, 1995. Univ. Dortmund.

Appendix

Expert analysis of the induced rules shows that some of them identify novel and interesting information. Especially revealing are the comments related to the changes of class frequency associated with the rules.

In the overall distribution, calls to line 21 are most common. The expert comments that this reflects his expectations, as the person at line 21 is a marketer, and people interested in products call this line most frequently. In subgroup `Tele1`, there is (a) an increase in line 21 frequency: clients not receiving an ordered package often wait until Friday and then complain with line 21; and (b) a decrease in line 13 frequency: the person at line 13 mostly collaborates with dealers who have less business on Fridays. For subgroup `Tele4` there is a) an increase in line 28 frequency: repeated attempts to reach line 28, and (b) an increase in line 21 frequency: the person at line 28 works as technical support for products sold by person on line 21.

Table 2. Subgroup descriptions in the form $H \leftarrow B$ $[TP, FP]$, definitions of used features, and subgroup interpretation including expert's comments.

`Tele1`: `line21(A)` ← `f40(A)` [56,268] `f40(A):-call_date(A,B),dow(B,fri).` Calls received on Fridays. *Expert's evaluation*: Not a novel information.
`Tele2`: `line11(A)` ← `f132(A)` [32,0] `f132(A):-ext_number(A,B), prefix(B,[8,5,1,3,1,1,1,1]).` Calls received from number 85131111. *Expert's explanation:* The caller is the secretary's husband. She does not have a direct-access line, thus this call is transferred by an operator. *Expert's evaluation*: Novel information. *Remark.* Although the last literal formally identifies a *prefix* of the calling number, it is in fact the complete number of the caller.
`Tele3`: `line21(A)` ← `f54(A)` [81,254] `f54(A):-ext_number(A,B),prefix(B,[0,4]).` Calls received from a number that starts with 04. *Expert's explanation:* Prefix 04 is too general (code covers a large area) to find an explanation. *Expert's evaluation:* Novel information. Uncertain.

`Tele4`: `line28(A)` ← `f7(A)` [22,11]
`f7(A):-call_date(A,B),call_time(A,C),`
    `ext_number(A,D), prev_attempt(B,C,D,[2,8], last_hour, unavailable).`
Calls received from a caller who has in the last hour attempted to directly (not through an operator) reach line 28, which was unavailable.
Expert's explanation: It is plausible that people try line 28 as the second attempt when line 21 is unavailable. Subgroup probably mostly covers people with technical difficulties with a product sold by person on line 21.
Expert's evaluation: Novel information.