THE 18TH EUROPEAN CONFERENCE ON MACHINE LEARNING
AND
THE 11TH EUROPEAN CONFERENCE ON PRINCIPLES AND PRACTICE
OF KNOWLEDGE DISCOVERY IN DATABASES

PROCEEDINGS OF THE SIXTH INTERNATIONAL WORKSHOP ON MULTI-RELATIONAL DATA MINING

MRDM'07

September 17, 2007

Warsaw, Poland

Editors:
Donato Malerba
Department of Computer Science, University of Bari
Annalisa Appice
Department of Computer Science, University of Bari
Michelangelo Ceci
Department of Computer Science, University of Bari

Preface

The 6^{th} International Workshop on Multi-Relational Data Mining (MRDM 2007) was held in Warsaw, Poland, on September 17^{th} 2007 in conjunction with ECML/PKDD 2007: the 18^{th} European Conference on Machine Learning (ECML) and the 11^{th} European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD).

Data mining algorithms look for patterns in data. While most existing data mining approaches look for patterns in a single data table, multi-relational data mining (MRDM) approaches look for patterns that involve multiple tables (relations) from a relational database. 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. Following the mainstream of MRDM research, the most common types of patterns and approaches considered in data mining have been extended to the multi-relational case and MRDM now encompasses relational association rule discovery, relational classification rules, relational decision and regression trees, and probabilistic relational models, among others. At same time, MRDM methods have been successfully applied across many application areas, ranging from the analysis of business data, through bioinformatics and pharmacology to Web mining and Spatial Data mining. Our goal is to bring together researchers and practitioners of data mining interested in methods for finding patterns in expressive languages from multi-relational / structured data and their applications. The workshop is the sixth of its kind. It follows the success of the workshops on Multi-Relational Data Mining, held both in Europe (ECML/PKDD 2001) and in USA (KDD 2002, 2003, 2004 and 2005).

Sixteen contributions were originally submitted, twelve of which were accepted for presentation. Each submission was evaluated by three independent referees. Besides paper presentations, the scientific programme also featured an invited talk by Luc De Raedt (Department of Computer Science, Katholieke Universiteit Leuven, Belgium).

We would like to thank the invited speaker, all the authors who submitted papers and all the workshop participants. We are also grateful to members of the program committee members and external referees for their thorough work in reviewing submitted contributions with expertise and patience. A special thank is due to both the ECML/PKDD Workshop Chair and the members of ECML/PKDD Organizing Committee who made this event possible.

Warsaw, September 2007

Donato Malerba
Annalisa Appice
Michelangelo Ceci

Workshop Organization

Workshop Chairs

Donato Malerba	University of Bari - Italy
Annalisa Appice	University of Bari - Italy
Michelangelo Ceci	University of Bari - Italy

Program Committee

Hendrik Blockeel	Katholieke Universiteit Leuven - Belgium
Jean-François Boulicaut	INSA Lyon - France
Sašo Džeroski	Jožef Stefan Institute - Slovenia
Peter Flach	University of Bristol - UK
Thomas Gärtner	Fraunhofer Institute for Autonomous Intelligent Systems - Germany
Lise Getoor	University of Maryland - USA
David Jensen	University of Massachusetts - USA
Kristian Kersting	MIT Computer Science and Artificial Intelligence Laboratory - USA
Joerg-Uwe Kietz	Kdlabs AG, Zurich - Switzerland
Arno Knobbe	Universiteit Utrecht - The Netherlands
Joost Kok	Leiden University - The Netherlands
Stefan Kramer	Technical University Munich - Germany
Nada Lavrač	Jožef Stefan Institute - Slovenia
Celine Rouveirol	University Paris Sud XI - France
Michele Sebag	University Paris Sud XI - France
Arno Siebes	Universiteit Utrecht - The Netherlands
Stefan Wrobel	Fraunhofer Institute for Autonomous Intelligent Systems / University of Bonn - Germany

Additional Reviewers

Marenglen Biba	University of Bari - Italy
Anton Dries	Katholieke Universiteit Leuven - Belgium
Aneta Ivanovska	Jožef Stefan Institute - Slovenia
Arne Koopman	Universiteit Utrecht - The Netherlands
Christine Körner	Fraunhofer Institute for Autonomous Intelligent Systems - Germany
Wannes Meert	Katholieke Universiteit Leuven - Belgium
Jan Struyf	Katholieke Universiteit Leuven - Belgium
Bernard Zenko	Jožef Stefan Institute - Slovenia

Table of Contents

ProbLog and its Application to Link Mining in Biological Networks

(Invited Talk)

Luc De Raedt

Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200 A, B-3001 Heverlee, Belgium
luc.deraedt@cs.kuleuven.be

Abstract. ProbLog is a recently introduced probabilistic extension of Prolog [De Raedt, Kimmig, Toivonen, IJCAI 07]. A ProbLog program defines a distribution over logic programs by specifying for each clause the probability that it belongs to a randomly sampled program, and these probabilities are mutually independent. The semantics of ProbLog is then defined by the success probability of a query in a randomly sampled program. It has been applied to link mining and discovery in a large biological network. In the talk, I will also discuss various learning settings for ProbLog and link mining, in particular, I shall present techniques for probabilistic local pattern mining, probabilistic explanation based learning and theory compression from examples [De Raedt et al, ILP 96].

A Multi-Relational Approach to Clustering Trajectory Data

Gianni Costa, Alfredo Cuzzocrea, Giuseppe Manco,
Riccardo Ortale, and Howard Scordio

ICAR Inst., National Research Council, Italy
{costa,cuzzocrea,manco,ortale,scordio}@icar.cnr.it

Abstract. We propose a novel methodology for clustering multi-relational trajectory data. Our methodology consists of two steps. Initially, tuple linkages, defined in the database schema of the multi-relational trajectories, are leveraged to *virtually* organize the available route data into as many transactions, i.e. as sets of feature-value pairs. The identified transactions are then partitioned into homogeneous groups. Each discovered cluster is equipped with a *representative*, that provides an explanation of the corresponding group of trajectories, in terms of those feature-value pairs that are most likely to appear in a transaction belonging to that particular group. Outliers trajectories are placed into a *trash cluster*, that is finally partitioned to mitigate the dissimilarity between the trash cluster and the previously generated clusters.

1 Introduction

The wide exploitation of new techniques and systems for monitoring, collecting and storing location-aware data, provided by a wealth of technological infrastructures, such as GPS positioning [5], sensor and mobile-device networks, has made available various trajectory databases. This is at the basis of an increasing demand for intelligent tools, targeted at effectively and efficiently discovering actionable knowledge from such repositories, that are expected to foster innovative, customized applications and services, i.e. serendipitous meetings, colocation based interactions, intelligent interruptions [1], traffic engineering [2], business data analysis, location model learning [9] and so forth.

In the present paper, we deal with trajectory clustering, which is of great practical relevance in several applicative domains, ranging from the analysis of storm behaviour, the enhancement of physical route planning, to the evaluation of drug treatment effects on gene expression [7] and prediction systems [6, 8].

Traditional approaches to trajectory clustering generally assume a simple spatio-temporal representation, and exploit a suitable *distance measure* for grouping similar trajectories. This requires to cope with challenging issues such as differences in trajectory length or sampling rates, the incapability at taking into account further details of route context (that may influence similarity computation), and lack of efficiency (primarily due to costly alignments).

As a preliminary research effort, we here propose an alternative approach based on an underlying multi-relational representation, capable of modeling all

required aspects of trajectories. Precisely, the generic trajectory is described by fragments of categorical data, suitably organized within multiple database tables, each of which catches relevant features of the data at hand. Features correspond to trajectory properties and divide into *explicit* and *implicit*. Explicit features include traditional spatio-temporal data, i.e. coordinates of moving entities w.r.t. some reference system and position sampling times. Implicit features, instead, involve further properties of trajectories, such as spatial aspects, motion dynamics and background knowledge (i.e., morphological and topological descriptions of the traversed regions). Enabling a richer trajectory representation, such a multi-relational representation, allows more accurate evaluation of similarity from multiple perspectives and at different levels of granularity.

Two problematic alternatives exist in the current literature for performing multi-relational clustering. On one hand, the application of a single-table clustering scheme to the outcome of integrating the original tuples within multiple database tables, by means of joins or aggregations. This is a very expensive strategy, since the generic tuple can be joined with a huge number of referenced tuples. Moreover, joining typically yields a huge space of tuple features, among which the ones actually relevant for the specific clustering target must be identified. The other option consists in the design of suitable algorithms that directly manipulate data within different tables. In such case, feature selection is even more challenging, since feature relevance must be dynamically evaluated, in order to ensure effective and efficient clustering.

We propose a *hybrid* approach, that consists of two main phases. In the first step, tuple linkages defined in the original database schema are leveraged to *virtually* organize the available multi-relational trajectories into as many transactions, i.e. as sets of feature-value pairs. Such a preprocessing is fundamental to avoid the prohibitively high time and space cost of physical table joins. In the second step, the previously identified transactions are partitioned into homogeneous groups. The divisive scheme [3] considers all underlying features for proximity evaluation, and simply relates similarity between any two trajectories to their degree of overlap, i.e. to the commonality of values for corresponding categorical features. This allows us to overcome the amount of computation and possible inaccuracies typical of similarity measures, and also avoids the prior identification of relevant features. Each discovered cluster is equipped with a *representative*, which provides an explanation of the corresponding group of trajectories, in terms of those feature-value pairs that are most likely to appear in a transaction belonging to that particular group. Outlier trajectories are placed into a *trash cluster*, that is finally partitioned to mitigate the effects of the dissimilarity between the trash cluster and the other clusters previously generated.

The plan of the paper is as follows. Section 2 delineates the formal framework of our approach. Section 3 discusses the method employed for representing multi-relational trajectory data as transactions. Details about the clustering scheme adopted for partitioning such transactions are provided within Section 4. Section 5 draws conclusions and mentions promising directions of future research.

2 Formal Framework

We here fix some notations used throughout the paper. We are given a collection of database tables $R_M, R_1, R_2, \ldots, R_m$, such that (i) R_M denotes the main table, i.e. the one including the main information concerning the trajectories at hand, and (ii) $R_1, \ldots, R_m$ are supplementary tables including further objective route details as well as domain-specific background knowledge. The information stored within the generic table R_t conforms to the following scheme: $A_1^t, A_2^t, \ldots, A_{n_t}^t$, where n_t indicates the number of features (i.e. attributes) within R_t. By virtually joining the above tables, it is possible to augment table tuples with suitable annotations, that define an alternative view of the original data as $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i = \{A_1^h = v_1^h, \ldots, A_{n_h}^h = v_{n_h}^h | h = M, 1, \ldots, m\}$ is the conceptual representation of a multi-relational trajectory in transactional form, and v_l^h indicates the value assumed by the attribute A_l^h in the context of transaction x_i. Here, we assume that v_l^h is in a range of nominal values: numeric values can be managed by resorting to appropriate discretization techniques, which can be included either as a preprocessing or by a tighter integration within the proposed approach.

Henceforth, for ease of notation, we refer to transactions in D rather than to relational data within $R_M, R_1, R_2, \ldots, R_m$. However, we underline that data actually exists only as tuples within $R_M, R_1, R_2, \ldots, R_m$ and that the transactional view of a trajectory denotes the focussed selection of linked tuples from such tables, i.e. the ones annotated with the identifier of the specific trajectory.

Various schemes for clustering transactional data can now be applied to the partition D. The most well-known partitional approaches to clustering data are the *centroid-based methods*, such as the *K-Means* algorithm. In such approaches, each object $\mathbf{x}_i$ is assigned to a cluster j according to its distance $d(\mathbf{x}_i, \mathbf{m}_j)$ from a value $\mathbf{m}_j$ representing the cluster itself. $\mathbf{m}_j$ is called the *centroid* (or *representative*) of the cluster. In this paper, we adopt the *Transactional K-Means* scheme [3] to find a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$, of D, such that:

1. each C_i groups similar trajectories and is associated to a centroid $\mathbf{m}_i$;
2. $\mathbf{x}_i \in C_j$ if $d(\mathbf{x}_i, \mathbf{m}_j) \leq d(\mathbf{x}_i, \mathbf{m}_l)$ for $1 \leq l \leq k$, $j \neq l$;
3. the partition $\mathcal{C}$ minimizes $\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} d^2(\mathbf{x}_j, \mathbf{m}_i)$.

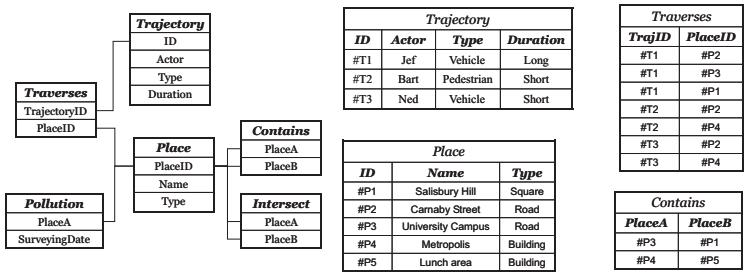
□

The preprocessing and clustering steps are discussed in the next sections.

3 From Multi-Relational Data to Virtual Transactions

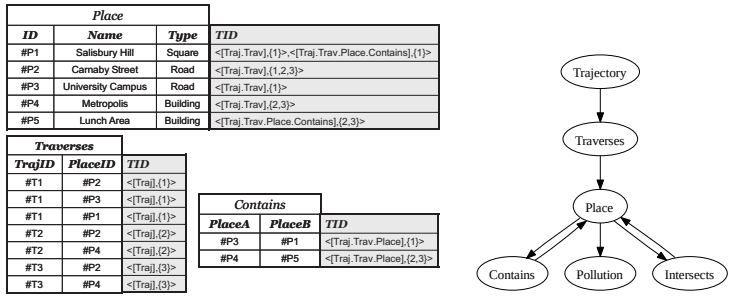
As already anticipated, the time and space cost for joining tables $R_M, R_1, R_2, \ldots, R_m$ is prohibitively high. To avoid physically joining the original tables, we employ the *tuple ID propagation* technique [10], a mechanism that virtually joins tables $R_M, R_1, R_2, \ldots, R_m$, by propagating the IDs (i.e., primary keys) of the tuples within the main table, R_M, to the tuples stored in the other tables, $R_1, R_2, \ldots, R_m$. The referential integrity constraint of available tuples is exploited, so that propagation flows along the links among tuples, represented by the references between foreign and primary keys.

In details, the propagation mechanism consists in the identification of a fixed join order for the tables, and in the annotation of each individual tuple within the supplementary tables taken one by one according to that order. Consider, e.g., the schema in fig. 1(a).



(a) Schema (b) Instances

Fig. 1. Relational trajectory database



(a) Tuple ID Propagation (b) FKL graph

Fig. 2. (a)Trajectories marked with virtual transaction IDs. (b) FKL-graph

This schema models several features of trajectories ranging from duration to traversed places to geometrical relationships among places (e.g., containment, intersection etc). Since we are interested in clustering trajectories, we consider *Trajectory* as the main table of our analysis, and we fix the join order as in the following foreign-key-linkage (FKL)-graph 2(b)

The tuple ID propagation then starts from the entry point in the FKL-graph (in our case, the `Trajectory` table), and moves towards the other tables accordingly. For each tuple t within any supplementary table R_i $(i = 1, \ldots, m)$, annotation augments t by a list of identifiers, which refer tuples of R_M joinable

with t (i.e., having foreign keys that point to t). The propagation process is incremental: the annotations of a tuple t_i within table $R_i(\neq R_M)$ are propagated to tuple t_j within $R_j(\neq R_M)$, if t_i has a foreign key that references t_j.

In the above example, edges in the FKL-graph drive the propagation process. As a result, the table instances in fig. 1(b) are annotated as shown in fig. 2(a). As an example, the first tuple of *Traverses* is annotated with $\langle[\texttt{Traj}], \{1\}\rangle$, as it has a join with the tuple of *Trajectory* having ID equal to 1. Similarly, the second tuple of the *Place* table is $\langle[\texttt{Traj.Trav}], \{1, 2, 3\}\rangle$ as the second tuple of `Place` has joins with the tuples of *Trajectory* having ID equal to 1, 2, and 3 throughout the first, fourth, and sixth tuple of *Traverses*, respectively.

Notice that, for each propagated ID, information concerning the join path is annotated as well. This allows us to effectively manage multiple references to a same tuple due to cycles within the FKL-graph. In fig. 2(a), the first tuple of the `Place` table is annotated twice by ID 1, as there are two join paths (`Traj.Trav` and `Traj.Trav.Place.Contains` respectively) which link the two tuples.

At the end of identifier propagation, semantic connections between main and supplementary tables are established, so that the transactional representation $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of the original trajectory data can be easily constructed by a simple linear scan of each relation (i.e. without physically joining tables $R_M, R_1, R_2, \ldots, R_m$). Each item of a transaction in D is modeled as *JoinPath.AttributeName.Value*, such that (*i*) *JoinPath* is the join path originating the actual item, (*ii*) *AttributeName* is the name of the attribute considered, and (*iii*) *Value* is the value of such attribute. In particular, i-th transaction collects all the attribute values of tuples annotated by value i. Within the transactional representation, both primary and external key attributes are ignored, as they do not provide significant information content. Figure 3 shows the transactional representation of trajectory data of figure 1(b).

$\mathbf{x}_1$ = {*Traj.Actor.Jef*, *Traj.Type.Vehicle*, *Traj.Duration.Long*, *Traj.Trav.Place.Name.SalisburyHill*, *Traj.Trav.Place.Type.Square*, *Traj.Trav.Place.Name.CarnabyStreet*, *Traj.Trav.Place.Type.Road*, *Traj.Trav.Place.Name.UniversityCampus*, *Traj.Trav.Place.Contains.Place.Name.SalisburyHill*, *Traj.Trav.Place.Contains.Place.Type.Square*}

$\mathbf{x}_2$ = {*Traj.Actor.Bart*, *Traj.Type.Pedestrian*, *Traj.Duration.Short*, *Traj.Trav.Place.Name.CarnabyStreet*, *Traj.Trav.Place.Type.Road*, *Traj.Trav.Place.Name.Metropolis*, *Traj.Trav.Place.Type.Building*, *Traj.Trav.Place.Contains.Place.Name.LaunchArea*, *Traj.Trav.Place.Contains.Place.Type.Building*}

$\mathbf{x}_3$ = {*Traj.Actor.Ned*, *Traj.Type.Vehicle*, *Traj.Duration.Short*, *Traj.Trav.Place.Name.CarnabyStreet*, *Traj.Trav.Place.Type.Road*, *Traj.Trav.Place.Name.Metropolis*, *Traj.Trav.Place.Type.Building*, *Traj.Trav.Place.Contains.Place.Name.LaunchArea*, *Traj.Trav.Place.Contains.Place.Type.Building*}

Fig. 3. Transactions obtained from virtual joining

4 Clustering Process

We adopt the transactional K-Means algorithm [3] to $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ into K homogeneous groups. The scheme is an effective and efficient enhancement of

the basic K-Means algorithm, capable to process volumes of (high-dimensional) categorical data. In the following, we discuss how to derive transactional K-Means from traditional K-Means.

The K-Means algorithm works as follows. First of all, K objects are randomly selected from D. Such objects correspond to some initial cluster centroids, and each remaining object in D is assigned to the cluster with the nearest centroid. Next, the algorithm iteratively recomputes the centroid of each cluster and re-assigns each object to the cluster of the nearest centroid. The algorithm terminates when the centroids do not change anymore. In that case, in fact, $\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} d^2(\mathbf{x}_j, rep(C_i))$ is minimized.

The general schema of K-Means is parametric w.r.t. both d and rep, that respectively denote the notions of distance measure and cluster centroid. Such concepts in turn are parametric to the domain of D. By suitably defining d and rep, so that they fit to the transactional domain, we can obtain an effective clustering scheme. In particular, we choose to compare transactions by means of the *Jaccard distance* [4] (denoted by d_J in the following), that measures the degree of overlap (i.e., the number of common feature-value pairs) between two transactions. Within a high-dimensional feature space, this has a main advantage of automatically preventing to consider irrelevant features for the comparison.

4.1 Cluster representative

Transactional data requires the computation of a cluster representative. In general, given a transaction domain $\mathcal{U}$ equipped with a distance function $d : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$ and a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\} \subseteq \mathcal{U}$, the centroid of $\mathcal{S}$, $rep(\mathcal{S})$, is the element that minimizes the sum of the squared distances $rep(\mathcal{S}) = min_{\mathbf{v} \in \mathcal{U}} \sum_{i=1}^{m} d^2(\mathbf{x}_i, \mathbf{v})$. In general, computing a cluster representative is a computationally expensive process. Moreover, optimal cluster centroids are not necessarily unique. To overcome such issues, we here exploit a general property of representatives of transaction clusters, grouped based on d_J, according to which frequent feature-value pairs across transactions in a cluster are very likely to belong to the representative of that cluster [3]. This enables the exploitation of the greedy heuristic rep_H, sketched in fig. 4, which initially computes an approximation of the representative of a given cluster as the intersection of all transactions in that cluster. The approximation is iteratively refined by adding the most frequent feature-value pairs in the cluster, until the sum of the distances can be minimized.

Computing rep_H can still be expensive, since feature-value pairs have to be sorted on the basis of their frequencies. A more efficient approximation, rep_γ, can be defined by introducing a user-defined threshold value γ, representing the minimum occurrence frequency that a feature-value pair must have to be inserted into the approximation of the cluster representative.

Definition 1. *Given a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ of transactions and a threshold $\gamma \in [0, 1]$, an approximate representative of $\mathcal{S}$, $rep_\gamma(\mathcal{S})$, can be defined as $rep_\gamma(\mathcal{S}) = \{\mathbf{v} \in \bigcup_i \mathbf{x}_i \mid freq(\mathbf{v}, \mathcal{S})/m \geq \gamma\}$ where $freq(\mathbf{v}, \mathcal{S}) = |\{\mathbf{x}_i | \mathbf{v} \in \mathcal{S}\}|$.* □

The approaches $rep_H(\mathcal{S})$ and $rep_\gamma(\mathcal{S})$ represent two viable alternatives. rep_γ represents an approximation that is extremely efficient to compute. However, it

Algorithm $rep_H(\mathcal{S})$

Input : A set of transactions $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.
Output : A transaction $\mathbf{m}$ that minimizes $f(\mathbf{m}) = \sum_{\mathbf{x} \in \mathcal{S}} {d_J}^2(\mathbf{x}, \mathbf{m})$.
Method :

- sort $\bigcup \mathcal{S}$ by increasing frequency, obtaining the list $a_1, \ldots, a_m$ such that $freq(a_i, \mathcal{S}) > freq(a_{i+1}, \mathcal{S})$;
- let initially $\mathbf{m} = \bigcap_{\mathbf{x} \in \mathcal{S}} \mathbf{x}$;
- while $f(\mathbf{m})$ decreases
  - add a_h to $\mathbf{m}$, for increasing values of h.

Fig. 4. The greedy technique for computing a representative trajectory

is influenced by the value of γ. Greater γ values correspond to a stronger intra-cluster similarity, less populated clusters and low-cardinality representatives. By the converse, lower γ values correspond to a weaker intra-cluster similarity, huge clusters and high-cardinality representatives. On the other side, rep_H yields a less efficient but parameter-free clustering scheme.

4.2 The Transactional *K*-Means Algorithm

The main scheme of the transactional K-Means algorithm is shown in Figure 5. The algorithm divides into two main phases. In the first phase, it computes $k+1$ clusters. Trajectories are assigned to the first k clusters, according to the distance measure d_J. The $(k+1)$-th cluster is referred to as trash, since it includes outlier trajectories, i.e. those transactions of D that have empty intersection with the representatives of first k clusters. According to the adopted notion of Jaccard distance, such trajectories are equally distant from clusters $C_1, \ldots, C_k$, which means that their assignment to any such a cluster is not significant. Therefore, outliers are placed within the trash.

The second phase is targeted at appropriately mitigating the effects of the high dissimilarity between the trash cluster and the other clusters previously generated. The basic idea consists in trying to reiterate the partitional scheme of the first phase on the trash cluster, in order to split the latter into l further clusters. Of course, the resulting final partition may include clusters with a single trajectory, since trajectories with substantially different transactional form can remain within the trash, until they are chosen as cluster centroids.

5 Conclusions and Future Work

We discussed a hybrid methodology to clustering multi-relational trajectory data. The approach initially exploits foreign-key linkages, defined in the schema of the available data, to alternatively view original trajectories as transactions. The latter are then partitioned into homogeneous groups by an effective clustering scheme, which also provides a proper management of outlier trajectories.

There are several lines of future research. First, an intensive empirical evaluation of the proposed technique has to be accomplished in order to investigate its effectiveness on real-world data.

Algorithm TrK-Means(D,k,γ)

Input : A dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of transactional trajectories, the desired number k of clusters. A cluster representative threshold value γ.
Output : A partition $\mathcal{C} = \{C_1, \ldots, C_{k+l}\}$ of D in $k + l$ clusters, where $l \geq 0$.
Method :

- randomly choose $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ and set $\mathbf{m}_j = \mathbf{x}_{i_j}$ for $1 \leq j \leq k$;
- repeat
  - for each j, set $C_j = \{\mathbf{x}_i | d_J(\mathbf{x}_i, \mathbf{m}_j) < d_J(\mathbf{x}_i, \mathbf{m}_l), 1 \leq l \leq k\}$;
  - set $C_{k+1} = \{\mathbf{x}_i |$ for each j $d_J(\mathbf{x}_i, \mathbf{m}_j) = 1\}$;
  - set $\mathbf{m}_j = rep(C_j)$ for $1 \leq j \leq k$;

  until m_j do not change;
- recursively apply the algorithm to C_{k+1}, producing a partition of C_{k+1} in l clusters.

Fig. 5. Overall scheme of the transactional K-Means algorithm

Also, though crucial to ensure effective data clustering, the notion of cluster representative is a transaction, which does not correspond to a true trajectory. Hence, we need to develop a method for reconstructing trajectories from corresponding representatives.

Finally, we plan to develop a suitable strategy for the dynamic selection of trajectory features, that avoids to consider all basic table attributes in the computation of trajectory proximity and still ensures clustering effectiveness.

References

1. D. Ashbrook and T. Starner. Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. *Personal Ubiquitous Computing*, 7(5):275–286, 2003.
2. S. Dzeroski. Relational Data Mining. *Springer-Verlag*, 2001.
3. C. Gozzi, F. Giannotti and G. Manco. Clustering Transactional Data. In *Procs. 6th PKDD'02 Conf.*, 175–187, 2000.
4. J. Han and M. Kamber. *Data Mining Techniques*. Morgan Kaufman, 2001.
5. A. Harrington and V. Cahill. Route Profiling: Putting Context to Work.. *ACM Symposium on Applied Computing*, 1567–1573, 2004.
6. K. Laasonen. Clustering and Prediction of Mobile User Routes from Cellular Data. In *Proc. of PKDD*, 569–576, 2005.
7. Y. Liang, B. Tayo, X. Cai, and A. Kelemen Differential and Trajectory Methods for Time Course Gene Expression Data. *Bioinformatics*, 21(13):3009–3016, 2005.
8. C. McCue. *Data Mining and Predictive Analytics in Public Safety and Security*. IT Professional, 8(4):12–18, 2006.
9. T. Takada, S. Kurihara, T. Hirotsu and T. Sugawara. Proximity Mining: Finding Proximity using Sensor Data History. *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, 129–138, 2003.
10. X. Yin, J. Han, J. Yang and P.S. Yu. Efficient Classification across Multiple Database Relations: A CrossMine Approach. In IEEE Transaction on Knowledge and Data Engineering, 18(6): 770–783, 2006.

Choosing the Right Patterns

An Experimental Comparison between Different Tree Inclusion Relations

Jeroen De Knijf * and Ad Feelders

Algorithmic Data Analysis Group
Department of Information and Computing Sciences, Universiteit Utrecht
PO Box 80.089, 3508 TB Utrecht

Abstract. In recent years a variety of mining algorithms has been developed, to derive all frequent subtrees from a database of labeled ordered rooted trees. These algorithms share properties such as enumeration strategies and pruning techniques. They differ however in the tree inclusion relation used and how attribute values are dealt with. In this work we investigate the different approaches with respect to 'usefulness' of the derived patterns, in particular, the performance of classifiers that use the derived patterns as features. In order to find a good trade-off between expressiveness and runtime performance of the different approaches, we also take the complexity of the different classifiers into account, as well as the run time and memory usage of the different approaches. The experiments are performed on two real datasets. The results show that significant improvement in both predictive performance and computational efficiency can be gained by choosing the right tree mining approach.

1 Introduction

Frequent tree mining has become an important and popular problem in the field of knowledge discovery and data mining. The main reasons for the increase in interest are the growing amount of semi-structured data (e.g. XML databases) and the urge to analyze and mine these databases. Furthermore, the availability of tree mining algorithms to exploit these databases, without losing information on the structure of the data, has increased the interest of the research community. Frequent tree mining can be seen as an extension of the apriori algorithm [1], to handle tree structured data in the mining process. Briefly, given a set of tree data, the problem is to find all subtrees that satisfy the minimum support constraint, that is, all subtrees that occur in at least $n\%$ of the data records.

Due to the popularity of tree mining, different approaches to mine labeled ordered trees have been proposed, see for example [2, 9, 12, 13]. Although, these methods share enumeration strategy and pruning techniques, the fundamental differences lie in the different notions of when a tree matches another tree and how attributes are handled. Also, when the notion of rooted ordered trees is broadened to unordered rooted trees, the

* Supported by the Netherlands Organisation for Scientific Research (NWO) under grant no. 612.066.304.

existing mining algorithms [3, 10, 14] use different tree inclusion relations. The fundamental question arises which of the mining algorithms should be used, when analyzing tree structured data. Comparisons between these methods are mainly based on time and memory performance. But besides run time and memory usage, the effect of the different tree inclusion relations and different approaches to handling attributes are of great importance to the data analysis task. To compare the different approaches, we consider the task of building a classifier from the frequent patterns produced by the tree miner. We consider several aspects in the comparison: accuracy and complexity of the classier produced, as well as runtime and memory usage of the tree miner.

In the next section we describe the basic notations and formally define the different tree inclusion relations and the different ways to handle attribute value-pairs in the mining process. In the following section the research questions are formulated. In section 4 we describe how a classification model is constructed from frequent patterns. In section 5 the experiments to answer the questions are carried out and evaluated. In the final section, we answer the stated questions and discuss the conclusions drawn.

2 Preliminaries

In this section we provide the basic concepts and notation used in this paper. A labeled rooted ordered tree $T = \{V, E, \leq, L, v_0, M\}$ is an acyclic directed connected graph which contains a set of nodes V, and an edge set E. The labeling function L is defined as $L : V \rightarrow \Sigma$, i.e., L assigns labels from alphabet Σ to nodes in V. The special node v_0 is called the root of the tree. If $(u, v) \in E$ then u is the parent of v and v is a child of u. For a node v, any node u on the path from the root node to v is called an ancestor of v. If u is an ancestor of v then v is called a descendant of u. Furthermore there is a binary relation '$\leq$' $\subset V^2$ that represents an ordering among siblings.

In some tree structured data, such as XML, attributes are used to describe properties of nodes. To model this, we assume a set of attribute-value pairs, denoted by : $\mathcal{A} = \{(A_1 : a_1), \ldots, (A_n : a_n)\}$, where each attribute-value has a finite domain. We further assume that there is an ordering specified on the attribute-value pairs, which can be arbitrary. To each node v in V, a subset of $\mathcal{A}$ is assigned; we call this set the attributes of v. More formally, we define a mapping $M : V \rightarrow \mathcal{P}(\mathcal{A})$. The size of a tree is defined as the number of nodes it contains; we refer to a tree of size k as a k-tree.

The notion of tree inclusion is dependent on how to handle attributes, up till now three different approaches are used:

- No attributes are associated with the data, or the attributes are considered irrelevant and are ignored in the mining process [2].
- No distinction is made between attributes and nodes; attribute-value pairs are added as child nodes of the corresponding original node [13].
- Attributes are modeled as properties of nodes. If a node has attributes associated to it in the database, then this node can only occur in a frequent pattern if the combination of the node with at least one of its attribute-value pairs is frequent [9].

In the remainder of this work, we will refer to these three cases as NOATR, NAIVE and ATR respectively. The two most commonly used tree inclusion notions, that handle attributes as described in the first two cases are:

Definition 1 *Given two labeled rooted trees T_1 and T_2 we call T_2 an* induced *subtree of T_1 and T_1 an* induced *supertree of T_2, denoted by $T_2 \preceq_i T_1$, if there exists an injective matching function Φ of V_{T_2} into V_{T_1} satisfying the following conditions for any $v, v_1, v_2 \in V_{T_2}$:*

1. *Φ preserves the labels: $L_{T_2}(v) = L_{T_1}(\Phi(v))$.*
2. *Φ preserves the order among the siblings: if $v_1 \leq_{T_2} v_2$ then $\Phi(v_1) \leq_{T_1} \Phi(v_2)$.*
3. *Φ preserves the parent-child relation: $(v_1, v_2) \in E_{T_2}$ iff $(\Phi(v_1), \Phi(v_2)) \in E_{T_1}$.*

Definition 2 *Given two labeled rooted trees T_1 and T_2 we call T_2 an* embedded *subtree of T_1 and T_1 an* embedded *supertree of T_2, denoted by $T_2 \preceq_e T_1$, if there exists an injective matching function Φ of V_{T_2} into V_{T_1}, satisfying the conditions* 1 *and* 2 *of definition* 1*. Additionally, Φ has the following property for any $v_1, v_2 \in V_{T_2}$:*

3'. Φ preserves the ancestor-descendant relation: if $(v_1, v_2) \in E_{T_2}$ then $\Phi(v_1)$ is an ancestor of $\Phi(v_2)$ in T_1.

The induced tree inclusion notion is used in the tree mining algorithm FREQT by Asai et. al [2] and the work by Termier et al. [12]. The embedded subtree relation is used in work by Zaki [13].

In the case were attributes are modeled as properties of nodes, an additional criterion to the definitions 1 and 2 is that the subtree relation should preserve the attributes; i.e., we add to definitions 1 and 2:

4. $\forall v \in V_{T_2}$: if $M(v) \neq \varnothing$ then $M(v) \subseteq M(\Phi(v))$.

These subtree relations are used in previous work of one of the authors [9].

In the remainder of this paper we use $\preceq$ to denote either an induced or embedded subtree relation. Let $D = \{d_1, \ldots, d_m\}$ denote a database where each record $d_i \in D$, is a labeled rooted ordered tree. For a given labeled rooted ordered tree T we say T occurs in a transaction d_i if T is a subtree of d_i. Let $\sigma_{d_i}(T) = 1$ if $T \preceq d_i$ and 0 otherwise. The support of a tree T in the database D is then defined as $\psi(T, D) = \sum_{d \in D} \sigma_d(T)$, that is the number of records in which T occurs one or more times. T is called frequent if $\psi(T, D)/|D|$ is greater than or equal to a user defined minimum support ($minsup$) value. The goal of frequent tree mining is to find all frequently occurring subtrees in a given database. Notice that ψ is an anti-monotone function: $T_i \preceq T_j \Rightarrow \psi(T_i, D) \geq \psi(T_j, D)$. The anti-monotonicity property of ψ is used to efficiently compute all the frequent subtrees of a database.

3 Research Question

The first question we consider is whether the predictive performance of a classifier constructed from the patterns is better when attributes are included in the mining process. In previous work we claimed that the NAIVE approach produces lots of uninformative patterns [9]. In particular, if a node has attributes associated to it in the database, a frequent pattern in which this node occurs without attributes is uninteresting. Clearly, following the NAIVE approach, for every frequent pattern where some nodes have attributes there is a corresponding frequent pattern without attributes attached to the node.

Furthermore, every pattern from either the NOAT or the ATR approach is also a pattern of the NAIVE approach. As such, one can expect that the predictive performance achieved by the NAIVE approach, is as least as good as the predictive performance of the other two approaches. So, the question arises if the patterns excluded by the "at least one attribute per node" constraint are valuable in terms of predictive performance. Finally, what is the added value of the embedded tree inclusion relation? Every pattern derived with the induced tree inclusion relation is also part of the patterns derived with the embedded tree inclusion relation. As such, in the ideal case the difference in performance between the induced and the embedded approach is caused by the patterns that are embedded patterns only. Summarizing, the questions are what effect it has on the different performance measures whether we:

1. Include attribute-value pairs or not.
2. When including attribute-value pairs; use the NAIVE or the ATR approach.
3. Use the induced or the embedded tree inclusion relation.

4 Methodology

To compare the different approaches on the selected performance measures, we first need to construct the classifiers. In this section we describe the procedure that, given the tree mining algorithm, constructs a classification model.

In general, the goal of frequent tree mining algorithms is to find all frequently occurring subtrees in a database. In case of constructing a classifier the objective is to find discriminative patterns, i.e. patterns that discriminate between the different classes. To do so, we use the support-confidence framework. Given a dataset $D = \{d_1, \ldots, d_m\}$ consisting of m records, each record belongs to exactly one class,where the class label is assigned from the set of class labels $C = \{c_1, \ldots, c_k\}$. With D_{c_i} we denote the set of records in the database that has class label c_i, likewise with $D_{\overline{c_i}}$ the set of records in the database is denoted that has a class label different from c_i. The first step of finding discriminating patterns, is to compute all frequently occurring subtrees within a class. A tree T is called frequent within the class c_i if $\psi(T, D_{c_i})/|D_{c_i}|$ is greater than or equal to a user defined minimum support ($minsup$) value. The computation of all frequent trees within a class over a database with multiple classes can be done simultaneously; only small changes to the original mining algorithms are needed.

The next step is to select from all frequent patterns within a class, those patterns that are good descriptors of the class. This is done by means of determining the confidence $P(c_i|T)$ of a rule $T \rightarrow c_i$, that is the probability of a class given the patterns. If this confidence is greater than 0.5, then T is regarded as a good discriminator for class c_i. Note that we did not optimize for the 'optimal' confidence value, nor did we use other measures such as the lift of a rule. The reason for this is, that our goal is to compare different tree matching relation by means of classification and not to build the best classifier. In the set of discriminating patterns found, there still might be redundancy between the patterns. This occurs when a pattern T_1 is a supertree of T_2 and $P(c_i|T_1) \leq P(c_i|T_2)$. In order to reduce the risk of overfitting on the training set and to limit the training time of the classification algorithms all redundant patterns were removed.

The resulting set of discriminating patterns are used as binary features in a standard classification algorithm. Each feature indicates the presence or absence of a discriminating pattern in a record. We used decision trees [11] to learn a classification model, more specifically the implementation provided by Borgelt [5]. This implementation uses a top down divide and conquer technique to build the decision tree. At each node in the decision tree, an attribute is selected—in a greedy manner—that best discriminates between the classes. As selection criterion, information gain was used. Additionally, after construction of the decision tree we used confidence level pruning to avoid overfitting. The parameter used with confidence level pruning was set to 50%. Finally, we estimated the area under the ROC curve. This was done with the method described in the work by Hand and Till [8], where an estimate for the area under the ROC curve is given that can be used for problems with multiple classes.

5 Experiments

In this section we describe the experiments that we performed to answer the research questions stated in section 3. For the two datasets, all minimum support thresholds used and every mining algorithm considered, we computed the following measures:

1. runtime in seconds: the time needed to compute all frequent patterns.
2. # of trees: the number of frequent trees.
3. # of features: the number of non-redundant discriminating patterns.
4. # of nodes in the decision tree: the number of nodes in the decision tree, this is used as a measure for the complexity of the classification model.
5. Accuracy: the ratio of the number of correctly classified documents and the total number of documents in the test set.
6. AUC : area under the ROC curve; this is a measure that compares the classification model with a classifier that has random performance [8]. This measure was also computed on the test set.

All estimates for the different measures were obtained using ten-fold cross-validation. In order to determine whether the difference in estimated area under the ROC curve and the accuracy measure for the different approaches were significant, the standard error for these two measures was computed.

5.1 Data sets

We performed the experiments on two real datasets namely the CSLOG dataset created and used by Zaki and Aggarwal [15] and the Wikipedia XML dataset [7].

The CSLOG dataset consist of user sessions of the RPI CS website, collected over a period of three weeks. Each user session consists of a graph and contained the websites a user visited on the RPI CS domain. These graphs were transformed to trees by only enabling forward edges starting from the root node. The goal of the classification task is to discriminate between users who come from the edu domain and users from another domain, based upon the user's browsing behavior. In total there are $23,011$ trees in the CSLOG dataset, where the average tree size is 8.02. This dataset does not contain any attributes (in the processed version). The relative frequency of the classes is 23.57% for the class that represents the edu domain and 76.43% for the other class.

The Wikipedia XML dataset was provided for the document mining track at INEX 2006. The collection consists of $150,094$ XML documents with 60 different class labels. The collection was constructed by converting web pages from the Wikipedia project to XML documents. The class labels correspond to the Wikipedia portal categories of the web pages. For the document mining track a subset of the total Wikipedia collection was selected such that each document belonged to exactly one class. In our experiments we only used the structure and the attributes of the XML documents. The algorithms that use the embedded subtree relation, were not able to run with the minimum support threshold set to 2% on a server having 4GB of main memory available. The reason for this is described in [6]: in the worst case, for the embedded subtree mining algorithm [13] the scopelist size is exponential in the size of the data tree. In order to include the tree mining algorithms that use the embedded subtree relation into the comparision, we selected only those documents where the number of nodes in the tree was less than 20. Compared with the average size of a tree in the whole collection (161.35) it is a drastic reduction. On the other hand compared with the average tree size of the most commonly used tree-structured dataset (8.02) it is still quite large. Additionally, we only used documents that belong to classes that contained more than 400 documents. The resulting reduced dataset consists of $24,222$ documents that are distributed over 13 classes, where the largest class contains about 22% of the documents and the smallest class about 3%. The average size of tree in the database equals 13.95 and the average number of attribute value pairs attached to a node equals 0.56. For the algorithm where the attribute value pairs are mapped to nodes, the average size of a tree in the database equals 21.73.

5.2 Result and analysis on the Wikipedia dataset

We start with an experiment to determine if the inclusion of attributes has any impact on the quality of the classifiers. Comparing the results of the two methods that include attribute-value pairs into the mining process (ATR and NAIVE) with the algorithm where the attributes were left out (NOATR), the former have a substantially higher predictive performance on the Wikipedia dataset. Both the accuracy and the area under the ROC curve are significantly higher, as shown in figure 2. This result holds regardless of the tree inclusion relation or minimum support value used.

A closer investigation of the patterns used for the ATR and NAIVE classifier, reveals that the main advantage of including attribute values is that these values often describe the document to which a link points. Consider for example the discriminating pattern shown in figure 1. This pattern has very common structural properties, but due to the attribute values it becomes a discriminating pattern. In the example shown, the attribute value points to a picture that was the US Navy Jack for some time. Clearly, the values is a good indicator of its class ("Portal:War/Categories").

A further remarkable difference between the mining algorithms for the ATR and the NOATR approach, is the number of frequent trees both produce. Although less information is included, the NOATR approach computes far more frequent trees than its counterpart. As a consequence extra computing time is needed for the NOATR mining algorithms. The reason for the increase in frequent trees is that contrary to the ATR approach, the same node labels with different attribute-value pairs are counted as one pattern in the NOATR approach. For example, almost every document in the Wikipedia

<figure>

</figure>

Fig. 1. A discriminating pattern found on the training set. This pattern, describing class 2879927 ("Portal:War/Categories") has a support of 48 in its class and of 0 for all other classes.

collection contains a couple of nodes of type "wikipedia link". These nodes differ only in the attribute-value pairs attached to them. Consequently, in the NOATR approach this will be a highly frequent pattern. This effect is especially noticeable in case the embedded subtree relation is used: the NOATR embedded approach has almost four times as many frequent trees as the ATR embedded approach. On the other hand, for most support values, this is limited to almost two times as many in case the induced subtree relation is used. However, with regard to the number of non-redundant discriminating patterns, the NOATR approach with the induced subtree relation, produces less discriminating patterns than its counterpart with attributes. The ratio between number of discriminating patterns and the number of frequent trees is between $1/18$ and $1/38$ for the ATR approach, while for the NOATR approach the ratio is between $1/58$ and $1/83$. For the embedded subtree relation these ratios are respectively between $1/38$ and $1/53$ and between $1/107$ and $1/149$.

The difference in predictive performance between the classifiers produced with the induced and embedded tree inclusion relation, for the ATR approach is very small, both in terms of accuracy and area under the curve. Although the number of frequent trees and the number of non-redundant discriminating patterns is respectively about two and one and a half times higher for the embedded subtree relation, the complexity of the classification model (measured by the number of nodes in the decision tree) is slightly lower for most cases. For the NOATR classifier, the accuracy and area under the curve estimates achieved are slightly worse for the embedded subtree relation than for the induced subtree relation. Furthermore, the classifier based on the embedded subtrees is by far the most complex classification model for the lower support values.

The patterns produced by the NAIVE mining algorithm are a superset of both the patterns computed with the ATR approach and those computed with the NOATR approach. But, contrary to expectation the results obtained in terms of accuracy are for all but one setting considerably below the results obtained via the ATR approach. And with respect to the area under the curve measure, the performance of the NAIVE approach is for all settings significantly below the performance of the ATR approach. We can not give a complete explanation for this, although with the embedded subtree relation, the classification model is more complex than the classification models based on patterns computed with the ATR approach. This might be an indication for overfitting, however this observation does not hold for the induced subtree relation. Notice, that the features produced by the NAIVE approach are not necessarily a superset of the features produced by the ATR approach. This is because of redundancy pruning.

Furthermore, worth noticing are the number of frequent trees the NAIVE approach produces. Especially in the case that the embedded subtree relation is used, the number of frequent trees and the time needed to compute these are extremely high compared

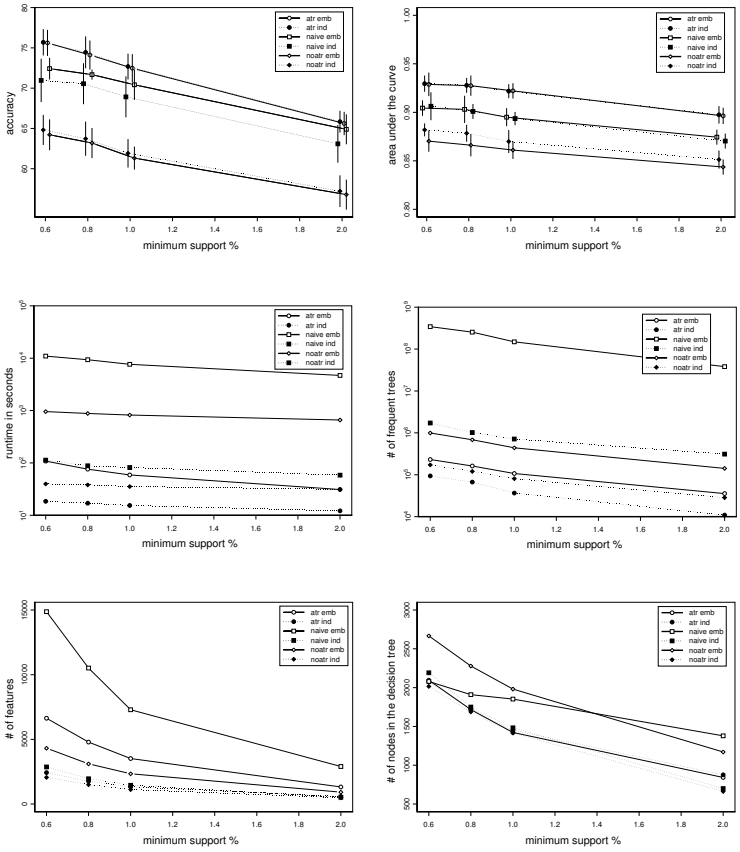


Fig. 2. Results obtained on the Wikipedia dataset. From left to right, top to bottom: accuracy estimates, estimates for the area under the curve, runtime in seconds (log scale), number of frequent trees (log scale), number of non-redundant discriminating patterns, and the number of nodes used in the decision tree. In the two plots on top (accuracy estimates and AUC estimates), for each estimated value we also show $\text{value} \pm 2\text{SE}$, this is shown as vertical bars going through the points. Furthermore, the points in these two plots, are shifted a bit to the right or left, such that the different points and vertical bars could be distinguished.The different lines indicate different combinations of the tree inclusion relation and attribute handling. For example, atr emb denotes the embedded tree inclusion relation with ATR attribute handling.

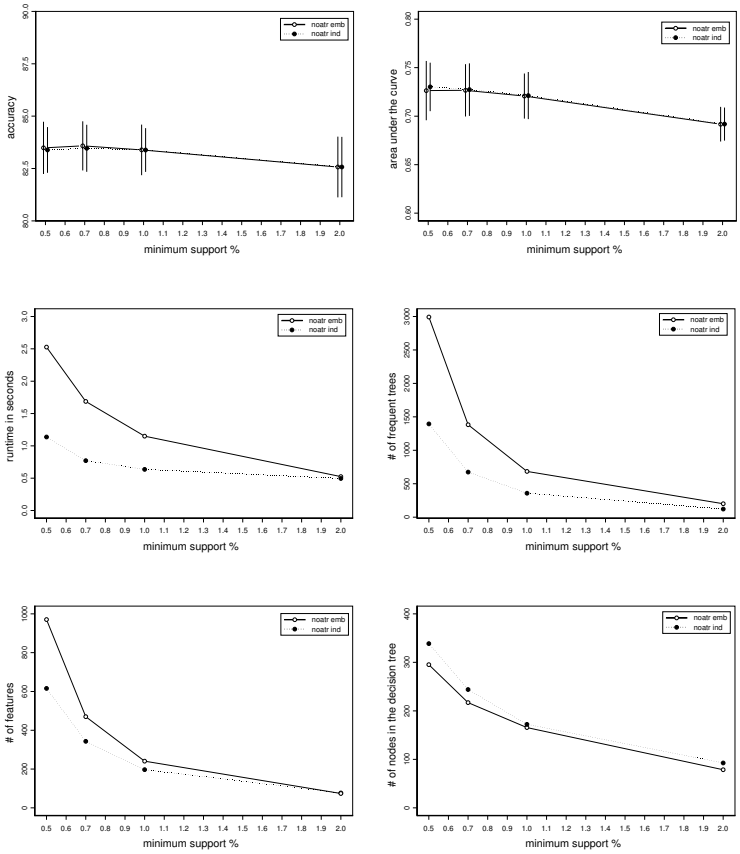


Fig. 3. Results obtained on the Weblog dataset. From left to right, top to bottom: accuracy estimates, estimates for the area under the curve, runtime in seconds (log scale), number of frequent trees (log scale), number of non-redundant discriminating patterns, and the number of nodes used in the decision tree. In the two plots on top (accuracy estimates and AUC estimates), for each estimated value we also shown $\text{value} \pm 2\text{SE}$, this is shown as vertical bars going through the points. Furthermore, the points in these two plots, are shifted a bit to the right or left, such that the different points and vertical bars could be distinguished.

to the other approaches. Furthermore, the ratio between the number of frequent trees and the number of non-redundant discriminating patterns ranges between $1/491$ and $1/595$ for the induced tree inclusion relation and between $1/13146$ and $1/24109$ for the embedded inclusion relation. In comparison with both the ratios of the ATR and the NOATR approach, the NAIVE approach proportionally spends a great effort in computing patterns that will be discarded in a later stage. With respect to AUC, the difference between the induced and embedded subtree relation for the NAIVE approach is negligible. However in terms of accuracy, the embedded subtree relation performs for better than the induced subtree relation.

For all performed experiments on the Wikipedia data set it holds that the difference in predictive performance between the induced and embedded tree inclusion relation is small, and well within the range of the standard error of the estimated accuracy and the are under the ROC curve.

With regard to the run time we earlier noticed that the NAIVE algorithm needs considerably more time than both the ATR and the NOATR approaches, and in turn the NOATR approach consumes some extra time in comparison with the ATR approach. Furthermore, for the embedded tree inclusion relation, considerably more time is needed to compute all frequent trees than with the induced tree inclusion relation. Instead of the total run time, often the average time per tree is derived to compare algorithms. Although this would change the comparison completely, it is for our goal not an interesting measure: we are not interested in the efficiency of an algorithm per se, but in the computation time needed in order to obtain a certain result. Another performance issue is the memory usage of the different approaches. As noted earlier, the Wikipedia dataset had to be reduced such the mining algorithms that uses the embedded subtree relation could be executed with an upper bound of 4GB. For the reduced Wikipedia dataset, the memory usage of all approaches with the induced subtree relation was constant for all minimum support values. These values are respectively 26MB, 46MB and 48MB for the NOATR, ATR and NAIVE approach. For the embedded subtree relation we only measured the worst case, i.e. the case with the lowest minimum support value. For this case the different approaches (NOATR, ATR, NAIVE) used respectively 339MB, 211MB and 520MB. The difference in memory usage is caused by the fact that the only known implementations of the embedded subtree mining algorithm, need to store for every node in the pattern tree a reference to corresponding values in the database. Hence, the larger the pattern tree becomes, the more values (links) need to be stored. This is also the reason why the embedded NAIVE approach uses more main memory than the other embedded approaches: the NAIVE approach has larger frequent patterns. For the induced subtree relation, known implementations need to store only a link to all references in the database from the rightmost node of the pattern tree. Hence, the memory requirement does not depend on the size of the frequent tree but only on the size of the database.

5.3 Results and analysis on the weblog data

The results of the experiments on the weblog data are displayed in figure 3. In comparison with the Wikipedia dataset, the weblog data is relatively simple: both the number of frequent trees and the number of non-redundant discriminating patterns are in comparison quite modest. Also the fact that there are only two classes makes the classification

task less complicated. With the embedded subtree relation, both the number of frequent trees and the number of discriminating patterns produced is larger than with the induced subtree relation. However, the accuracy score is, for the lowest support values used, slightly higher for the former. Also in this case, this somewhat higher accuracy score is gained with slightly less features in the classification model. With regard to the area under the ROC curve, the induced subtree relation performs slightly better for the two lowest minimum support values. The run time needed for both tree inclusion relations on this training set is quite small, but is slightly worse for the embedded tree inclusion relation. Also the memory usage of the algorithms is modest: for respectively the embedded and induced approach with the lowest minimum support value it equals 30MB and 25MB.

6 Discussion and Conclusion

In this paper we experimentally compared frequent tree mining algorithms using different tree inclusion definitions, and different ways of handling attributes. Besides the traditional comparison in terms of run time and memory usage, we also constructed classifiers from frequent patterns produced by the different approaches, and compared these classifiers on predictive performance and model complexity.

The three main conclusions of this work are:

1. The inclusion of attribute-values pairs in the mining process significantly improves the classification result.
2. The ATR approach performs considerably better than the NAIVE approach, both in terms of run time and predictive performance.
3. The embedded tree inclusion relation does not result in better classification performance, nor in simpler classification models and has major computational disadvantages.

The results show that significant performance gain can be obtained by including attribute value-pairs into the mining algorithms. Especially the case where attributes are modeled as properties of a node, such as in the ATR approach, delivers good results. But, also in the case where attribute value-pairs are added as elements to the dataset, the results are far better than in case the attributes are ignored.

How the attribute value-pairs are handled in the mining process, is also of great importance for both the predictive performance as well as the runtime. The results show that for all but one minimum support value used, both the accuracy and the area under the ROC curve are considerably better for the ATR approach than for the NAIVE approach. Furthermore, the NAIVE approach requires substantially extra computation time. We conclude from this that, the constraint of “at least one attribute per node” makes sense from both a computational and a usability point of view.

The experiments show no significant difference in performance between the induced an embedded subtree relation. Contrary to the common expectation, the embedded tree inclusion relation often results in a slightly lower predictive performance than the induced subtree relation. Furthermore, a major practical limitation is the excessive memory usage of currently known embedded tree mining algorithms. Even if this implementation issue is solved, then still the embedded approach has as major disadvantage that

more computation time is needed, which is spent on patterns that will be thrown away in a later stage.

Although, it is of great advantage to construct less complex classification models that preserve equivalent predictability, the experiments with the embedded tree inclusion relation, did not result in less complex classification models. Concluding, the addition of "hidden patterns" did, for the examined datasets, not result in better predictive performance.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499, 1994.
2. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *SIAM Symposium on Discrete Algorithms*, 2002.
3. T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering frequent substructures in large unordered trees. In *Discovery Science*, pages 47–61, 2003.
4. R. Bathoorn, A.Koopman, and A. Siebes. Reducing the frequent pattern set. In *Workshops Proceedings of ICDM 2006*, pages 55–59, 2006.
5. C. Borgelt. A decision tree plug-in for dataengine. In *Proc. 6th European Congress on Intelligent Techniques and Soft Computing*, 1998.
6. Y. Chi, R. Muntz, S. Nijssen, and J. Kok. Frequent subtree mining - an overview. *Fundamenta Informaticae.*, 66(1-2):161–198, 2005.
7. L. Denoyer and P. Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 2006.
8. D. J. Hand and R. J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.
9. J. De Knijf. FAT-miner: Mining frequent attribute trees. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, 2007.
10. S. Nijssen and J.N. Kok. Efficient discovery of frequent unordered trees. In *In Proceedings of the first International Workshop on Mining Graphs, Trees and Sequences (MGTS2003)*, pages 55–64, 2003.
11. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
12. A. Termier, M. Rousset, M. Sebag, K. Ohara, T. Washio, and H. Motoda. Efficient mining of high branching factor attribute trees. In *IEE International Conference on Data Mining*, pages 785–788, 2005.
13. M. J. Zaki. Efficiently mining frequent trees in a forest. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2002.
14. M. J. Zaki. Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae.*, 66(1-2):33–52, 2005.
15. M. J. Zaki and C. C. Aggarwal. Xrules: an effective structural classifier for XML data. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316–325, 2003.

Mining Frequent Patterns from Multi-Dimensional Relational Sequences

Nicola Di Mauro, Teresa M.A. Basile, Stefano Ferilli, and Floriana Esposito

Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy

Abstract. The problem addressed in this paper regards the discovering of frequent multi-dimensional patterns from relational sequences. In a multi-dimensional sequence each event depends on more than one dimension, such as in spatio-temporal sequences where an event may be spatially or temporally related to other events. In literature, the multi-relational data mining approach has been successfully applied to knowledge discovery from complex data. This work takes into account the possibility to mine complex patterns, expressed in a first-order language, in which events may occur along different dimensions. A complete framework and an *Inductive Logic Programming* algorithm to tackle this problem is presented with preliminary experiments focussing on artificial multi-dimensional sequences.

1 Introduction

The great variety of applications of sequential pattern mining, such as user profiling, medicine, local weather forecast and bioinformatics, makes this problem one of the central topics in data mining as showed by the research efforts produced in recent years [1, 22, 7, 17, 18]. Sequential information may concern data on multiple dimensions and, hence, the mining of sequential patterns from multi-dimensional information results very important. The first work on mining multi-dimensional patterns has been presented in 2001 by Pinto et al. [18]. However, all the works in multi-dimensional data mining have been restricted to the propositional case, not involving a first-order representation formalism.

Some works facing the problem of knowledge discovery from spatial and temporal data in the multi-relational data mining research area [15, 19, 5, 20, 12] are present in literature, but there exists no contributions to manage the general case of multi-dimensional data in which, for example, spatial and temporal information may co-exist.

In this paper an Inductive Logic Programming (ILP) [16] algorithm for discovering *first-order* (DATALOG) *maximal frequent patterns* in *multi-dimensional* relational sequences is provided. Multi-dimensional patterns are defined as a set of atomic first-order formulae in which events are explicitly represented by a variable and the relations between events are represented by a set of dimensional predicates (next, follow, follow-at).

Although encoding temporal predicates in ILP is very simple, making a system able to understand and use their semantic is crucial for efficiency. Some

recent works on mining logical patterns [9, 11, 13, 4] take into account temporal sequences (i.e., 1-dimensional sequences) by using a purposely defined logical temporal formalism. Instead, this work proposes a dedicated framework which incorporates a specific language bias for multi-dimensional data, expressed in a first-order logic, in order to rise a faster execution and a smaller search space. The first-order logical representation gives us the possibility to encode temporal, spatial and other dimensional spaces without requiring to discriminate between them. Furthermore, it is possible to represent any other domain relations and let them to co-exist with other dimensional ones.

An interesting application of multi-dimensional logical pattern mining is modelling. A logical formalism for mining temporal patterns in a task of user modelling has been proposed in [8] in which the user behaviour is described according to the temporal sequences of his actions. The approach proposed in this paper allows us to tackle many complex scenarios such as context modelling, in which a situation and the actors involved in it evolve both in time and space. For instance, we should think to profile a user accessing to a room (home, office, museum, etc.) by describing contextual information (such as position in the room described by two spatial dimensions) and temporal information.

2 Mining Multi-Dimensional Patterns

We used DATALOG [21] as representation language for the domain knowledge and patterns, that here is briefly reviewed. A *term* is defined as a constant symbol or a variable. An atom $p(t_1, \ldots, t_n)$ is a predicate p of arity n applied to n terms t_i. A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \ldots, X_n \leftarrow a_n\}$ where $X_i, 1 \leq i \leq n$ is a variable and $a_i, 1 \leq i \leq n$ is a term. A substitution θ is applicable to an expression e, obtaining the expression $e\theta$, by replacing all variables X_i with their corresponding terms a_i.

Definition 1. *A 1-dimensional relational sequence may be defined as an ordered list of* DATALOG *atoms separated by the operator* $<$, $l_1 < l_2 < \cdots < l_n$.

Example 1. The following list of DATALOG atoms

$\mathsf{p(a,b)} < \mathsf{p(b,c)} < \mathsf{p(c,a)} < \mathsf{p(b,b)}$

represents a 1-dimensional sequence. In general, for this kind of sequences, referring to one dimension only, the operator $<$ can be omitted as follows

$\mathsf{p(a,b)}\ \mathsf{p(b,c)}\ \mathsf{p(c,a)}\ \mathsf{p(b,b)}$

where is implicit, for instance, that the atom $\mathsf{p(b,c)}$ follows the atom $\mathsf{p(a,b)}$.

Hovewer, in order to make the proposed framework more general, the concept of *fluents* introduced by J. McCarthy in [14] should be considered: "After having defined a *situation*, s_t, as the complete state of the universe at an instant of time t, then a fluent is defined as a function whose domain is the space of situations. In particular, a *propositional fluent* ranges in (true,false). For example, raining(x, s_t) is true if and only if it is raining at the place x in the situation s_t."

In our description language we can distinguish two kinds of DATALOG atoms: *dimensional* and *non-dimensional* atoms. Specifically:

- non-dimensional atoms, that may be divided into
  - *fluent atoms*: explicitly referring to a given event (i.e., in which one of its argument denotes an event);
  - *non-fluent atoms*: denoting relations between objects (with arity greater than 1), or characterizing an object (with arity 1) involved in the sequence;
- dimensional atoms: referring to dimensional relations between events involved in the sequence.

Example 2. The following set of DATALOG atoms

$\mathsf{p}(e_1,\mathsf{a},\mathsf{b})$ $(e_1 < e_2)$ $\mathsf{p}(e_2,\mathsf{b},\mathsf{c})$ $\mathsf{q}(\mathsf{b},\mathsf{c})$

denotes a 1-dimensional sequence with three non-dimensional atoms and one dimensional atom. Specifically, $\mathsf{p}(e_1,\mathsf{a},\mathsf{b})$ denotes the fluent $\mathsf{p}(\mathsf{a},\mathsf{b})$ at the event e_1, $\mathsf{p}(e_2,\mathsf{b},\mathsf{c})$ denotes the fluent $\mathsf{p}(\mathsf{b},\mathsf{c})$ at the event e_2, $(e_1 < e_2)$ indicates that the event e_2 is the direct successor of e_1 and $\mathsf{q}(\mathsf{b},\mathsf{c})$ represents a generic relation between the objects b and c.

Another way to read the previous example is the following: " $\mathsf{p}(\mathsf{a},\mathsf{b})$ is true in the event e_1, the event e_1 gives rise to the event e_2 where is true $\mathsf{p}(\mathsf{b},\mathsf{c})$, and there is a relation q between b and c".

The choice to add the event as an argument of the predicates is necessary in the general case of n-dimensional sequences with $n > 1$. In this case, indeed, the operator $<$ is not sufficient to express multi-dimensional relations and we must use its general version $<_i, 1 \leq i \leq n$. Specifically, $(e_1 <_i e_2)$ denotes that the event e_1 gives rise to the event e_2 in the dimension i. Hence, in our framework a multi-dimensional data is supposed to be a set of events, and to each dimension corresponds a sequence of events.

Definition 2. *A multi-dimensional relational sequence is a set of* DATALOG *atoms, involving k events and regarding n dimensions, in which there are non-dimensional atoms (fluents and non-fluents) and each event may be related to another event by means of the $<_i$ operators, $1 \leq i \leq n$.*

After having defined what is a logical multi-dimensional sequence, in the following we give a detailed description of the dimensional operators used to describe multi-dimensional patterns.

2.1 Multi-Dimensional Patterns

In order to represent multi-dimensional patterns, some dimensional operators must be introduced. The following symbols for describing general event relationships along many dimensions has been adopted. In particular, given a set $\mathcal{D}$ of dimensions, in the following are reported the multi-dimensional operators:

- $<_i$: *next step on dimension* $i, \forall i \in \mathcal{D}$. This operator indicates the direct successor on the dimension i. For instance, $(x <_{time} y)$ denotes that the event y is the direct successor of the event x on the dimension *time*. $\mathsf{next\_i}/2$ is the corresponding Datalog predicate used to denote the successor operator;

- $\lhd_i$: *after some steps on dimension* $i, \forall i \in \mathcal{D}$. This operator encodes the transitive closure of $<_i$. For example, $(y \lhd_{spatialx} z)$ states that the event z occurs somewhere after the event y on the dimension *spatialx*. follows_i/2 is the corresponding Datalog representation;
- $\bigcirc_i^n$: *exactly after* n *steps on dimension* $i, \forall i \in \mathcal{D}$. In particular it calculates the n-th direct successor. For instance, $(x \bigcirc_{spatialz}^n w)$ states that the event w is the n-th direct successor of the event x on the dimension *spatialz*. The follows_at_1/3 Datalog predicate is used to represent such a situation.

The dimensional characteristics in the sequences will be described by using the $<_i$ operator, while the two dimensional operators $\lhd_i$ and $\bigcirc_i^n$, will be used, in combination with $<_i$ operator, to represent the frequent discovered patterns.

Example 3. With the above defined dimensional operators, an example of a simple temporal sequence could be:

p(e_1,a,b) ($e_1 <_{time} e_2$) q(e_2,b,c) ($e_2 <_{time} e_3$) p(e_3,e,f) ($e_3 <_{time} e_4$) q(e_4,f,g)

and the relative temporal patterns that may be true when applied to it are

p(E_1,X,Y) ($E_1 <_{time} E_2$) q(E_2,Y,Z)
p(E_1,X,Y) ($E_1 \lhd_{time} E_2$) q(E_2,Z,W)
p(E_1,X,Y) ($E_1 \bigcirc_{time}^2 E_2$) p($E_2$,Z,W)

In general, given a sequence $\sigma = (e_1 e_2 \cdots e_m)$ of m elements, a sequence $\sigma' = (e'_1 e'_2 \cdots e'_k)$ of length k is a *subsequence* (or pattern) of σ if for a given $h < m - k$: $e_{h+i} = e'_{i+1}$, $1 \leq i \leq k$. The frequency of a subsequence in a sequence is the number of all the possible values of h such that the previous condition holds.

We are interested in finding maximal frequent patterns with a high frequency in long sequences. A pattern σ' of a sequence σ is *maximal* if there is no pattern σ'' of σ more frequent than σ' and such that σ' is a subsequence of σ''.

Definition 3. *A multi-dimensional relational pattern is a set of* DATALOG *atoms, involving* k *events and regarding* n *dimensions, in which there are non-dimensional atoms and each event may be related to another event by means of the operators* $<_i$, $\lhd_i$ *and* $\bigcirc_i^n$ *operators,* $1 \leq i \leq n$.

2.2 The algorithm

In this section we describe the algorithm for frequent pattern discovery based on the same idea of the generic level-wise search method, known in data mining from the APRIORI algorithm [1]. The level-wise algorithm makes a breadth-first search in the lattice of patterns ordered by a specialization relation $\preceq$. The search starts from the most general patterns, and at each level of the lattice the algorithm generates candidates by using the lattice structure and then evaluates the frequencies of the candidates. In the generation phase, some patterns are taken out using the monotonicity of pattern frequency (if a pattern is not frequent then none of its specializations is frequent).

The main method is outlined in Algorithm 1. The generation of the frequent patterns is based on a top-down approach. The algorithm starts with the most general patterns. These initial patterns are all of length 1 and are generated by adding to the empty pattern a non-dimensional atom. Successively, at each step it tries to specialize all the potential frequent patterns, discarding the non-frequent patterns and storing the ones whose length is equal to the user specified input parameter *maxsize*. Furthermore, for each new refined pattern, semantically equivalent patterns are detected, by using the θ-subsumption relation, and discarded. Note that the length of a pattern is defined as the number of non-dimensional atoms. In the specialization phase, the specialization operator under θ-subsumption is used. Basically, the operator adds atoms to the pattern.

Algorithm 1 MDLS

Require:
maxsize: maximal pattern length (i.e., the maximum number of non-dimensional predicates appearing in the pattern);
minfreq: the threshold;
Ensure: P_{max}: the set of maximal frequent patterns

```
1: P ← { initial patterns }
2: P_max ← ∅
3: while P ≠ ∅ do
4:    P_s ← ∅
5:    for all p ∈ P do
6:       {generation step}
7:       P_s ← P_s ∪ {all the specializations of p }
8:    P ← ∅
9:    for all p ∈ P_s do
10:      {evaluation step}
11:      if freq(p) ≥ minfreq then
12:         if length(p) = maxsize then
13:            P_max ← P_max ∪ {p}
14:         else
15:            P ← P ∪ {p}
```

In particular, given the set $\mathcal{D}$ of dimensions, the set $\mathcal{F}$ of fluent atoms, the set $\mathcal{P}$ of non-fluent atoms, the refinement operator for specializing patterns is defined as follows:

adding a non-dimensional atom
- the pattern S is specialized by adding a non-dimensional atom $F \in \mathcal{F}$ (a fluent) referring to an event already introduced in S;
- the pattern S is specialized by adding a non-dimensional atom $P \in \mathcal{P}$;

adding a dimensional atom
- the pattern S is specialized by adding the dimensional atom $(x <_i y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S with x as

its event argument and there not exist the atoms $(x \lhd_i y)$ and $(x \bigcirc_i^n y)$ in S;
- the pattern S is specialized by adding the dimensional atom $(x \lhd_i y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S with x as its event argument and there not exist the atoms $(x <_i y)$ and $(x \bigcirc_i^n y)$ in S;
- the pattern S is specialized by adding the dimensional atom $(x \bigcirc_i^n y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S with x as its event argument and there not exist the atoms $(x <_i y)$ and $(x \lhd_i y)$ in S.

The dimensional atoms are added iff there exists a fluent atom referring to its starting event. This is to avoid unuseful chains of dimensional predicates like this $\mathsf{p}(e_1,\mathsf{a})$ $(e_1 <_i e_2)$ $(e_2 <_i e_3)$ $(e_3 <_i e_4)$, that is naturally subsumed by $\mathsf{p}(e_1,\mathsf{a})$ $(e_1 \bigcirc_i^3 e_4)$.

As regards the language bias, classical mode and type declarations are used to specify which predicates can be used in patterns and to formulate constraints on the binding of variables.

3 Experiments

In order to evaluate the proposed technique we made preliminary experiments, applying the algorithm to a simple example about 3D data and to an artificial dataset.

3.1 3D example: Cellular automaton data

In the following we evaluated the algorithm on the best-known example of a cellular automaton, named *The Game of Life*, devised by J.H. Conway in 1970 [6]. This simulation game resembles the processes of a society of living organisms.

The universe of the game involves a plane, assumed to be infinite, divided into cells, each of which is in one of two possible states, *live* – meaning that there is an organism – or *dead* . The idea is to start with a simple configuration of organisms and then observe how it changes as one applies the "genetic laws" for births, deaths, and survivals. Note that each cell of the plane has eight neighboring cells, four adjacent orthogonally and four adjacent diagonally. The rules are:

- **Births**: each empty cell adjacent to exactly three neighbors is a birth cell. An organism is placed on it in the next population;
- **Survivals**: every organism with two or three neighboring organisms survives for the next generation;
- **Deaths**: each organism with four or more neighbors dies (is removed) for overpopulation. Every organism with one neighbor or none dies for isolation.

Note that all births and deaths occur simultaneously.

We can model the plane by using two dimensions (say x and y), while the time may be modeled by another dimension (say t). The plane containing the

organisms has been viewed as a two-dimensional array. Hovewer, since the plane is in principle infinite, its left and right edges are considered to be stitched together, like the top and bottom edges, thus yielding a toroidal array.

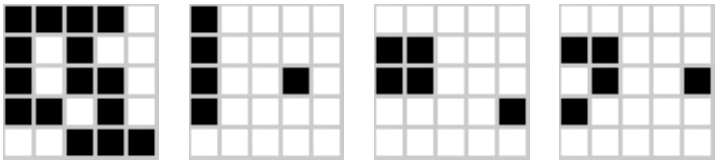


Fig. 1. A sequence of evolving populations

In Figure 1 is reported a sequence of evolving populations, form an initial population of 25 organisms, that can be described in the defined domain language as follows.

/* 1st population */
live(f_1) ($f_1 <_x f_2$) live(f_2) ...
($f_1 <_y f_6$) live(f_6) ($f_6 <_x f_7$) ($f_2 <_y f_7$) dead(f_7) ...
($f_6 <_y f_{11}$) live($f_1 1$) ($f_1 1 <_x f_{12}$) ($f_7 <_y f_{12}$) dead(f_{12}) ...
/* 2nd population */
live(s_1) ($s_1 <_x s_2$) dead(s_2) ...
($s_1 <_y s_6$) live(s_6) ($s_6 <_x s_7$) ($s_2 <_y s_7$) dead(s_7) ...
($s_6 <_y s_{11}$) live($s_1 1$) ($s_1 1 <_x s_{12}$) ($s_7 <_y s_{12}$) dead(s_{12}) ...
/* 3th population */
...
/* 4th population */
...
/* temporal relations */
($f_1 <_t s_1$) ($f_2 <_t s_2$) ($f_3 <_t s_3$) ($f_4 <_t s_4$) ($f_5 <_t s_5$)...

We used the operators $<_x$ and $<_y$ to indicate that an event is a direct successor, respectively, in horizontal and in vertical direction. While, the operator $<_t$ represents direct successor of an event along the time dimension. In particular, this kind of 3-dimensional sequences combines spatial and temporal data.

Executing the algorithm on some artificial population we obtained many different multi-dimensional patterns, including *still lifes*, and *oscillators*[1] as that reported in Figure 2. The *block* and *boat* patterns are still lifes, while the *blinker* is a two-phase oscillator.

[1] In cellular automata, a still life is a pattern that does not change from one generation to the next, while, an oscillator is a pattern that returns to its original state, in the same orientation and position, after a finite number of generations.

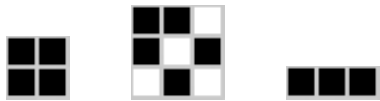


Fig. 2. Some patterns occur in the Game of Life

block

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (A <_y C)\ \mathsf{live}(C)\ (C <_x D)\ (B <_y D)\ \mathsf{live}(D)$
$(A <_t A')\ \mathsf{live}(A')\ (A' <_x B')\ \mathsf{live}(B')\ (A' <_y C')\ \mathsf{live}(C')\ (C' <_x D')$
$(B' <_y D')\ \mathsf{live}(D')$

boat

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (A <_y C)\ \mathsf{live}(C)\ (C \bigcirc_x^2 D)\ \mathsf{live}(D)\ (B \bigcirc_y^2 E)$
$\mathsf{live}(E)\ (A <_t A')\ \mathsf{live}(A')\ (A' <_x B')\ \mathsf{live}(B')\ (A' <_y C')\ \mathsf{live}(C')\ (C' \bigcirc_x^2 D')$
$\mathsf{live}(D')\ (B' \bigcirc_y^2 E')\ \mathsf{live}(E')$

blinker

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (B <_x C)\ \mathsf{live}(C)$
$\mathsf{dead}(D)\ (D <_y B)\ \mathsf{dead}(E)\ (B <_y E)\ (B <_t B')\ \mathsf{dead}(A')\ (A' <_x B')$
$\mathsf{live}(B')\ (B' <_x C')\ \mathsf{dead}(C')\ \mathsf{live}(D')\ (D' <_y B')\ \mathsf{live}(E')\ (B' <_y E')$

3.2 Artificial relational data

In order to evaluate the proposed algorithm on more convincing data, a random problem generator has been implemented and used to generate multi-dimensional relational sequences. In particular, it randomly generates a sequence containing a frequent pattern taking as input the following parameters. The domain language is defined by a set $\mathcal{D}$ of d dimensions, a set $\mathcal{R}$ of r binary predicates, and a set $\mathcal{F}$ of f fluent predicates with arity 3. By using these predicates, a sequence, made up of Es events and Os objects, is generated by randomly selecting Rs relational literals and Fs fluent literals per event. A relational literal is generated by randomly selecting its predicate from $\mathcal{R}$ and randomly selecting its arguments from the set of Os objects. For each event, Fs fluent literals are generated by randomly selecting their predicates from $\mathcal{F}$ and randomly selecting its two relational arguments from the set of Os objects. The sequence contains $freq$ patterns with the same logical structure, made up of Ep events and Op objects. Each pattern contains Fp fluents literals per event and Rp relational literals randomly generated by using the above method.

Two kind of problems, $\mathcal{P}_1$ and $\mathcal{P}_2$, have been generated, with r and f set to 3, Fs and Fp set to 1. In the former we fixed the length of the pattern, while in the latter we fixed the length of the sequence. In particular, the problem $\mathcal{P}_1$ has been divided into 5 sub-problems, where the number of events Es of the sequence has been set, respectively, to 100, 200, 300, 400 and 500, while the number of events Ep of the pattern has been fixed to 4. The problem $\mathcal{P}_2$ has been divided into 5 sub-problems, where the number of events Ep of the pattern

Table 1. Warmr and MDLS performances (time in secs.).

	p_1		p_2		p_3		p_4		p_5	
	Warmr	MDLS	Warmr	MDLS	Warmr	MDLS	Warmr	MDLS	Warmr	MDLS
1D	5,17	1,46	5,32	2,33	5,0t, F0	2,66	5,85	3,92	6,44	5,52
$\mathcal{P}1$ 2D	3,75	1,37	4,23	1,97	4,22	2,84	3,68	3,38	4,36	4,59
3D	3,98	1,32	3,46	1,80	4,00	2,72	4,08	3,47	4,02	4,04
1D	5,82	1,53	12,22	3,14	26,52	5,83	45,84	9,3	63,79	13,59
$\mathcal{P}2$ 2D	4,46	1,43	8,61	2,77	19,62	5,07	37,41	8,52	62,23	14,47
3D	3,78	1,16	10,30	2,84	18,68	5,13	38,52	9,06	66,67	14,57

has been set, respectively, to 4, 5, 6, 7 and 8, fixing the number of events Es of the sequence to 100. For each sub-problem 10 sequences have been generated.

Our system has been compared to Warmr [5], using the package ACE-ilProlog [2] kindly made available by Hendrik Blockeel. Table 1 reports the mean time, over the 10 sequences for each sub-problem (p_i, $1 \leq i \leq 5$), by executing both Warmr and MDLS. For each sub-problem of $\mathcal{P}_1$ we fixed $Ep = 4$, $Op = 3$ and $Rp = 2$, while the others parameter have been set, respectively, as follows $Es = 100, 200, 300, 400, 500$, $Os = 10, 20, 30, 40, 50$, $Rs = 40, 60, 80, 100, 120$, $freq = 10, 20, 30, 40, 50$. While, for the problem $\mathcal{P}_2$ we fixed $Es = 100$, $Os = 10$ and $Rs = 40$, $Ep = 4, 5, 6, 7, 8$, $Op = 3$, $Rp = 2$, $freq = 10$. t, F The first column of Table 1 indicates the kind of sequence (1D, 2D, 3D) for each problem, while the others the mean time in seconds for each corresponding sub-problem. As one can see, MDLS outperforms Warmr that is limited with respect to the length of the pattern. Indeed, the time increases as the length of the pattern grows, as reported for the the problem $\mathcal{P}_2$.

4 Related Work

In this section we will review some recent work on mining logical sequences.

In [9] is presented a work, in the domain of user modelling, that helps shell users by creating scripts (a sequence of commands) from shell logs, that automate frequent performed tasks. The authors see this task as a relational learning problem, indeed commands may be interrelated by their execution order, and each command is possibly related to one or more parameters, giving out a representation of a shell log as a set of logical ground atoms. After having transformed shell logs in a relational representation, they applied the Warmr [5] system, an upgrade of the propositional Apriori algorithm that can detect first order logic association rules, for generating scripts. They used a specific predicate to specify that two commands are considered next to each other in a sequence.

Warmr [5] is based on the level-wise search of conventional association rule learning systems of the Apriori-family [1]. It extends these systems by looking for frequent patterns that may be expressed as conjunction of first-order literals. In Warmr a pattern is defined as a conjunction of first order literals. It performs a top-down level-wise search, starting with the key and refining patterns by adding

literals to them. Infrequent patterns (i.e. patterns whose frequency is below a predefined threshold) are pruned as are their refinements. With Warmr it is possible to generate patterns that are syntactically different but semantically equivalent. This is due to the redundant conditions that may be added to a pattern or to the fact that the same pattern may be expressed in different ways.

As already described in [3], this problem may be avoided by using the Warmr's configurable language bias or by its constraint specification language. However, this solution does not solve the problem at all. Indeed, the constraints that may be defined in Warmr, by using its constraint specification language, are only syntax based, and they are not sufficient to handle semantic dependencies. For this and other limitations already described in [9, 8], in some cases Warmr system is not able to calculate frequent subsequences and it is difficult to correctly represent the specific sequence mining task.

In [11] are presented a logic language, SeqLog, for mining sequences of logical atoms, and the inductive mining system MineSeqLog, that combines principles of the level-wise search algorithm with the version space in order to find all patterns that satisfy a constraint by using an optimal refinement operator for SeqLog. SeqLog is a logic representational framework that adopts two operators to represent the sequences: one to indicate that an atom is the direct successor of another and the other to say that an atom occurs somewhere after another. Furthermore, based on this language, the notion of subsumption, entailment and a fix point semantic are given. However, with SeqLog one can represent unidimensional sequences only.

In [10] has been proposed an algorithm for selecting logical hidden Markov models from data. Hidden Markov models are one of the most popular methods for analyzing sequential data, but they can be exploited to handle sequence of flat/unstructured symbols. The proposed logical extension [10] overcomes such weakness by handling sequences of structured symbols by means of a probabilistic ILP framework. However, the mining of multi-dimensional sequence is not taken into account from these methods both logical and propositional.

5 Conclusions

In this paper we proposed a logical framework for mining multi-dimensional patterns in which many dimensions can be specified. What we can obtain are maximal frequent multi-dimensional patterns described in a first order language.

One of the most important characteristic of using logical framework for sequences is that we can incorporate additional information by using a background knowledge, and that any relation between atoms can be expressed or learned.

The result is a dedicated system in which are incorporated specific language bias for multi-dimensional data in order to rise a faster execution and a smaller search space. Preliminary experimental results prove the validity of the proposed approach.

Acknowledgements This work is partially founded by the Italian COFIN project "Learning Hierarchical, Abstract Models from Temporal/Spatial Data". The authors would like to thank Jan Ramon for giving useful suggestions on setting the system Warmr.

References

1. R. Agrawal, H. Manilla, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.
2. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ilp. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 60–77. Springer, 2000.
3. H. Blockeel, J. Fürnkranz, A. Prskawetz, and F. Billari. Detectin temporal change in event sequences: an application to demographic data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 29–41. Springer, 2001.
4. S. de Amo and D.A. Furtado. First-order temporal pattern mining with regular expression constraints. *Data & Knowledge Engineering*, 62(3):401–420, 2007.
5. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
6. M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 2(223):120–123, October 1970.
7. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD Int. Conf.Management of Data (SIGMOD'00)*, pages 1–12, 2000.
8. N. Jacobs. *Relational Sequence Learning and User Modelling*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, October 2004.
9. N. Jacobs and H. Blockeel. From shell logs to shell scripts. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157, pages 80–90. Springer, 2001.
10. K. Kersting and T. Raiko. 'Say EM' for selecting probabilistic models for logical sequences. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI05)*, pages 300–307. AUAI Press, 2005.
11. S.D. Lee and L. De Raedt. Constraint based mining of first order sequences in SeqLog. In R. Meo, P.L. Lanzi, and M. Klemettinen, editors, *Database Support for Data Mining Applications*, volume 2682 of *LNCS*, pages 155–176. Springer, 2004.
12. D. Malerba and F.A. Lisi. Discovering associations between spatial objects: An ilp application. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *LNCS*, pages 156–166. Springer, 2001.
13. C. Masson and F. Jacquenet. Mining frequent logical sequences with SPIRIT-LoG. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 166–181. Sringer Verlag, 2003.
14. J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.

15. S. Moyle and S. Muggleton. Learning programs in the event calculus. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 205–212. Springer, 1997.
16. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
17. P.J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*, pages 18–25, 2002.
18. H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88, New York, NY, USA, 2001. ACM Press.
19. L. Popelínsky. Knowledge discovery in spatial data by means of ILP. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 185–193. Springer, 1998.
20. J. Rodríguez, C. Alonso, and H. Böstrom. Learning first order logic time series classifiers. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Workshop on Inductive Logic Programming*, pages 260–275. Springer, 2000.
21. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.
22. M.J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal: Special issue on Unsupervised Learning*, 42(1/2):31–60, 2001.

ILP: Compute Once, Reuse Often *

Nuno A. Fonseca[1], Ricardo Rocha[2], Rui Camacho[3], and Vítor Santos Costa[2]

[1] Instituto de Biologia Molecular e Celular (IBMC), Universidade do Porto, Portugal
nf@ibmc.up.pt
[2] DCC-FC, Universidade do Porto, Portugal
{ricroc,vsc}@ncc.up.pt
[3] Faculdade de Engenharia & LIAAD, Universidade do Porto, Portugal
rcamacho@fe.up.pt

Abstract. Inductive Logic Programming (ILP) is a powerful and well-developed abstraction for multi-relational data mining techniques. However, ILP systems are not particularly fast, most of their execution time is spent evaluating the hypotheses they construct. The evaluation time needed to assess the quality of each hypothesis depends mainly on the number of examples and the theorem proving effort required to determine if an example is entailed by the hypothesis. We propose a technique that reduces the theorem proving effort to a bare minimum and stores valuable information to compute the number of examples entailed by each hypothesis (using a tree data structure). The information is computed only once (pre-compiled) per example. Evaluation of hypotheses requires only basic and efficient operations on *trees*. This proposal avoids re-computation of hypothesis' value in theory-level search and cross-validation algorithms, whenever the same data set is used with different parameters. In an empirical evaluation the technique yielded considerable speedups.

Keywords: Mode Directed Inverse Entailment, Efficiency, Data Structures, Compilation

1 Introduction

Several multi-relational data mining approaches have been proposed, such as tree-mining, graph-mining, or cross-relational mining [13]. One powerful and well-developed abstraction for multi-relational data mining techniques is Inductive Logic Programming (ILP) [15,16]. ILP has been successfully applied to problems in several application domains [14]. Nevertheless, improvements in efficiency and scalability are necessary to successfully tackle applications that learn from large data-sets and/or generate large hypothesis spaces.

* This work has been partially supported by Myddas (POSC/EIA/59154/2004) and by funds from the *Programa Operacional "Ciência, Tecnologia, Inovação" (POCTI) e do Programa Operacional "Sociedade da Informação" (POSI) do Quadro Comunitário de Apoio III (2000-2006)*. Nuno A. Fonseca is funded by FCT grant SFRH/BPD/26737/2005.

Research in improving the efficiency of ILP systems has focused on reducing their sequential execution time, either by reducing the number of hypotheses generated (see, e.g., [18,5]), or by efficiently testing candidate hypotheses (see, e.g., [4,7,8]). Another line of research, recommended by Page [19] and pursued by several researchers is the parallelization of ILP systems[10].

It is well known that an ILP system generates many candidate hypotheses which have many similarities among them. Usually, these similarities tend to correspond to common prefixes among the hypotheses. Blockeel et al. [4] defined query-packs as a technique to exploit this pattern and improve the execution time of ILP systems. Inspired by their work, we focus on how to reduce the amount of theorem proving to a minimum. As a first step, in a previous work [6], we argued that in MDIE [17] based systems the ILP search process can be efficiently coded by considering the set of all clauses that can be generated from the bottom clause. This led to an algorithm where a *prefix-tree* is used to represent all clauses that can be generated during the search (much in the way of query packs). But, instead of actually *evaluating* these clauses, we estimate coverage by counting the number of bottom-clauses that generated those clauses. Initial results showed that such an approach can indeed improve performance over standard ILP search.

The above work uses a single tree to represent the whole search space. In this work, we go one step further by proposing a novel two step approach to MDIE-based ILP where:

1. A pre-compilation step defines the search space by generating a set of clauses per example (where a tree can be used to encode the set of clauses).
2. A search step implements a search using algorithms constructed from an algebra of set operations implemented over these sets of clauses.

Our original motivation was the observation that the same set of clauses is generated from the same example at different computation steps (i.e., at different steps of theory construction or when performing cross-validation). Hence, computing the set of clauses for each example a single time, before execution, could significantly improve performance. Indeed, experimental results do show a large reduction in execution time. Moreover, we believe that our approach also provides a novel, and very modular, framework for ILP algorithm design, where the search can be easily encoded using set operations.

The remainder of the paper is organised as follows. In Section 2 we provide a brief introduction to ILP and MDIE. Section 3 introduces the reader to the rationale of seeing the examples as set of clauses and in Section 4 we present a first algorithm that exploits this idea. Next, in Section 5, we describe the proposed two step algorithm ($T-once$ MDIE). In Section 6 some implementation details are discussed. In Section 7 we present an empirical evaluation of the impact in execution time and accuracy of our algorithm. Finally, in Section 8 we discuss our work and draw conclusions.

2 Background

The predictive ILP problem can be defined as follows. Let E^+ be the set of positive examples, E^- the set of negative examples, $E = E^+ \cup E^-$, and B the prior knowledge (*background knowledge*). In general, B and E can be arbitrary logic programs. The aim of an ILP system is to find a hypothesis (also referred to as a theory) H, in the form of a logic program, such that $B \wedge E^- \wedge H \not\models \Box$ (Consistency) and $B \wedge H \models E^+$ (Completeness), assuming that $B \not\models E^+$ and $B \wedge E^- \not\models \Box$.

Mode-Directed Inverse Entailment (MDIE) [17] uses *inverse entailment* together with *mode restrictions* as the basis to perform induction. The key idea in MDIE is to find all literals that could be used in hypotheses that explain the example. This is achieved through the construction of the *bottom-clause*, that can be considered as the set of all such literals.

Construction of the bottom-clause $\perp_e$ often proceeds as a standard fixed-point calculation algorithm. Starting from the example e, and using the mode declarations, we scan the mode language for all possible clauses of the form $e \leftarrow l_1$. We collect all answers for l_1 as a set $\mathcal{L}_1 = \cup_i l_{1i} \setminus \mathcal{L}_0$, where $\mathcal{L}_0 = \{e\}$. Next we generate the set $\mathcal{L}_2$ with all clauses of the form $l_1 \leftarrow l_2$, where $l_1 \in \mathcal{L}_1$. We keep repeating the process until reaching a fixed point (which may be the whole data-base) or reaching some user-defined constraint. Therefore, the bottom clause $\perp_e$ can be seen as $\cup_j \mathcal{L}_j$. Most ILP systems use the bottom clause in order to bound (anchor) the search space lattice. Therefore, most applications try to have relatively small bottom-clauses, as otherwise the search space is as big as if one just enumerates clauses.

3 Examples as Set of Clauses

MDIE-based systems use bottom-clauses to generate sets of clauses. Given a bottom-clause $\perp_e$, the refinement operator generates clauses from $\perp_e$ that will cover at least the example e. Let us call this set $\mathcal{S}$. The clauses in $\mathcal{S}$ share e, so we can say that e *forms* $\mathcal{S}$. Note that, in general, $\mathcal{S}$ will be arbitrarily large, and we will need to impose some further restrictions, such as clause length restrictions. Moreover, note that even if complete, $\mathcal{S}$ does not correspond to all clauses that cover e. Indeed, it is well known that the bottom-clause is not complete: we can generate clauses that cover an example e which cannot be refined from $\perp_e$ [20].

Still, it is interesting to try to understand the meaning of $\mathcal{S}$. An important question in this regard is: if a clause c generated for example e covers example x, will c or, to be more precise, a variant of c, be in x's bottom clause, $\perp_x$? We would expect this to be true for ground clauses. Indeed, if this was not the case there must be at least a ground clause $h \leftarrow g_1, \ldots, g_{i-1}, g_i$ not refined from $\perp_x$, such that $h \leftarrow g_1, \ldots, g_{i-1}$ can be refined from $\perp_x$. Moreover, g_i must be in $\perp_e$ but not in $\perp_x$. On the other hand, if g_i was in $h \leftarrow g_1, \ldots, g_{i-1}, g_i$ it can be reached from $h, g_1, \ldots, g_{i-1}$, so it must also be in $\perp_x$.

Consider, for example, the following bottom-clause for an example e:

$$\bot_e = l(A) \leftarrow h\_c(A,B), h\_c(A,C), d(B), o\_c(B), f(C).$$

and the following clause c:

$$c = l(A) \leftarrow h\_c(A,B), h\_c(A,C), d(C), o\_c(B).$$

Careful examination shows that $\bot_e$ is entailed by clause c. On the other hand, the closest clause c' that can be generated from the bottom-clause is:

$$c' = l(A) \leftarrow h\_c(A,B), h\_c(A,C), d(B), o\_c(B).$$

Although $c' = c\theta$, c' is a more *specific* version of the original clause, it is not a variant. In this case, we cannot find a variant, even though the example indeed covers the clause.

This suggests the following approach: given an example e construct the corresponding bottom clause $\bot_e$ and generate a set $\mathcal{S}$ with all legal clauses c such that c θ-subsumes $\bot_e$. Next, given a set of examples $\{e_1^+, e_2^+, \ldots, e_n^+, e_1^-, e_2^-, \ldots, e_m^-\}$ construct the corresponding sets of clauses $\{\mathcal{S}_1^+, \mathcal{S}_2^+, \ldots, \mathcal{S}_n^+, \mathcal{S}_1^-, \mathcal{S}_2^-, \ldots, \mathcal{S}_m^-\}$: finding the best clauses should be just a question of searching for clauses that appear in most $\mathcal{S}_i^+$ and not in $\mathcal{S}_i^-$. More precisely, if we allow no noise, then we would like to find the clause with the largest coverage from $\cup_i \mathcal{S}_i^+ \setminus \cup_j \mathcal{S}_j^-$.

We are not interested in the examples, but in the set of all clauses of interest, $\mathcal{S}$ (which would to a first approximation be close to $\cup_i \mathcal{S}_i^+$). Now, this set may grow quickly, and therefore needs a compact and fast representation. It makes sense to represent sets of clauses by structures optimised for quick access and sharing, such as the *tries* discussed in Section 6.

4 T-MDIE

Assuming that the above representation works, one approach to *estimate* the coverage of all clauses is: walk over all examples and generate all clauses subsuming the bottom-clause such that for each clause c generated from an example $e \in E$:

- If $c \in \mathcal{S}$, somehow state that c covers e.
- If $c \notin \mathcal{S}$, add c to $\mathcal{S}$ and state that c covers e.

This basic algorithm can be optimised if we visit positive examples first, and assume we do not care about clauses that only cover negative examples:

- If the example $e \in E^+$ and $c \in \mathcal{S}$, state that c covers one more positive example.
- If the example $e \in E^+$ and $c \notin \mathcal{S}$, add c to $\mathcal{S}$ and state that c covers one positive example.
- If the example $e \in E^-$ and $c \in \mathcal{S}$, state that c covers one more negative example.

– If the example $e \in E^-$ and $c \notin \mathcal{S}$, do nothing.

We therefore need to define an abstract set that we call *decorated set* $\mathcal{S}$ with all clauses and their coverage. A *decorated set* $\mathcal{S}$ is a set whose elements are clauses, and attached to each element are several counters (one counter for each class of the learning problem). With this abstraction we can easily implement any theory construction algorithm as shown in Figure 1. The main difference with systems like Progol or Aleph concerns the inner procedure $learn\_T\_MDIE()$. We next describe how clauses are being learned in the *T-MDIE* approach [6].

$generalise\_T\_MDIE(B, E^+, E^-, C)$:
Given: background knowledge B, finite training set $E = E^+ \cup E^-$, constraints C.
Return: a hypothesis H that explains E and satisfies C.

1. $H = \emptyset$
2. **while** $E^+ \neq \emptyset$ **do**
3. $\quad h = learn\_T\_MDIE(B, E^+, E^-, C)$
4. $\quad E^+ = E^+ \setminus covered(h)$
5. $\quad H = H \cup h$
6. $\quad B = B \cup h$
7. **endwhile**
8. **return** H

Fig. 1. The greedy cover algorithm of a MDIE system implementation.

The *T-MDIE* algorithm has two basic stages (see Figure 2). First a decorated set $\mathcal{S}$ is constructed (lines 1 to 9) and then the best clause (according to some metric) is found by inspection of the set (line 10). The decorated set $\mathcal{S}$ is constructed as described above. First, all positive examples are processed and then a pruning procedure, $prune()$, is invoked to remove useless clauses from $\mathcal{S}$ (e.g., clauses with positive coverage lower than some predefined minimum number of positive examples). Next, all negative examples are also processed and then the set is pruned again. While processing the negative examples, the negative counters of the clauses in $\mathcal{S}$ are updated whenever a negative example generates a matching clause. This means that the clauses generated from the negative examples that are not in $\mathcal{S}$ are discarded.

$learn\_T\_MDIE(B, E^+, E^-, C)$:
Given: background knowledge B, finite training set $E = E^+ \cup E^-$, constraints C.
Return: the *best* hypothesis that explains some of the E^+ and satisfies C.

1. $\mathcal{S} = \emptyset$
2. **foreach** $e \in E^+$ **do**
3. $\quad fillSet(\mathcal{S}, B, e, C)$
4. **endforeach**
5. $\mathcal{S} = prune(\mathcal{S}, C)$
6. **foreach** $e \in E^-$ **do**
7. $\quad fillSet(\mathcal{S}, B, e, C)$
8. **endforeach**
9. $\mathcal{S} = prune(\mathcal{S}, C)$
10. **return** $bestClauseInTree(\mathcal{S}, C)$

Fig. 2. The learning algorithm of $T - MDIE$.

The set $\mathcal{S}$ is filled in three main steps (see Figure 3): i) for each example we generate a bottom clause (line 2); ii) using the bottom clause we generate all valid clauses[4] (line 4), normalise them (line 5), and insert them in the set (line 6). Normalisation orders the literals according to the Prolog "@ <" order relation. We generate all renaming of existential variables to check if a variant already exists in the tree, therefore guaranteeing a unique representation for each clause. The $insertUpdateInSet()$ procedure works as follows. If the example class is positive the clause is inserted into $\mathcal{S}$ and the positive counter updated. If the class is negative, only the negative counter of the clause is updated (the clause is not added to $\mathcal{S}$, only the coverage is updated).

$fillSet(\mathcal{S}, B, e, C)$:
Given: decorated set $\mathcal{S}$, background knowledge B, example e, constraints C.

1. $class = getExampleClass(e)$
2. $bottom = saturate(e, B, C)$
3. **do**
4. $\quad clause = findNewValidClause(bottom, C)$
5. $\quad clause = normalise(clause)$
6. $\quad insertUpdateInSet(clause, \mathcal{S}, class)$
7. **while** $clause\ != \emptyset$

Fig. 3. From an example to a set of clauses.

The algorithm is shown to be complete when compared to the traditional approach of computing the coverage (PROLOG resolution). Therefore, the coverage calculated for a clause by the algorithm should be interpreted as an estimate since it may not be the exact (*correct*) value.

5 T-once MDIE

The construction of the set $\mathcal{S}$ requires saturating all the examples (both positives and negatives). Often, one may want to perform repeated runs on the same examples. For instance, one may want to experiment with different parameters, or one may be performing cross-validation. Next, we show how repeated saturation and clause generation can be avoided by *decoupling* the generation of $\mathcal{S}$ from its usage. The process is divided into two steps: a compilation step, where a $\mathcal{S}_e$ is generated for each example e and stored on disk. The stored $\mathcal{S}_e$ are loaded at runtime, therefore avoiding the saturation and generation of clauses.

Our algorithm works as follows. At compilation time, we construct a *set of clauses per example*; more precisely, we represent the saturated clause for each example e as a separate $\mathcal{S}_e$ as depicted in the algorithm presented in Figure 4. At learning time, we obtain clause coverage information through an algebra of *basic operations*, such as *union* or *subtraction*, on decorated sets. As an example, search for the best clause can be described as a search in the decorated set obtained from the *union* of all $\mathcal{S}_e$. Next, we show in more detail how such approach can

[4] Clauses satisfying the language and bias constraints.

be used to implement greedy coverage in a MDIE-based ILP system. It should be clear that similar operations can be used to implement other ILP algorithms.

$compileAnswerSet(B, E, C)$:
Given: background knowledge B, finite training set $E = E^+ \cup E^-$, constraints C.

1. **foreach** $e \in E$ **do**
2. $class = getExampleClass(e)$
3. $bottom = saturate(e, B, C)$
4. $\mathcal{S}_e = \emptyset$
5. **do**
6. $clause = findNewValidClause(bottom, C)$
7. $clause = normalise(clause)$
8. $\mathcal{S}_e = insertInSet(clause, \mathcal{S}, class)$
9. **while** $clause\ != \emptyset$
10. $saveSet2File(\mathcal{S}_e, e, C)$
11. **endforeach**

Fig. 4. Compilation of examples procedure.

Figure 5 shows one approach to implement a MDIE-based algorithm (such as the default algorithms in Progol [17] and Aleph [1]) using the decorated sets. Like the algorithm presented in the previous section, the $learn\_T - once\_MDIE()$ algorithm has two main stages: first, it generates a $\mathcal{S}$ by loading the compiled decorated sets and merging them using a *joinAdd()* procedure; then the best clause is recursively selected using greedy cover removal.

$learn\_T - once\_MDIE(E, C)$:
Given: finite training set $E = E^+ \cup E^-$, constraints C.
Return: a hypothesis H that explains E and satisfies C.

1. $\mathcal{S} = \emptyset$
2. **foreach** $e \in E^+$ **do**
3. $\mathcal{S}_e = loadSetFromFile(e, C)$
4. $\mathcal{S} = joinAdd(\mathcal{S}, \mathcal{S}_e)$
5. **endforeach**
6. $\mathcal{S} = prune(\mathcal{S}, C)$
8. $H = \emptyset$
9. **while** $E^+ \neq \emptyset$ **do**
10. $h = bestClauseInTree(\mathcal{S}, C)$
11. $E^+ = E^+ \setminus covered(h)$
12. $H = H \cup h$
13. $\mathcal{S} = subtract(\mathcal{S}, \{\mathcal{S}_e |\ e \in covered(h)\ and\ e \in E^+\})$
14. **endwhile**
15. **return** H

Fig. 5. The greedy cover algorithm of a MDIE system implementation with pre-compilation.

6 Implementation Issues

Our algorithms depend on the ability to implement efficiently operations such as *union* and *subtraction* of sets. Furthermore, we need a data structure to store the

decorated sets (clauses and respective coverages). To do so efficiently, we used tries [12]. A trie is a tree structure where each different path through the trie data units, the *trie nodes*, corresponds to a term (clause). An essential property of the trie data structure is that common prefixes are represented only once. This naturally applies to ILP since the hypothesis space is structured as a lattice and hypotheses close to one another in the lattice have common prefixes (literals).

Using Tries to Represent Hypotheses In order to maximise the number of common trie nodes when storing clauses in a trie, we used Prolog lists to represent the clauses. A clause of the form $Head :- Body_1, ..., Body_n$ is stored in the trie structure as an unique path corresponding to the list $[Head, Body_1, ..., Body_n]$. Such a path always starts at the root node in the trie, follows a sequence of trie nodes and terminates at a leaf data structure, the *ilp frame* data structure, that we used to extend the original trie structure to store associated information with the clause, namely information concerning the number of positive and negative examples covered by the clause. Figure 6 presents an example of a trie storing three clauses.

An important point when using tries to represent terms is the treatment of variables. We follow the formalism proposed by Bachmair *et al.* [2], where each variable in a term is represented as a distinct constant. Formally, this corresponds to a function, $numbervar()$, from the set of variables in a term t to the sequence of constants $VAR_0, ..., VAR_N$, such that $numbervar(X) < numbervar(Y)$ if X is encountered before Y in the left-to-right traversal of t. For example, in the term $[eastbound(T), has\_car(T, C), long(C)]$, $numbervar(T)$ and $numbervar(C)$ are respectively VAR_0 and VAR_1.

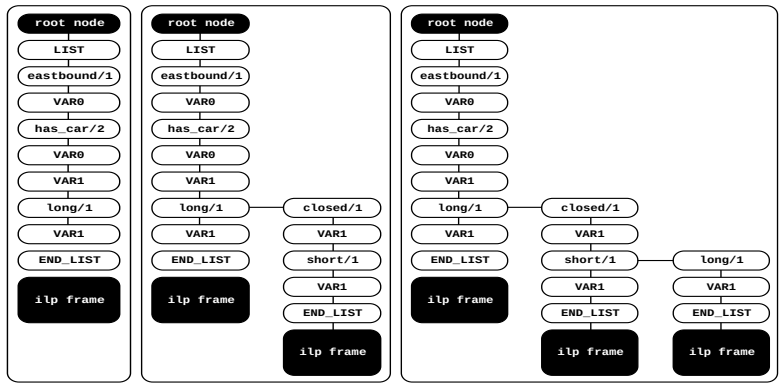


Fig. 6. Using tries to represent:
(a) *C = eastbound(T) :- has_car(T,C), long(C).*
(b) *C and D = eastbound(T) :- has_car(T,C), closed(C), short(C).*
(c) *C, D and E = eastbound(T) :- has_car(T,C), closed(C), long(C).*

Basic Trie Operations In the proposed *T-once* algorithm, an decorated set is constructed once and for each example. Therefore, since tries are used to represent the decorated sets we need to be able to perform some basic trie operations such as the union and subtraction of tries. The *trie-joinAdd()* and *trie-subtract()* procedures implement these operations. Given two tries, A and B, the *trie-joinAdd(A,B)* procedure returns a trie R representing the union of both tries, that is, if a term $t \in A$ or $t \in B$ then $t \in R$ and $ilp\_frame(t_R) = ilp\_frame(t_A) + ilp\_frame(t_B)$, where $ilp\_frame(t)$ represents the information concerning the number of positive and negative examples covered by t.

The *trie-subtract(A,B)* procedure returns a trie R equivalent to A but with the information concerning the number of positive and negative examples covered by the terms in B subtracted from the terms in A. More formally, if a term $t \in A$ then $t \in R$ and $ilp\_frame(t_R) = ilp\_frame(t_A) - ilp\_frame(t_B)$. Terms represented in B but not in A are ignored.

Searching through a chain of sibling trie nodes that represent alternative paths is done sequentially. When the chain becomes larger than a threshold value (8 in our implementation), we dynamically index the nodes through a hash table to provide direct node access and therefore optimise the search. Further hash collisions are reduced by dynamically expanding the hash tables. Hence, if the total number of trie nodes in tries A and B is respectively N_A and N_B, then the time complexity of the *trie-joinAdd(A,B)* and *trie-subtract(A,B)* procedures is $O(N_A + N_B)$.

7 Experiments and Results

The goal of the experiments was to evaluate the impact of the proposed approach on the execution time and quality of the models when dealing with real application problems. We implemented the two algorithms in the April ILP system [11]. For each data set the system was executed with the following configurations: standard MDIE implementation using a deterministic top-down breadth-first search (DTD-BF), $T - MDIE$, and $T - once$.

7.1 Experimental Settings

The experiments were performed on an AMD Athlon(tm) MP 2000+ dual-processor PC with 2 GB of memory, running Linux (kernel 2.6.12) Fedora. The data sets used were downloaded from the Machine Learning repositories at the Universities of Oxford[5] and York[6]. Table 1 characterises the data sets in terms of number of positive and negative examples as well as background knowledge size (number of relations used). The total number of examples ranges from 205 in the Mutagenesis data set up to 1762 in the Pyrimidines data set.

[5] `http://www.comlab.ox.ac.uk/oucl/groups/machlearn/`
[6] `http://www.cs.york.ac.uk/mlg/index.html`

Table 1. Data sets. $| E^+ |$ is the number of positive examples, $| E^- |$ is the number of positive examples, and $| B |$ is the number of relations in the background knowledge.

Data set	$\| E^+ \|$	$\| E^- \|$	$\| B \|$
Carcinogenesis	202	174	44
Mutagenesis	136	69	21
Pyrimidines	881	881	244

The search was constrained to clauses with 3 literals (maximum) in the body. The clause length was constrained due to the time taken by the DTD-BF algorithm. In practice we tested $T - once$ algorithm on some datasets with a clause length up to 5 literals in the body. We performed a 10-fold cross validation to evaluate the training time and accuracy.

7.2 Results and Discussion

Naturally, $T - once$ MDIE execution time requires some time to compile the examples. Table 2 presents the compilation (pre-processing) time, in seconds, taken for each data set, the average number of clauses compiled (refined), and the average file size of each compiled example (in kbytes). Cleary, the compilation is not a particularly fast process and further improvements should be made. Nevertheless, compilation is performed only *once*, as long as saturation-related settings are not changed or the clause length used is not increased. Therefore, in subsequent runs where other parameters (e.g., as noise) are changed there is no need to recompile the examples.

Table 2. Compilation (pre-processing) time, average number of clauses compiled by example (in thousands), and average file size (in kbytes) of each compiled example.

Data set	Time (sec)	Clauses	Size
Carcinogenesis	2,840	19,351 k	920 kb
Mutagenesis	6,054	12,659 k	301 kb
Pyrimidines	1,451	2,079 k	33 kb

Table 3 compares the execution times of $DTD-BF$, $T-MDIE$, and $T-once$ algorithms. The values presented are the average of a 10-fold cross validation and the sum of the execution times (within brackets). The results show that once the examples are compiled, $T-once$ is several times faster than $T-MDIE$ and $DTD-BF$ in all datasets. However, if we take into account the compilation time then the best approach, for the 10-fold runs, is clearly $T-MDIE$. Naturally, the gains of using $T - once$ increase with the number of runs performed. Therefore, it is well suited for cross-validation and for performing parameter tuning.

Finally, Table 4 presents the average accuracy for the two approaches. It shows that in spite of the coverage computed to be an estimate, the improvements in performance are not obtained at a cost of the quality of the models generated.

Table 3. Average execution time (in seconds) and cross-validation total execution time (within brackets). The total execution time of $T - once$ algorithm includes the compilation time.

Data Set	$DTD - BF$	$T - MDIE$	$T - once$
Carcinogenesis	617 (*6,170*)	59 (*590*)	14 (*2,980*)
Mutagenesis	2,487 (*24,870*)	43 (*430*)	34 (*6,394*)
Pyrimidines	570 (*5,700*)	89 (*890*)	20 (*1,651*)

Table 4. Average accuracy (standard deviation within brackets)

Data set	$DTD - BF$	$T - MDIE$	$Diff$
Carcinogenesis	50 (3)	58 (9)	*+8*
Mutagenesis	75 (11)	74 (9)	*-1*
Pyrimidines	83 (3)	80 (1)	*-3*

8 Conclusions

We have presented a novel approach to the execution of MDIE algorithms. Our approach proceeds in two steps. In the first step we compile each example as a set of clauses. In the second step we implement ILP search as a set of operations over these sets of clauses. Since such operations can be implemented very efficiently, our approach can generate major speedups over traditional ILP execution.

The reuse of the initial computation of the pre-compilation step pays-off whenever there is a large amount of repetition in clause evaluation. That happens when the induced theory has several clauses. In this case, after each iteration the covered examples are *removed* and we only need to perform subtraction operations between the sets of clauses, an operation that can be efficiently implemented using tries. The technique also pays-off when using cross-validation.

A further advantage of the approach is that it can be easily parallelisable, as the first step runs independently for every example. Moreover, we believe that our approach is a step forward in facilitating experimentation with different parameters, and namely in using internal cross-validation for parameter selection in ILP. On the other hand, the approach applies to MDIE-based algorithms only, and it needs further investigation when exploring longer clauses or in data sets with large numbers of examples (some techniques from [3] may help in that direction).

Last, an interesting insight from our approach is that we can abstract the ILP search procedure as a process of *tree-mining* over the trees representing individual examples. We believe that this suggests new and exciting directions for future research in this area.

References

1. Aleph. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.
2. L. Bachmair, T. Chen, and I. V. Ramakrishnan. Associative-Commutative Discrimination Nets. In *Proceedings of the 4th International Joint Conference on*

Theory and Practice of Software Development, number 668 in LNCS, pages 61–74, 1993. Springer-Verlag.
3. H. Blockeel and L. De Raedt and N. Jacobs and B. Demoen, *Scaling Up Inductive Logic Programming by Learning from Interpretations*, Data Mining and Knowledge Discovery, vol. 3, N. 1, pp 59-93, 1999.
4. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of Inductive Logic Programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.
5. R. Camacho. Improving the efficiency of ilp systems using an incremental language level search. In *Annual Machine Learning Conference of Belgium and the Netherlands*, 2002.
6. R. Camacho, N. A. Fonseca, R. Rocha and V. S. Costa. *ILP :- Just Trie It.* 17th International Conference on Inductive Logic Programming, 2007.
7. V. S. Costa, A. Srinivasan, and R. Camacho. A note on two simple transformations for improving the efficiency of an ILP system. *LNCS*, 1866, 2000.
8. V. S. Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W. V. Laer. Query transformations for improving the efficiency of ILP systems. *Journal of Machine Learning Research*, 4:465–491, 2003.
9. N. A. Fonseca and R. Rocha and R. Camacho and F. Silva *Efficient Data Structures for Inductive Logic Programming*, Proceedings of the 13th International Conference on Inductive Logic Programming LNAI vol 2835, pp 130–145, 2003.
10. N. A. Fonseca and F. Silva and R. Camacho, *Strategies to Parallelize ILP Systems*, Proceedings of the 15th International Conference on Inductive Logic Programming (ILP 2005), LNAI, vol 3625, pp 136–153, 2005.
11. N. A. Fonseca, F. Silva, and R. Camacho. April - An Inductive Logic Programming System. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA06)*, volume 4160 of *LNAI*, pages 481–484, 2006. Springer-Verlag.
12. E. Fredkin. Trie Memory. *Communications of the ACM*, 3:490–499, 1962.
13. J. Han and M. Kimber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
14. Ilp applications. http://www.cs.bris.ac.uk/ ILPnet2/Applications/.
15. S. Muggleton. Inductive logic programming. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 43–62. Ohmsma, Tokyo, Japan, 1990.
16. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–317, 1991.
17. S. Muggleton, *Inverse Entailment and Progol*, New Generation Computing, Special issue on Inductive Logic Programming. 245-286, vol 13, N. 3-4, 1995.
18. 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*, pages 82–103. IOS Press, 1996.
19. D. Page. ILP: Just do it. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 3–18. Springer-Verlag, 2000.
20. A. Yamamoto *W*hich Hypotheses Can Be Found with Inverse Entailment? ILP '97: Proceedings of the 7th International Workshop on Inductive Logic Programming, pp 296–308, 1997.

Mining Imbalanced Classes in Multirelational Classification

Hongyu Guo and Herna L. Viktor

School of Information Technology& Engineering,
University of Ottawa, Canada
{hguo028,hlviktor}@site.uottawa.ca

Abstract. Multirelational classification algorithms search for patterns across multiple interlinked tables (relations) in a relational database. This type of method searches for relevant features both from a target relation (in which each tuple is associated with a class label) and relations related to the target, in order to better classify tuples in the target relation. Unfortunately, most of these methods implicitly assume that the classes in the target relation are equally represented. They thus tend to produce poor predictive performance over the underrepresented class in the data. This paper presents a novel strategy to deal with imbalanced multirelational data where the number of examples of one class in the target relation is much higher than the others. The algorithm learns from multiple views of a relational database and then combines the knowledge acquired by the view learners in order to find a better quality model for the skewed classes. Experiments performed on six real-world data sets show that the proposed method achieves promising results when compared with other popular relational data mining algorithms, in terms of the ROC curve and AUC value obtained. In particular, our method outperforms the others for very highly imbalanced data sets.

1 Introduction

There is a rich tradition of imbalanced class learning applied to single table (flat file) data sets, where the number of examples of one class (majority) is much higher than the others (minority classes) [19]. Data mining algorithms may be biased towards the majority class (negative examples), thus producing poor predictive accuracy over the minority classes (positive examples). Typical approaches to dealing with imbalanced data sets include under-sampling negative examples, over-sampling positive instances, penalizing misclassification of minority classes (cost-sensitive learning), weighting examples in an effort to bias the learning toward the minority class, applying boosting algorithms, and creating synthetic data to balance the examples of the classes [19].

In contrast to the rich tradition in the conventional machine learning community, little attention has been paid to the imbalanced class problem in the multirelational data mining research field. Over the past few years, methods such as FOIL [16], CrossMine [21], TILDE [2], RelAggs [13], and RollUp [12],

amongst others, have been introduced to learn from multirelational data. Most of these methods implicitly assume that the classes to be learned are equally represented. Unfortunately, target tuples in many practical database applications, such as credit card fraud detection and disease diagnosis, are highly imbalanced. Often, correctly classifying the minority examples is of importance. To our best knowledge, only one such approach was proposed by Sen and Getoor in [17]. They introduce the so-called "structured" cost functions to address the cost-sensitive learning problem. That is, misclassification costs assignment not only depends on the cost of individual example, but also the correction of related instances. Their method requires that relational classifiers be able to capture the correlations of the examples. The goal of their approach is to minimize the expected cost of misclassification of the classifiers. We here present a new strategy to learn from imbalanced target tuples in relational databases. In contrast to the method proposed by Sen and Getoor, our strategy enables traditional single-table classifiers to learn from skew class data. In addition, the goal of our algorithm is to produce a classifier which can better predict both the minority and majority classes in a relational database.

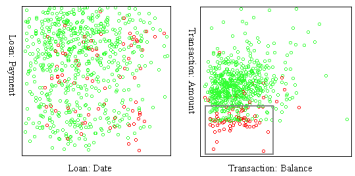


Fig. 1. Different dimensional projections of feature space

Our approach is based on the following three observations. First, often a relational database is designed by domain experts using an Entity Relationship (ER) model, where multiple relations are connected via entity-relationship links [8]. Each relation usually groups a set of features which come from a certain type of observation of the relational domain. Second, different feature subsets in a multirelational domain may have different awareness of the skewed classes to be learned. As an example, Figure 1 visualizes the data from the *Loan* and *Transaction* relations of a banking database as presented in the PKDD 99 challenge [1]. Figure 1 shows the distribution of the data, when considered in terms of the two different tables. Here, the minority class is depicted using red circles, and the majority class is shown in green. When one considers the class distribution from the perspective of the *Loan* table, using the *date* and *payment* attributes (left hand side of the table), it follows that there is no clear division. However, when utilizing the *Transactions* table, with attributes *amount* and *balance*, as shown in the right hand side of the Figure, most of the positive examples (red circles) were gathering at the left-bottom corner. In this case, a simple decision region indicated by the solid-line rectangle can have a good prediction against the positive examples. Third, research shows that in many cases, ensembles of classifiers can be effective in producing a classifier that is superior to any of the

individual classifiers, especially when the combined classifiers are diverse and accurate. Thus, classifiers built using features in different relations of a relational database can potentially be integrated to improve the classification of the underrepresented examples in the data.

Our proposed method learns from multiple views (feature sets) of a relational database. Each view learner initially learns from a separate feature set of the database. As a result, each learner possesses a different awareness of the skewed classes in the target relation. Subsequently, the view learners are combined using majority voting in order to construct a better quality model for the imbalanced classes. Our experiments performed on six real-world databases show that the proposed method achieved promising results when compared with popular relational algorithms, in terms of the ROC curve and AUC obtained. In particular, our method outperforms the others for very highly imbalanced data sets.

This paper is organized as follows. Section 2 introduces the background and related work. Next, in Section 3, we present our proposed strategy. This is followed, in Section 4, by our experimental results. Section 5 concludes the paper.

2 Background and Related Work

Measurement metrics: Imbalanced classification algorithms are often evaluated using the Receiver Operating Characteristics (ROC) curves and AUC values [5, 9, 14, 19]. This is due to the ROC curve's insensitivity to changes in the class distribution of the data set [7]. A ROC curve represents the trade-offs between the TP rate and FP rate over a range of decision thresholds. Although it can summarize a classifier's performance over a range, the ROC curve is a two-dimensional technique. The area under the ROC curve (AUC), therefore, is often calculated to reduce the measurements of an ROC curve to a single value. This value provides an average expected performance of a classifier.

Learning from multiple feature sets: Learning from multiple feature subsets has been investigated by many researchers [3, 4, 10, 14]. In our opinion, the most promising one is multi-view learning. Multi-view learning describes the problem of learning from multiple independent sets of features, i.e. views, of the presented data. Following the same line of thought, Lazarevic and Kumar proposed a strategy which uses bagging features to detect outliers [14]. Bryll et al. used attribute bagging to improve the classifier's accuracy [4].

Multirelational data mining: Multirelational classification use a relational database as input. A schema for a relational database $\Re$ describes a set of tables $\Re = \{T_i\}_1^n$ and a set of relationships between pairs of tables. A table T_i consists of a set of tuples, a primary key (denoted by $T.{}_{key}$), a set of foreign keys (in this paper, we refer to primary key and foreign keys as *key attributes*); the other attributes are *descriptive attributes*. Foreign key attributes[1] link to primary keys of other tables: this link specifies a *join* between two tables.

In a multirelational classification setting, there is a database $\Re$, which consists of a target table T_{target} and a set of background relations $\{T_i\}_1^n$. In addition, the

[1] For simplicity, we only consider key attribute as a single attribute here.

target relation T_{target} has a target variable Y. That is, each tuple in this table (target tuple) is associated with a class label which belongs to Y. Typically, the relational classification task is to find a function $F(x)$ which maps each target tuple x to the category Y:

$$Y = F(x, T_{1\cdots n}, T_{target}), x \in T_{target}$$

3 The Multiple View Approach

3.1 The MVC framework

The so-called MVC-IM strategy presented here applies the MVC framework as proposed in [10]. This framework aims to enable propositional methods to explore relational domains directly. It consists of the following five sequential stages:

1) Information Propagation Stage: The *information propagation stage* constructs the training data sets for use by a number of multiple view learners, using a relational database as input. The *information propagation element* propagates the tuple IDs (identifiers for each tuple in the relation) and classes from the target relation to the background relations.

2) Aggregation Stage: The *aggregation stage* summarizes information embedded in multiple tuples and squeezes them into one tuple. In this stage, aggregation functions are applied to each background relation (to which the tuple ID and target concepts were propagated). Each new constructed background relation is then used as training data for a particular view learner.

3) View Learners Construction Stage: The *view learners construction stage* constructs various hypotheses on the target concept, based on the multiple training data sets given by the *aggregation stage*. Conventional data mining methods are used to learn the target concept from each view of the data separately. In this stage, a number of view learners, which differ from one another are constructed.

4) View Validation Stage: This step evaluates all view learners constructed in the *construction stage*. This processing is needed to ensure that they are sufficiently able to learn the target concept on their respective training sets. In the MVC framework, learners with training error greater than 50% are discarded.

5) View Combination Stage: The *view combination stage* is the last step of the MVC strategy. This step incorporates the trained view learners into a meta-learner to construct the final classification model. The meta-learner is called upon to produce a function to control how the multiple view learners work together, to achieve maximum classification accuracy.

We here extend the MVC framework in order to handle skewed classes in multirelational data. These extensions are discussed next.

3.2 Extensions for Learning Skew Classes

The main goal of the MVC–IM method is to learn a model which can better predict both the majority and minority classes in imbalanced multirelational data. Two major extensions, namely view validation and view combination, have been introduced in the MVC-IM strategy and are discussed next.

View Validation In order to produce a combined classifier which is superior to the individual view learners, we need to ensure that each inducted view learner has sufficient knowledge on the minority classes. Consider a special case of a MVC-IM learning setting, where the database forms three disjoint data sets V_1, V_2, and V_3. In this case, a tuple x in the target relation T_{target} is viewed as

$$\boldsymbol{x} = \langle x^1, x^2, x^3, y \rangle$$

where x^1, x^2, and x^3 are instances in the data set V_1, V_2, and V_3, respectively. The variable y denotes the class label. Three view learners f^1, f^2, and f^3 are constructed by training using data sets V_1, V_2, and V_3, respectively. In this way, we have decision functions

$$f(x^1, x^2, x^3, y) = f^1(x^1, y) \cup f^2(x^2, y) \cup f^3(x^3, y)$$

where $\cup$ denotes a model combination scheme. Often, a common way to combine the three learners is to let

$$F(X) = \texttt{mode}\{f^1, f^2, f^3\}$$

That is, x is assigned the class that receives the largest number of classifications (or votes). However, poor predictors may be transferred into a poorly performing final model. We, therefore, need to evaluate the validation (in terms of sufficient knowledge on both the majority and minority classes) of the view learners f^1, f^2, and f^3, in order to obtained a better combined classifier.

Unlike the view validation criterion error rate used by the MVC approach [10], the MVC-IM strategy uses the AUC value to evaluate the *quality* of the multiple view learners. It removes view learners with a low AUC value. The reason for this choice is as follows. In imbalanced class applications, predictive accuracy is inappropriate for evaluating learning methods. For example, a classifier which always predicts the majority class in a highly skewed class problem can get a very high resultant accuracy. In addition, accuracy metric is sensitive to the change of the class distribution in the data. AUC values, on the other hand, can provide an average expected performance of a classifier over a range of trade-offs between TP rate and FP rate, as well as over different decision thresholds. We, therefore, employ the AUC value to evaluate the performance of view learners of the MVC-IM algorithm.

In the MVC-IM strategy, view learners with AUC value less than 0.5 are discarded. In the ROC space, random guessing algorithms produce the diagonal line between the points (0,0) and (1,1), thus obtaining an AUC value of 0.5. That is, only view learners with average performance better than random guessing are used in the MVC-IM method.

View Combination In contrast to the meta-learning used in the MVC method, voting combination technique is employed in the MVC-IM strategy in order to obtain better knowledge on both the majority and minority classes.

Strategies for combining models have been investigated thoroughly [20]. The most popular are those such as Stacking, Boosting, and Bagging [20]. Stacking

employs a meta learner to learn which base classifiers are the reliable ones. The Boosting and Bagging methods, on the other hand, apply a voting principle.

Recall that the MVC method applies meta-learning to combine the multiple view learners. That is, a meta-learner is called upon to produce a function to control how the multiple view learners work together, in order to achieve maximum classification accuracy. In other words, the combination process of the MVC method aims to yield a better accuracy regardless of the predictions on the minority class. Thus, such a combination technique may end up with very poor predictive performance on the underrepresented examples in an imbalanced data set. Based on this observation, a meta-learning combination technique is inappropriate for learning the minority class in a skew class data set.

In contrast to the meta-learning method, another popular model combination technique is the *voting* principle, which is used in Boosting and Bagging. Boosting applies a *weighted voting* strategy. In this method, each individual model has a different weight affecting the final results. Often, the more confident individual learners have more impact on the final results. In a *majority voting* combination method, such as the one used in *Bagging*, each individual model outputs one score, e.g. a probability. This score is either assigned to one class label as a whole or divided into several class labels. Next, the class label with the largest score, for example, is taken as the final result of the combined model.

Algorithm 1 The MVC-IM Algorithm

Input: A DB= $\{T_{target}, T_1, T_2, \cdots, T_n\}$; Target tuples $\{\langle x_i, y_i\rangle\} \in T_{target}$, where $x_i \in X$, $y_i \in Y$ (X is some instance space and Y is a label set); View learner $\mathcal{L}$.
Output: Classification model F.

1: Propagate information from T_{target} to each $\{T_t\}_1^n$, forming $\{T'_t\}_1^n$;
2: Aggregate multiple tuples in $\{T'_t\}_1^n$, along with relation T_{target}, forming view set $\{\mathcal{V}^t\}_0^n$ from each relation in the DB;
3: Train each $\mathcal{L}$ separately with $\mathcal{V}^t \in \{\mathcal{V}^t\}_0^n$, forming hypothesis set $\{f^t\}_0^n$, where $f^t : X \rightarrow Y$, with AUC value of α^t;
4: Remove f^t with $\alpha^t \leq 0.5$, forming hypothesis set $\{f^t\}_0^{n'}$;
5: Output the final hypothesis:

$$F(X) = arg\max_c \sum_{t=0}^{n'} \hat{p_{ct}}$$

where $\hat{p_{ct}}$ is the probability estimate from the t^{th} hypothesis f^t for the c^{th} class.

In the MVC-IM algorithm, we employ a majority voting method rather than a weighted voting strategy. This is due to the following observation. View learners in the MVC-IM strategy learn only from their own feature set. In addition, each view learner may be trained by different subset of the target tuples. We, therefore, have no reason to strongly believe that certain resultant learners are "better" than the others. Furthermore, unlike the weighted voting strategy, a majority voting scheme can prevent over-confidence in voters.

The majority voting strategy in the MVC-IM algorithm, as described in Algorithm 1, works as follows. For each test example x, each view learner f^t first outputs a probability score for assigning each class c of the class set Y, i.e. $\hat{p_{ct}}$. Subsequently, these scores are summed up according to each class in the learning task. Finally, the class with the largest probability score is assigned to

the test instance as the final result. In other words, the final classifier decision of a test example x is based on the sums of the probability outputs

$$F(X) = arg\max_c \sum_{t=0}^{n'} \hat{p_{ct}}$$

That is, each value of X classify to the class that receives the largest score.

The above two sections present our extensions to the MVC framework. Now we simply describe the entire process of the MVC-IM strategy (Algorithm 1). As shown in Algorithm 1, the method, first, constructs multiple views from the relational database through information propagation and aggregation. Next, it trains multiple learners, each learning from a different view constructed. Subsequently, view learners with AUC value less than 0.5 are removed. Finally, the trained view learners are combined using majority voting.

Table 1. Summary of the data sets used

Data Set	#target tuples	#related relations	target class distribution	# tuples in task
MUT188	188	3	125:63 (34%)	15,218
F400AC	400	7	324:76 (19%)	75,982
F234AC	234	7	203:31 (13%)	75,816
F682AC	682	7	606:76 (11%)	76,264
EMCL_I	4,065	8	3705: 360 (8%)	194,214
EMCL_II	7,329	8	6969: 360 (4%)	197,478

4 Experiments

This section provides the results obtained for the MVC-IM algorithm on benchmark real-world databases. These results are presented in comparison, in terms of AUC and ROC achieved, with three other state-of-the-art multirelational data mining systems, namely TILDE, RelAggs, and CrossMine, along with a "baseline" "flattening" approach (denoted as SimFlat) and the original MVC method (as presented in [10]). Recall that, in contrast with the MVC-IM algorithm, the original MVC method uses the predictive accuracy for view validation prior to a meta-learning based view combination strategy. The SimFlat strategy uses the same aggregate functions and follows the same join chains used by the MVC-IM method to "flatten" relations into a flat file. The goal of the SimFlat strategy is to provide a universal flat file which consists of all features used by individual views in the MVC-IM strategy.

Six learning tasks derived from three standard real-world databases were used to evaluate our algorithm. The derived six learning tasks present varying degree of class distribution in the target relation. That is, the percentage of the positive examples in these six learning problems ranges from 4% to 34%. Thus, these learning tasks provide us a diverse test bed.

We implemented the MVC-IM algorithm using Weka [20]. Also, our experiments applied C4.5 decision trees [15] and Naive Bayes probabilistic classi-

fiers [11] for the learning in the MVC-IM, RelAggs, MVC, and SimFlat approaches. The C4.5 decision tree learner was used due to its de facto standard for empirical comparisons. In addition, Naive Bayes was chosen because of its sensitivity to the changes of the input attributes [6]. The default settings of these two learning methods were used. Each of these experiments produces ROC curves and AUC results using ten-fold cross validation. In order to generate the average ROC curve of the 10 folds, all test instances of the 10 folds were put together and sorted according to their probabilities of belonging to the classes; the AUC values were then calculated using the Wilcoxon-Mann-Whitney statistic [20]. The TILDE, RelAggs, and CrossMine methods were obtained from their authors.

4.1 Databases Used

Mutagenesis Database: Our first experiment (denoted as MUT188) was conducted against the Mutagenesis database [18]. In this data set, 34% of the target tuples belong to the minority class. A summary of the characteristics for the learning data set is given in Table 1.
Financial Database: Our second experiment was conducted against the Financial database [1]. This database provides us with three different learning problems, namely F234AC (target tuples contain only finished loans), F682AC (target tuples contain all finished and unfinished loans), and F400AC (database as prepared in [21]). There are 11%, 13%, and 19% of minority class instances in the target tables of the three learning tasks, respectively (presented in Table 1).
ECML98 Database: Our last experiment used the database for the ECML 1998 Sisyphus Workshop. There are two learning tasks derived from this database. In the first task (denoted as ECML_I), we removed tuples in class 2 (minority class) with id smaller than 14700; in the second task (denoted as ECML_II), we reversed the class of these removed tuples and then added them back to the target relation in order to obtain a highly imbalanced data set. There are 8% and 4% of minority class instances in the target tables of the two learning tasks, respectively. In this experiment, we used the new start schemes prepared in [13]. A summary of this learning data set is also presented in Table 1.

4.2 Experimental Results

Using C4.5 decision trees In the first experiment, C4.5 decision trees were used by the MVC-IM, RelAggs, and SimFlat approaches. Also, C4.5 decision trees were used as the view learners and meta learners of the MVC method. We present the ROCs and AUCs obtained for each of the six tasks in Figure 2 and Table 2, respectively. In Table 2, the best results for each data set were highlighted in *bold*.
AUC analysis: The AUC values listed in Table 2 show that for almost all datasets the MVC-IM method is able to improve over the RelAggs, TILDE, CrossMine, MVC, and SimFlat algorithms, when C4.5 decision trees were used. Exceptions were against the F400AC and F234AC data sets when compared with the CrossMine algorithm. In other words, our MVC-IM strategy can provide an improvement in classifying underrepresented classes of the data against almost all data

Table 2. AUCs of the RelAggs, SimFlat, TILDE, CrossMine, MVC, and MVC-IM approaches (using C4.5)

Data set	RelAggs	SimFlat	TILDE	CrossMine	MVC	MVC-IM
MUT188	0.876	0.888	0.807	0.912	0.874	**0.914**
F400AC	0.797	0.582	0.591	**0.824**	0.777	0.798
F234AC	0.722	0.48	0.591	**0.785**	0.687	0.775
F682AC	0.762	0.506	0.590	0.793	0.792	**0.805**
ECML_I	0.957	0.945	0.553	0.961	0.946	**0.990**
ECML_II	0.965	0.949	0.608	0.971	0.952	**0.993**

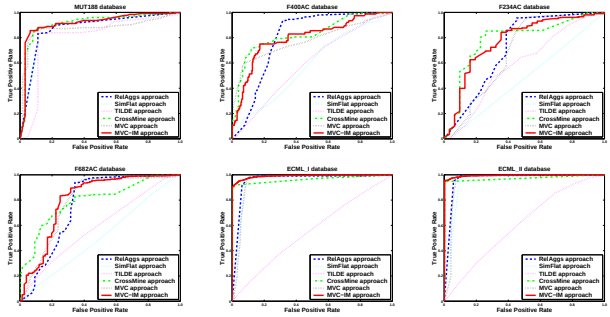


Fig. 2. ROC Curves for MUT188, F400AC, F234AC, F682AC, ECML_I, and ECML_II Databases (with C4.5 algorithms applied)

sets, regardless of the skewed rate of the datasets. Promisingly, the MVC-IM strategy conducted almost perfect AUC values against the two most highly imbalanced data sets, namely ECML_I and ECML_II, with AUC values of 0.990 and 0.993, respectively. These values outperformed that of all other five tested systems, namely TILDE, RelAggs, CrossMine, MVC, and SimFlat approaches.

Furthermore, the experimental results as presented in Table 2 show that the MVC-IM algorithms significantly benefited from the framework of learning from multiple views. Recall that, the flat file conducted by the SimFlat approach consists of attributes from all views of the MVC-IM strategy. That is, in the MVC-IM algorithm, a set of feature sets (each feature set corresponds to a view in the MVC-IM strategy) were separately employed to learn the skewed classes. On the other hand, the SimFlat approach combined all sets of feature set into a flat file, and then learned from all these features available. When comparing the AUC values resulted from the SimFlat and MVC-IM algorithms, the results presented in Table 2 show that the MVC-IM approach meaningfully outperformed the SimFlat method against almost all six data sets. For example, against the F682, F234AC, F400AC, ECML_I, and ECML_II data sets, the MVC-IM algorithm improved the AUC values over the SimFlat method by 29.9%, 29.5%, 21.6%, 4.5%, and 4.4%, respectively.

In addition, further study of the experimental results show that the MVC-IM algorithm benefited from the use of the majority voting strategy, when compared with the MVC method, where a meta learning approach is employed for view combination. The results in Table 2 show that the MVC-IM approach outperformed the MVC method against all tested cases, in terms of AUC obtained.

ROC analysis: The ROC graphs in Figures 2 again demonstrate that, in general, the MVC-IM method potentially performs better against both the minority and majority classes, as compared with the RelAggs, TILDE, and SimFlat approaches. The results show that, when against the F400AC, F234AC, and F682AC data sets, the ROC curves of the MVC-IM strategies dominated that of the TILDE and SimFlat approaches. In these three cases, the ROC curves of the MVC-IM method were overlapped with that of the RelAggs and CrossMine methods in the ROC space. Promisingly, against the two most imbalanced data sets ECML_I and ECML_II, the ROC curves of the MVC-IM method dominated that of all other tested systems. In other words, in these two highly skewed-class tasks, the MVC-IM strategy performed superior to the other five tested systems over all possible decision thresholds.

In addition, these ROC graphs confirmed that the MVC-IM algorithms significantly benefited from learning from multiple views of a relational database. For example, against almost all the tested data sets, the ROC curves conducted by the MVC-IM strategies dominated that of the SimFlat approaches. One exception is against the less skewed MUT188 database, in which the MVC-IM's ROC curve was slightly overlapped with the ROC curve yielded by the SimFlat method (at the ROC space with high TP rates). Also, these experimental results indicate that the MVC-IM method benefited from the majority voting strategy, when compared to the MVC meta-learning approach.

Table 3. AUCs of the RelAggs, SimFlat, TILDE, CrossMine, MVC, and MVC-IM approaches (using Naive Bayes)

Data set	RelAggs	SimFlat	TILDE	CrossMine	MVC	MVC-IM
MUT188	0.899	0.909	0.807	0.912	0.848	**0.947**
F400AC	0.650	0.685	0.591	**0.824**	0.823	0.751
F234AC	0.630	0.591	0.591	0.785	0.685	**0.812**
F682AC	0.686	0.692	0.590	0.793	**0.845**	0.769
ECML_I	0.956	0.929	0.553	0.961	**0.980**	**0.980**
ECML_II	0.964	0.919	0.608	0.971	0.967	**0.989**

Using Naive Bayes probabilistic learners In the second experiment, Naive Bayes was used by the MVC-IM, RelAggs, and SimFlat approaches. Also, Naive Bayes was employed as the view learners of the MVC method. We present the ROCs and AUCs obtained for each of the six learning tasks in Figure 3 and Table 3, respectively.

AUC analysis: The AUCs listed in Table 3 again show that for most of the learning tasks the MVC-IM method was able to improve over all other tested systems, namely the RelAggs, TILDE, CrossMine, MVC, and SimFlat algorithms,

when Naive Bayes learners were used. Exceptions were against the F400AC and F682AC data sets when compared with the CrossMine and MVC algorithms. Again, the MVC-IM method produced the best results for very highly imbalanced data sets.

When comparing results obtained by using C4.5 as learning methods, the MVC-IM strategy yielded consistent results in terms of AUCs obtained regardless of the use of different learning methods. However, the RelAggs methods performed slightly worse in five of the six learning tasks, as compared to that of using the C4.5 as learners. Experimental results also indicate that the MVC method performed somewhat better than when using C4.5 decision trees as view learners. In addition, the SimFlat method had AUC improvement in half of the learning problems, but yielded AUC loss against another half of the learning tasks, compared to results of using C4.5 as mining strategy. Note that, the applied propositional learning algorithm didn't affect the performance of the TILDE and CrossMine methods.

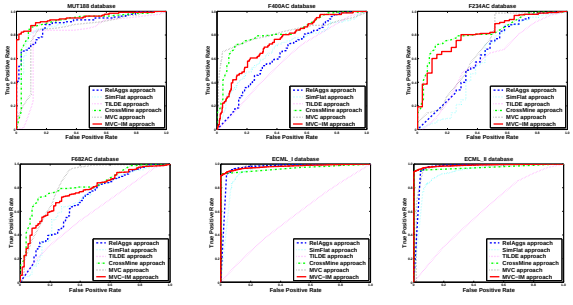


Fig. 3. ROC Curves for MUT188, F400AC, F234AC, F682AC, ECML_I, and ECML_II Databases (with Naive Bayes algorithms applied)

ROC analysis: Analysis on ROC graphs as shown in Figures 3 confirmed that the MVC-IM method was superior as compared to the RelAggs, TILDE, and SimFlat approaches. The ROC curves of the MVC-IM methods dominated over ROC curves produced by the other three approaches. When compared with the CrossMine and MVC algorithms the ROC curves produced by the MVC-IM strategy dominated that of the CrossMine and MVC methods against the MUT188 and ECML_II data sets. Against the other four tested data sets, the curves obtained by these three methods overlapped in the ROC space.

5 Conclusions

Knowledge discovery applications of commercial databases have often been hindered by the lack of powerful skew-class learning algorithms. This paper presents a novel strategy to deal with imbalanced multirelational data. The proposed algorithm achieved promising results when compared with five other multirelational

learning methods against six imbalanced learning problems, in terms of AUCs and ROC curves obtained. Furthermore, an important result indicates that the MVC-IM method is superior when the class imbalanced is very high.

Our future work will involve testing this method against very large and highly imbalanced databases. It would also be interesting to examine the influence of different model combination techniques and view validation strategies.

References

1. P. Berka. Guide to the financial data set. In *A. Siebes and P. Berka, editors, PKDD2000 Discovery Challenge*, 2000.
2. H. Blockeel and L. D. Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
3. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Workshop on Computational Learning Theory*, 1998.
4. R. Bryll, R. Gutierrez Osuna, and F. Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *PR*, 36(6):1291–1302, 2003.
5. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Jour. of Arti. Inte. and Rese.*, 16:321–357, 2002.
6. P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD '99*, pages 155–164, 1999.
7. T. Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.
8. H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.
9. H. Guo and H. L. Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *SIGKDD Explr. Newsl.*, 6(1):30–39, 2004.
10. H. Guo and H. L. Viktor. Mining relational databases with multi-view learning. In *MRDM '05*, pages 15–24. ACM Press, 2005.
11. G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *UAI*, pages 338–345, 1995.
12. A. J. Knobbe. *Multi-Relational Data Mining*. PhD thesis, Uni. Utrecht, 2004.
13. M.-A. Krogel. *On Propositionalization for Knowledge Discovery in Relational Databases*. PhD thesis, Otto-von-Guericke-Universit Magdeburg, 2005.
14. A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD '05*, pages 157–166, 2005.
15. J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, USA, 1993.
16. J. R. Quinlan and R. M. Cameron-Jones. Foil: A midterm report. In *ECML*, pages 3–20, 1993.
17. P. Sen and L. Getoor. Cost-sensitive learning with conditional markov networks. In *ICML '06*, pages 801–808, New York, NY, USA, 2006. ACM Press.
18. A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: a study in first-order and feature-based induction. *Artif. Intell.*, 85(1-2):277–299, 1996.
19. G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.
20. I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, CA, USA, 2000.
21. X. Yin, J. Han, J. Yang, and P. S. Yu. Crossmine: Efficient classification across multiple database relations. In *ICDE '04*, Boston, 2004.

Stratified Gradient Boosting for Fast Training of Conditional Random Fields ⋆

Bernd Gutmann[1] and Kristian Kersting[2]

[1] Department of Computer Science, Katolieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
[2] CSAIL, Massachusetts Institute of Technology
32 Vassar Street, Cambridge, MA, 02139-4307, USA

Abstract. Boosting has recently been shown to be a promising approach for training conditional random fields (CRFs) as it allows to efficiently induce conjunctive (even relational) features. The potentials are represented as weighted sums of regression trees that are induced using gradient tree boosting. Its large scale application such as in relational domains, however, suffers from two drawbacks: induced trees can spoil previous maximizations and the number of generated regression examples can become quite large. In this paper, we propose to tackle the latter problem by injecting randomness into the regression estimation procedure by subsampling regression examples. Experiments on a real-world data set show that this sampling approach is comparable with more sophisticated boosting algorithms in early iterations and, hence, provides an interesting alternative as it is much simpler to implement.

1 Introduction

Sequential data are ubiquitous and are of interest to many communities. Such data can be found in virtually all application areas of machine learning including computational biology, activity recognition, information extraction. As an example, consider the task of marking all proteins in abstracts of biological publications. One appealing approach to such sequence labeling problems are [7]'s *conditional random fields* (CRFs). They are undirected models encoding the conditional dependency $P(Y|X)$ and have outperformed HMMs [9] on language processing tasks such as information extraction and shallow parsing. In contrast to generatively trained HMMs, the discriminatively trained CRFs are designed to handle non-independent input features such as such as the molecular weight and the neighboring acids of an amino acid.

The flexibility, however, comes at the expense of severe training costs. To this end, fast and integrated feature induction and parameter estimation techniques have been proposed. [8]'s Mallet system employs the BFGS algorithm,

⋆ An earlier version of this work appeared as 4-pages extended abstract in the electronic working notes of the 5th International Workshop on Mining and Learning with Graphs (MLG'07), August 1–3, 2007, Universita degli Studi di Firenze, Florence, Tuscany, Italy.

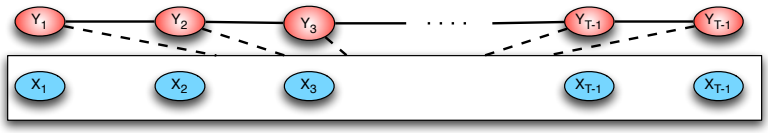


Fig. 1. Graphical representation of linear-chain CRF.

which is a second-order parameter optimization method that deals with parameter interactions, and induces features iteratively. Starting with a single feature, conjunctions of features are iteratively constructed that significantly increase conditional log-likelihood if added to the current model. Recently, [1] proposed a boosting approach, called TreeCRF, which is competitive to Mallet. TreeCRF follows [3]'s gradient tree boosting algorithm, i.e., the potential functions are represented by sums of regression trees, which are grown stage-wise in the manner of Adaboost [2]. Each regression tree can be viewed as defining several new feature combinations, one for each path in the tree from the root to a leaf. Thus, the features can be quite complex; even relational conjunctions as shown by Gutmann and Kersting's TildeCRF [5].

One major drawback of the gradient tree boosting approach is that the number of generated regression examples can become very large. If we have 3 labels and 100 training sequences of length 200, then the number of training examples for each label k is $3 \cdot 100 \cdot 200 = 60,000$. To get around this, we propose a sampling strategy tailored to gradient tree boosting for (relational) CRFs. More precisely, we introduce a stratified sampling approach for CRFs, conduct an experimental evaluation, and examine the influence of the subsampling, the line search and the gradient method (steepest ascent vs. conjugated gradient) on the predictive performance.

We proceed as follows. After reviewing CRFs and gradient tree boosting, we discuss our stratified sampling scheme. Before concluding, we we evaluate our approach on a real-world information extraction dataset.

2 Gradient Tree Boosting for CRFs

CRFs are undirected graphical models that encode conditional probability distributions using a given set of features. We will focus on *linear-chain* CRF models, cf. Figure 1.

Let G be an undirected graphical model over sets of random variables X and Y. For linear-chain CRFs, $X = \langle x_{i,j} \rangle_{j=1}^{T_i}$ and $Y = \langle Y_{i,j} \rangle_{j=1}^{T_i}$ correspond to the input and output sequences such that Y is a labeling of an observed sequence X. The conditional probability of a state sequence given the observed sequence is defined as

$$P(Y|X) = Z(X)^{-1} \exp \sum_{t=1}^{T} \Psi_t(y_t, X) + \Psi_{t-1,t}(y_{t-1}, y_t, X),$$

where $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$ are potential functions [3] and $Z(X)$ is a normalization factor over all state sequences X so that each potential contributes overall probability.

2.1 Training

Typically, it is assumed that the potentials factorize according to a set of features $\{f_k\}$, which are given and fixed, so that

$$\Psi(y_t, X) = \sum \alpha_k g_k(y_t, X)$$
$$\text{and } \Psi(y_{t-1}, y_t, X) = \sum \beta_k f_k(y_{t-1}, y_t, X)$$

respectively. The model parameters are now a set of real-valued weights α_k, β_k; one weight for each feature. To estimate them, a conditional maximum likelihood approach is typically followed. That is, the (conditional) likelihood of the training data given the current parameter Θ_{m-1} is used to improve the parameters. Normally, one uses some sort of gradient search for doing this:

$$\Theta_m = \Theta_0 + \delta_1 + \ldots + \delta_m$$
$$\text{where } \delta_m = \eta_m - M \cdot \tfrac{\partial}{\partial \Theta_{m-1}} \textstyle\sum_i \log P\left(y_i | x_i; \Theta_{m-1}\right)$$

is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient.

2.2 Training via Gradient Tree Boosting

Gradient tree boosting interleaves parameter estimation and feature selection. More precisely, one starts with some initial potential Ψ_0, e.g. the zero function, and adds iteratively corrections

$$\Psi_m = \Psi_0 + \Delta_1 + \ldots + \Delta_m.$$

In contrast to the standard gradient approach, Δ_i denotes the so-called functional gradient, i.e.,

$$\Delta_m = \eta_m \cdot E_{x,y}\left[\frac{\partial}{\partial \Psi_{m-1}} \log P(y|x; \Psi_{m-1})\right].$$

Since the joint distribution $P(x, y)$ is unknown, one cannot evaluate the expectation $E_{x,y}$. Instead ones evaluates the gradient function at every position in every training example and fit a regression tree to these derived examples. More precisely, setting $F(y_{t-1}, y, t, X) = \Psi(y_t, X) + \Psi(y_{t-1}, y_t, X)$, the gradient becomes (see [1, 5] for more details),

$$\frac{\partial \log P(Y|X)}{\partial F(u, v, w_d(X))} = I(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v) - P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v | w_d(X)), \quad (1)$$

[3] A *potential function* is a real-valued function that captures the degree to which the assignment y_t to the output variable fits the transition from y_{t-1} and X.

Algorithm 1 The inner loop of gradient tree boosting for generating the regression examples

function GENEXAMPLES($k, Data, Pot_m$)
  $S := \emptyset$ ▷ Initialize relational regression examples
  for all $(X_i, Y_i) \in Data$ **do** ▷ Iterate over all training examples
    $(\alpha, \beta, Z(X_i)) = \text{FORWARDBACKWARD}(X_i, T, K)$ ▷ Compute forward and backward probabilities
    for $1 \leq t \leq T_i$ **do** ▷ Iterate over all positions
      for $1 \leq k' \leq K$ **do** ▷ Iterate over all class labels
        ▷ Compute value of gradient at position t for class label k

$$P(y_{t-1} = k', y_t = k|X_i) := \frac{\alpha(k', t-1) \cdot \exp(F_m^k(k', w_t(X)) \cdot \beta(k, t)}{Z(X_i)}$$

$$\Delta(k, k', t) := I(y_{t-1} \subseteq_\Theta k', y_t \subseteq_\Theta k) - P(y_{t-1} \subseteq_\Theta k', y_t \subseteq_\Theta k|X_i)$$

        $S := S \cup \{((w_t(X_i), k'), \Delta(k, k', t))\}$ ▷ Update set of relational regression examples
      end for
    end for
  end for
  return S
end function

where I is the indicator function, $\subseteq_\Theta$ denotes that u matches/subsumes y, and $P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v|w_d(X))$ is the probability that class labels u, v fit the class labels at positions $d, d-1$. By evaluating the gradient at every known position in the training data and fitting a regression model such as a relational regression tree, cf. Figure 2, to these values, one gets an approximation of the expectation $E_{x,y}[\partial/\partial\Psi_{m-1}]$ of the gradient. In order to speed-up computations, not the complete input X is typically used but only a window $w_d(X) = x_{d-s}, \ldots, x_d, \ldots, x_{d+s}$, where s is a fixed window size.

2.3 Conjugate Direction Boosting

Reconsider the basic gradient-ascent optimization approach. One of the problems with choosing the step size doing a line search is that a maximization in

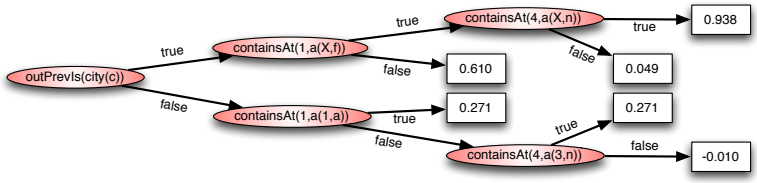


Fig. 2. A (relational) regression tree. Nodes denote tests; leaves are the predicted values.

Algorithm 2 Conjugated gradient tree boosting with line search and sampling.

```
1: function CGTreeBoost(Data, L)
2:     for 1 ≤ m ≤ M do                                  ▷ Iterate Functional Gradient
3:         S := ∅
4:         for 1 ≤ k ≤ K do                              ▷ Iterate through the class labels
5:             (S_Pos, S_Neg) := (S_Pos, S_Neg) ∪
6:                 GenExamples(k, Data, F_{m−1}, m)
7:         end for
8:         S_Sample := Sample((S_Pos, S_Neg))
9:         Δ_m := FitRelRegressTree(S_Sample, L)
10:        if m = 1 then
11:            d_1 = Δ_1                                 ▷ Initial conjugate direction
12:        else
13:            β_m = ⟨Δ_m, Δ_m − Δ_{m−1}⟩ / ⟨Δ_{m−1}, Δ_{m−1}⟩   ▷ Polak-Ribiére formula
14:            d_m = Δ_m + β_m · d_{m−1}                 ▷ Next conjugate direction
15:        end if
16:        η_m :=LineSearch(Data, F_{m−1}, d_m)          ▷ Line Search along d_m
17:        F_m := F_{m−1} + η_m · d_m                    ▷ Model udpate
18:    end for
19:    return F_M                                        ▷ Return Potential
20: end function
```

one direction could spoil past maximizations. To avoid this, conjugate gradient boosting methods [6] compute so-called conjugate directions in the function space, which are orthogonal and, hence, do not spoil previous maximizations. The step size is estimated along these directions doing line searches.

More precisely, the empirical angle β_m between Δ_m and Δ_{m-1} on the training examples given the current gradient Δ_m is computed. As shown in Alg. 2, the current gradient plus the old weighted gradient multiplied by the calculated angle is added to the current model, $d_m = \Delta_m + \beta_m \cdot d_{m-1}$. The angle β_m can be calculated by evaluating the Polak-Ribiére formula for each example. Every weighted gradient d_m is a linear combination of the gradients $\Delta_1, \ldots, \Delta_m$. It can be shown that

$$d_t = \sum_{i=1}^{m} \beta_{i,m} \cdot \Delta_i$$
$$\text{where } \beta_{m,m} = 1 \text{ and } \beta_{i,m} = \prod_{j=i+1}^{m} \beta_j \text{ if } i < m.$$

3 Stochastic gradient tree boosting

One major drawback of the gradient tree boosting approach is that the number of generated regression examples can become very large. In every iteration, $k^2 \times n \times l$ regression examples are generated (k is the number of labels, n the number of training examples, l the average sequence length).

An obvious modification to speed up gradient tree boosting is to use only a subset of the original data. This subset is drawn randomly in every iteration

Algorithm 3 Stochastic gradient tree boosting.

```
1: function CGTreeBoost(Data, L, N)
2:     for 1 ≤ m ≤ M do                          ▷ Iterate Functional Gradient
3:         S := ∅
4:         Data′ := randomly sample a subset of size N from Data
5:         for 1 ≤ k ≤ K do                      ▷ Iterate through the class labels
6:             S := S ∪ GenExamples(k, Data, F_{m−1}, m)
7:         end for
8:         Δ_m := FitRelRegressTree(S_Sample, L)
9:         F_m := F_{m−1} + d_m                  ▷ Model udpate
10:    end for
11:    return F_M                                ▷ Return Potential
12: end function
```

which ensures that the complete training data contributes to the learned model. This modification, *stochastic gradient tree boosting*, was originally proposed in [4]. Algorithm 3 shows the pseudocode, where N is the size of the sample. For $N = \text{size}(Data)$ the algorithm behaves like standard gradient tree boosting. Stochastic gradient tree boosting is a kind of bagging approach. Experimental results show that it improves the runtime (which is an obvious result) and can also increase the accuracy of the learned model.

Indeed, stochastic gradient tree boosting can directly be used for fast training of conditional random fields. It is, however, a general-pupose technique and methods tailored to the problem at hand, namely training conditional random fields are likely to improve performance. In [6] we showed that gradient tree boosting for CRFs induces an expectation bias on the generated regression examples: For every observed (called positive) example in the training data $k-1$ unobserved (negative) regression examples are generated (k is the number of possible labels in the output sequence). For higher k, the regression tree learner spends a lot of time to fit the tree to unobserved examples, whereas the predictive accuracy on positive examples is lower. We proposed to reweight the examples such that the empirical ratio of positive to negative examples is equal. However, the problem remained that both the regression tree learner and the generating step of gradient tree boosting hat to consider all data.

Indeed, this approach rebalances the positive and negative examples. The regression tree learner, however, must still consider all examples. To this end, we propose to use stratified sampling. This means that we use two different sampling strategies, one for the observed regression examples and one for the negative examples. The idea is to use reduce the difference between the number of positive and negative examples. Doing so, we increase both the predictive performance on the positive examples and we reduce the time needed by the regression tree learner.

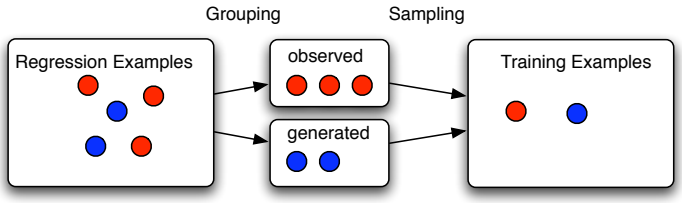


Fig. 3. Stratified sampling methodology.

4 Stratified Sampling

There are several other possible sampling approaches for gradient tree boosting. [3] suggested two techniques to speed up the learning process by reducing the number of regression examples. *Influence Trimming* ignores regression examples with absolute value smaller ϵ. We do not investigate this method but instead consider *sampling*:

> *use only a randomly drawn subset of the generated examples to train the regression trees.*

Friedman's original proposal is to subsample uniformly from all training examples. This strategy is suboptimal for training CRFs. Most of the generated examples are likely to have never been observed. Recall that for each *observed* y_{t-1} we generate $k-1$ many *expected* labels y_t. Thus, with an increasing number of labels, the probability of sampling an observed regression examples decreases (keeping the training set fixed); roughly at the scale of $\mathcal{O}(\frac{1}{k^2})$. In turn, the influence of the expected examples, the ones we have not observed, increases and gradient boosting is likely to induce meaningless models. This is truely an undesirable property of uniform sampling. To overcome this, we propose to a stratified sampling scheme.

Stratified sampling, cf. Figure 3 is a method of sampling in which it is desirable to maintain certain characteristics of the data set in any subset sampled. It is achieved by firstly partitioning the data set into a number of mutually exclusive subsets of cases (strata), each of which is representative of some aspect of the real-world process involved. Sampling from the population is then achieved by randomly sampling from the various subsets so as to achieve representative proportions.

In our case at hand there are two natural strata: the observed and the generated regression examples. Thus, when we generate the regression examples, we mark those examples as positive for which the identity function I returnes 1.0 (see Equation (1)). Then, we sample with different strategies for the positive and negative examples S_{Pos} and S_{Neg}. For instance, we can take all positive examples and sample a fraction of the negative examples that is 3 times bigger than the positive examples. We denote this strategy by 1p/3n.

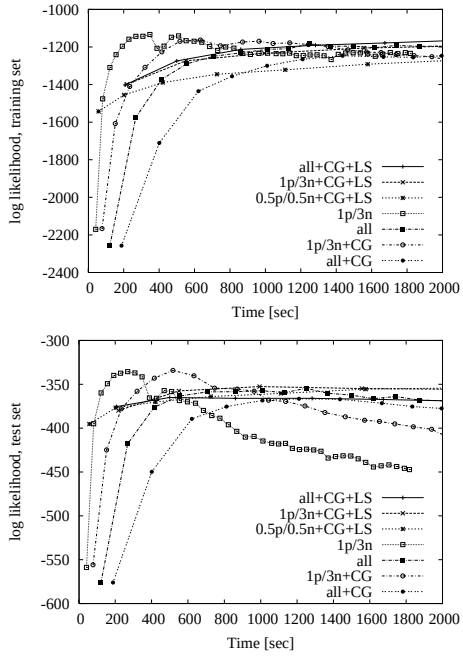


Fig. 4. 5-fold cross-validated log likelihoods on training and test set; all: all regression examples were used, 1p/3n: all positive and 3 times more negative regression examples were sampled, 0.5p/0.5n: only half of the positive and half of the negative examples were sampled, CG: conjugated gradients were used, LS: a line search was used

5 Experiments

Our intention was to examine the influence of the sampling rate, the line search and gradient method on the predictive performance. To this aim, we integrated stratified sampling in TildeCRF for boosting relational CRFs, which is implemented in YAP Prolog. The regression tree learner was implemented in hipP Prolog.

We ran several experiments on a subset of [10]'s *subcellular-location* data set. The data set consists of sentences of medical abstracts, where each word in the input sequence is augmented with the word type and the phrase type. The task is to find proteins and their location within the DNA. More precisely, the output sequence consists of sequences over `protein, location, none`. An example of

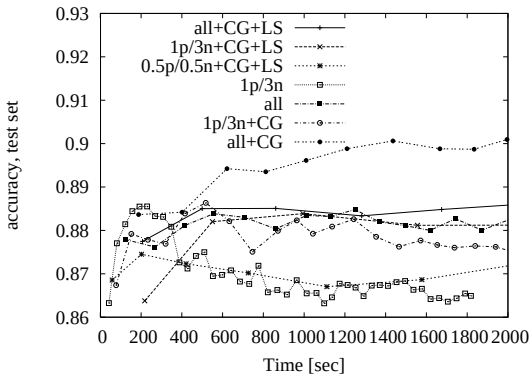


Fig. 5. 5-fold cross-validated accuracy on the test set.

for a training example is

$$[\texttt{npSeg}(\texttt{unk}(\texttt{'rat7p'})), \texttt{vpSeg}(\texttt{cop}(\texttt{'is'})), \texttt{vpSeg}(\texttt{v}(\texttt{'located'})),$$
$$\texttt{ppSeg}(\texttt{prep}(\texttt{'at'})), \texttt{npSeg}(\texttt{art}(\texttt{'the'})), \texttt{npSeg}(\texttt{adj}(\texttt{'nuclear'})),$$
$$\texttt{npSeg}(\texttt{unk}(\texttt{'rim'})), \ldots\}$$
$$[\texttt{protein}, \texttt{none}, \texttt{none}, \texttt{none}, \texttt{none}, \texttt{location}, \ldots]$$

We removed all training sequences that do not contain a protein or location tag. Because of memory limitations[4], we only took 25% of the resulting data set, namely 195 input/output sequence pairs. and only sampling techniques were running with. to still compare to the deterministic approach. For all our experiments, we did a 5-fold cross validation, 156 examples for training and 39 for testing the trained model. We tried several sampling strategies and ran the experiments with and without line search and conjugated gradients. We used a window size of 5 elements, the tree learner learned the regression trees up to depth 20 but it stop splitting a node (pre-pruning) when the variance was smaller than 0.0001.

Figure 4 shows the results plotted against the training time. Strategy 1p/3n, which uses all positive and 3 times as many negative regression examples as positive ones, outperforms all other approaches in early iterations. In this case we used only 4/9 of the training examples, since 1/9 of the examples where positive and 8/9 were negative ($k = 3$). It is remarkable that 1p/3n is even better than the 0.5p/0.5n+CG+LS strategy. Where we use 50% of the examples

[4] For larger data sets the deterministic boosting technique ran out of memory. Only the stochastic versions had a chance to keep running due to the reduction in memory consumptions.

and run furthermore a more sophisticated training algorithm. To understand why, one has to see that the line search is rather slow and furthermore 50% of the observed data is not considered. Whereas in the 1p/3n one uses all the observed data. Furthermore, all methods actually achieve similar performances in terms of likelihood, i.e., the objective function we are maximizing in later iterations. Note, however, that only the stratified sampling approach achieves the highest training and test likelihoods already after about 400 sec. This would also coincide with the stopping point of the optimization if we had implemented a stopping rule. We did not do so in order to see the long term behaviors of the boosting algorithms. To summarize,

> stratified sampling speeds up computations and achieves higher log-likelihood estimates.

To complete the view on the performance, we investigated the 5-fold cross-validated accuracy. Figure 5 shows the accuracy of the trained model on the test data. The accuracy is defined as c/a where c is the number of correctly predicted positions in all sequences and a is the number of all positions in all sequences. One can see that strategy 1p/3n performs best for the early iterations and afterwards all+CG outperforms all the other strategies. In other words,

> subsampling negative examples yields slightly lower accuracies but does not degrade performance.

Basically all methods are all in close range and comparable. This supports our results from [6]. Furthermore, one can see that the line search does not significantly increase the accuracy. We believe that the line search actually tends to overfit .

Finally, we ran several other sampling strategies, 1p/2n, 0.25p/0.25 and so on. The results are not presented here as they follow the general picture:

- Stratified sampling taking all positive examples but subsampling negative ones is much faster than deterministic gradient tree boosting and achieves comparable performance.
- The less examples (in particular positive ones) are used, the worse the performance.

6 Conclusions and Future Work

We have presented a stochastic gradient tree boosting approach to training (relational) CRFs. In contrast to existing stochastic techniques, we do not follow a uniform sampling strategy but sample positive/observed and negative/generated regression examples following different strategies. The experimental results have shown that this stratified sampling scheme indeed speeds up computations and can improve performance. It actually performs better than uniform sampling. Moreover, compared to more advanced boosting techniques such as conjugate direction boosting, it is much simpler to implement. Thus, we view stochastic

boosting based on stratified sampling as an attractive and promising approach to scaling up CRF training to large data sets.

Indeed more experiments on larger data sets have to be conducted to confirm the scale up behavior. The reduction of examples on our rather small data set are encouraging; for larger data sets, one can expect higher reduction rates. It is also interesting to investigate stratification of output class, not only of positive/negative regression examples.

Acknowledgments

The authors would like to thank the anonymous reviewers for the valuable comments, Luc De Raedt for his support, and Vítor Santos Costa for his help with the Prolog system YAP. This work has been supported by the Research Foundation-Flanders (FWO-Vlaanderen)

References

1. T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. 21st International Conf. on Machine Learning*, pages 217–224. ACM, 2004.
2. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *In Proceedings of ICML-96*, pages 148–156. Morgan Kaufman, 1996.
3. J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29, 2001.
4. J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, February 2002.
5. B. Gutmann and K. Kersting. Tildecrf: Conditional random fields for logical sequences. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proc. of the 15th European Conference on Machine Learning (ECML-2006)*, volume 4212 of *LNAI (Lecture Notes in Artificial Intelligence)*, pages 174–185, Berlin, Germany, September 2006. Springer.
6. K. Kersting and B. Gutmann. Unbiased conjugate direction boosting for conditional random fields. In T. Gärtner, G.C. Garriga, and T. Meinl, editors, *Working Notes of the ECML-2006 Workshop on Mining and Learning with Graphs (MLG 2006)*, pages 157–164, Berlin, Germany, September 2006. Short paper.
7. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th Int. Conf. on Machine Learning (ICML-01)*, pages 282–289, 2001.
8. A. McCallum. Effciently inducing features of conditional random fields. In F. Bacchus and T. Jaakkola, editors, *Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 403–410, Edinburgh, Scotland, July 26-29 2003.
9. L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–285, 1989.
10. M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden markov models for information extraction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 427–433, 2003.

A Restart Strategy for Fast Subsumption Check and Coverage Estimation

Ondřej Kuželka and Filip Železný

Intelligent Data Analysis Research Group
Dept. of Cybernetics, Czech Technical University in Prague
http://ida.felk.cvut.cz
{kuzelo1,zelezny}@fel.cvut.cz

Abstract. We study the runtime distributions of a simple subsumption check algorithm and show that in some conditions they exhibit heavy tails, indicating a possible runtime advantage achievable by randomizing and restarting the algorithm. Therefore we design ReSumEr, a restarted subsumption tester, incorporating randomization while preserving completeness. On generated graph data, ReSumEr outperforms the state-of-the-art subsumption algorithm Django (i) significantly in the YES region of the phase transition domain and (ii) in the entire phase transition domain given a sufficient size difference between the tested subsumer and subsumee. Importantly, we further show how, under a distributional assumption, a restarted strategy can be used to quickly obtain a maximum likelihood estimate of the coverage of a pattern (proportion of examples subsumed thereby) without requiring to verify subsumption for all examples. We implement this technique in the program ReCovEr and show that it provides accurate coverage estimates in favorable runtimes.

1 Introduction

Recent statistical performance studies of search algorithms in difficult combinatorial problems [1, 2] have demonstrated the benefits of randomizing and restarting the search procedure. Specifically, it has been found that if the search cost distribution of the non-restarted randomized search exhibits a slower-than-exponential decay (that is, a "heavy tail"), restarts can reduce the search cost expectation. In [6] we have demonstrated the benefits of randomized restarted strategies in the lattice search conducted by an inductive logic programming system. While the size of pattern spaces represents one source of the complexity of relational data mining, another such source follows from the problem of verifying the subsumption relation between a relational pattern and an example.

This paper first focuses on this latter problem by investigating the possible benefits of a randomized restarted strategy in subsumption testing. Previous research has demonstrated that vast gains in efficiency can be achieved by using unorthodox subsumption algorithms as opposed to standard procedures provided e.g. by a Prolog engine. The pioneering work [4] introduced a tractable

approximation to the subsumption test called *stochastic matching*. This randomized algorithm is incomplete in that its failure to prove subsumption in a finite number of steps does not refute the subsumption. On the contrary, we aim at preserving completeness in our randomized restarted procedure called ReSumEr. A complete deterministic approach, called Django, was presented in [3]. Django converts subsumption into a constraint satisfaction problem (CSP) then solved by state-of-the-art heuristic techniques. Django was shown to outperform by orders of magnitude the subsumption testing mechanism used in ILP. Therefore we use Django as the baseline algorithm for comparative experiments with ReSumEr.

Secondly, this paper focuses on the development of an algorithm for fast estimation of pattern coverage, i.e. the proportion of a given set E of examples subsumed thereby. The paper [5] is relevant to this part of our work, in that it estimates total coverage by checking subsumption with respect to a small sample of E. We take a different approach in our algorithm, called ReCovEr, enabled by the fact that ReSumEr's runtimes neccessarily follow an exponential distribution. ReCovEr computes pattern coverage through a maximum-likelihood estimation of parameters of this exponential distribution, on the basis of information collected during a (possibly small) sequence of restarts of ReSumEr.

Informally, the main intended contribution of this work is to support efficient mining in large relational structures by enabling fast pattern evaluation. In real-life applications, e.g. in bioinformatics, such structures can typically be represented by oriented graphs. For this reason, we will assume the oriented graph structure of examples and hypotheses, and the subsumption test will coincide with subgraph matching.

The rest of the paper is organized as follows. Section 2 defines the syntax of patterns and examples considered and explains how examples are generated for sakes of empirical measurements throughout the paper. In Section 3 we devise a simple subsumption test algorithm, investigate the runtime distributions of both its non-restarted and restarted version (ReSumEr), and empirically evaluate ReSumEr in comparison to Django. Section 4 explains the maximum-likelihood technique for estimating hypothesis coverage implemented in the algorithm ReCovEr, and tests its efficiency and estimation accuracy. Section 5 concludes the paper.

2 Preliminaries

In the rest of the paper we assume that patterns and examples are oriented graphs where each vertex may be assigned one of two possible colors. In the dual, relational-logic representation, examples e and patterns P are viewed as conjunctions of positive atoms, each being one of $edge(t_1, t_2)$, $black(t)$, $red(t)$ where t, t_1, t_2 are placeholders for terms. All terms in an example e are assumed to be constants and all terms in a pattern P are assumed to be variables. The correspondence between the graph and logic representation is such that vertices correspond to terms and the orientation of and edge is given by the order of

term appearance in the corresponding atom. We will refer to the described dual notions interchangeably. When needed, conjunctions will be treated as atom sets, e.g. for two conjunctions a and b, $a \subseteq b$ will denote that b contains all atoms contained by a.

Algorithm 1 *SubsumptionCheck*(P, e): A simple subsumption test algorithm

Input: Pattern P, example e;

if $P \subseteq e$ **then**
  return YES
else
  Choose variable V from P using a heuristic function *(see main text)*
  for $\forall S \in PossibleSubstitutions(V, P, e)$ *(see main text)* **do**
    $SearchedNodes \leftarrow SearchedNodes + 1$
    Substitute V with S
    if $\forall W \in Adjacency(V) : PossibleSubstitutions(W, P, e) \neq \emptyset$ **then**
      if $SubsumptionCheck(P, e) =$ YES **then**
        return YES
      end if
    end if
  end for
  return NO
end if

Algorithm 2 *SubstitutionPossible*(V, C, P, e): Returns NO if P cannot subsume e when V is substituted by C.

Input: Variable V, constant C, Pattern P, example e;

for $\forall A \in P$ such that atom A contains variable V **do**
  $A' \leftarrow$ replace all occurrences of variable V in atom A by C.
  if $A'\theta \nsubseteq E$ *(easy to check for a single atom A)* **then**
    return NO
  end if
end for
return YES

We consider a simple heuristic algorithm (Algorithm 1) for verifying whether a pattern P subsumes an example e. Similarly to Django [3] this algorithm is inspired by the CSP framework, conducting backtracking search with forward checking, using a variable selection heuristic and randomization. The heuristic function aims at choosing variables whose substitution makes it likely that an inconsistency, if exists, is detected soon. For a variable V, the function returns the sum of occurrences of variables in pattern P that have already been grounded and that share at least one literal with V. The variable which maximizes this function is selected; in case of a tie, a random choice is made with uniform probability among the highest scoring variables. The function $PossibleSubstitutions(V, P, e)$

returns all constants C for which $SubstitutionPossible(V, C, P, e)$ (Algorithm 2) returns YES.

Algorithm 3 $RandomGraph(n, p)$: A generator of uniform random graphs

Input: Integer n, Real p;

Let V be a set of n vertices and G an empty edge set.
for $\forall \{v_i \in V, v_j \in V | v_i \neq v_j\}$ **do**
  With probability p, $G \leftarrow G \cup \{v_i, v_j\}$
end for
For all edges in G choose a random orientation, and for all vertices in V choose a random color with uniform probability from $\{red, black\}$.
return graph with vertex set V and edge set G

Algorithm 4 $ScaleFreeGraph(n, k)$: A generator of scale-free random graphs

Input: Integers n, k;

Let V be a set containing one vertex v_1, G be an empty edge set.
for $i \leftarrow 2$ to n **do**
  $k' \leftarrow min\,(i-1, k)$
  Create vertex v_i
  Connect v_i to k' distinct vertices $v_1, ..., v_k$ chosen from the set V with probability proportional to their degrees
  $G \leftarrow G \cup \{(v_i, v_j) | j = 1...k\}$
end for
For all vertices in V choose a random color with uniform probability from $\{red, black\}$.
return graph with vertex set V and edge set G

To obtain a domain-independent runtime distribution of the algorithm, we test it on randomly generated patterns and examples. For generality, we devised two different graph generators for this purpose. The first (Algorithm 3) generates graphs where any two vertices are connected with a pre-set probability p (by an edge of a random orientation). The second (Algorithm 4) produces scale-free ("small world") graphs; here, an edge is attached to a vertex with probability increasing with the number of edges already connected to the vertex. In both algorithms, all vertices are colored as black with probability 0.5 and red otherwise. We will refer to the parameter p (k, respectively) of a random uniform (scale-free, respectively) graph as the *connectivity* of the graph.

3 ReSumEr: a restarted subsumption tester

We subjected Algorithm 1 to experiments with random sets of patterns and examples generated by Algorithm 3 (Algorithm 4, respectively), under various settings of n and p (n and k, respectively). Our objective was to verify the

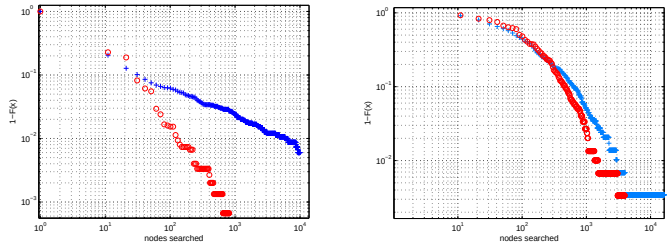


Fig. 1. The subsumption test runtime distribution for patterns with $n = 12$ (left) and $n = 14$ (right) vertices and connectivity $p = 0.2$. In both cases, examples had $n = 50$ vertices and connectivity $p = 0.5$. Both patterns and examples were randomly generated by Algorithm 3. A heavy tail is observed in the left panel for the non-restarted version (mostly upper curve) and significant speed-up is achieved by the restarted version (lower curve, exponential distribution despite its almost linear appearance). Although no heavy tail is observed in the right panel for the non-restarted version (mostly upper curve), small speed-up is still observed for the restarted version (lower curve).

presence of *heavy tails* in the runtime distributions $F(t)$. For a $t > 0$, $F(t)$ is the probability that the tested algorithm resolves a random subsumption instance in no more than t units of time, corresponding to the number of explored search nodes. A heavy tail is exhibited if $1 - F(t)$ decays at a power-law rate, i.e. slower than exponentially. Informally, a heavy-tailed distribution indicates the non-negligible probability of subsumption instances on which the checking algorithm gets stuck for an extremly long runtime. The presence of a heavy tail in an empirical runtime distribution $F(t)$ can be checked graphically, by plotting $1 - F(t)$ against t on a log-log scale. For a growing t, a heavy-tailed distribution here acquires a linear shape [2].

Our findings were not conclusive in that for various configurations mentioned above, some runtime distributions were heavy-tailed while others were not. Two examples are shown in Fig. 1 in blue; while differing very slightly in a single parameter (n) value, the respective distributions posses largely different shapes of the tails. We have not yet been able to establish a principled correspondence between the respective parameter values and the occurrence of heavy tails.

While the presence of heavy tails for some classes of subsumption instances indicates possible large runtime benefits achievable by a restarting strategy [2], its effect on the non-heavy-tailed classes may not be necessarily detrimental. We thus decided to assess the overall impact of restarting empirically. For this sake we designed a complete restarted randomized subsumption algorithm ReSumEr (Algorithm 5). Its completeness is guaranteed by the assumption that for the cutoff sequence $R(n)$, $R(n) \to \infty$ as $n \to \infty$. Note that the randomization is facilitated by tie-breaking in the heuristic function used in the embedded Algorithm 1.

Algorithm 5 *ReSumEr*(P, e, R): A restarted subsumption algorithm

Input: Pattern P, example e, cutoff sequence R;
$n \leftarrow 1$
repeat
  Answer $\leftarrow$ Run *SubsumptionCheck*(P, e) with number of searched nodes limited to $R(n)$
  $n \leftarrow n + 1$
until Answer YES or NO is returned
return Answer

The runtime distributions for ReSumEr, with an ad-hoc chosen restart sequence $R(n) = 10n^2 + 30$ are plotted in red in Fig. 1 for the earlier exemplified cases of both heavy-tailed and non-heavy-tailed behavior. In both cases, restarts generally reduce runtime, although the difference is more significant in the heavy-tailed case. The set of random subsumption instances naturally comprise of both satisfiable (where P subsumes e) and non-satisfiable instances. Of relevance, the times taken by ReSumEr on the non-satisfiable instances were in this experiment on average about 10^3 times higher than on the satisfiable ones. This is natural due to the 'iterative' character of ReSumEr; while satisfiability can in principle be shown in any single restart, non-satisfiability can only be shown after n restarts making $R(n)$ sufficiently high.

We next aimed to compare ReSumEr to a baseline algorithm used for subsumption in relational data mining. As explained earlier, the graph structures we here deal with are easily embedded into conjunctions of first-order positive atoms. Thus an obvious baseline algorithm candidate would have been the unification mechanism in Prolog. However the sizes of patterns and examples (tens of vertices in patterns, hundreds in examples) we focus on, consistently result in unmeasurably large runtimes of this procedure. A much faster alternative, which we adopt for comparisons, is represented by the state-of-the-art subsumption algorithm Django [3].

All experiments were conducted on the same computer. Django is implemented in C and we used its version 11. ReSumEr is implemented in JAVA. Figures 2 and 3 display the results for patterns and examples generated as uniform random graphs (Fig. 2) and scale-free graphs (Fig. 3). The comparative runtimes (top panels) are accompanied by the corresponding phase transition diagrams (bottom panels). The left (right, respectively) panels pertain to a smaller (larger, respectively) size difference between the patterns and the examples. Size is understood as the number of contained vertices.

We now note on the principled trends apparent from the results. First, ReSumEr consistently and significantly outperformed Django in the YES region of the phase transition spectrum.[1] Second, in the experiments with a larger size-difference between the patterns and the examples, ReSumEr was faster across the entire phase transition domain. Third, heavy-tailed behavior of Django was

[1] which corresponds to the left parts of all diagrams in Figures 2 and 3. Although the observed absolute difference is larger in the NO (right-hand side) region, in relative terms it is much smaller than the difference in the YES region.

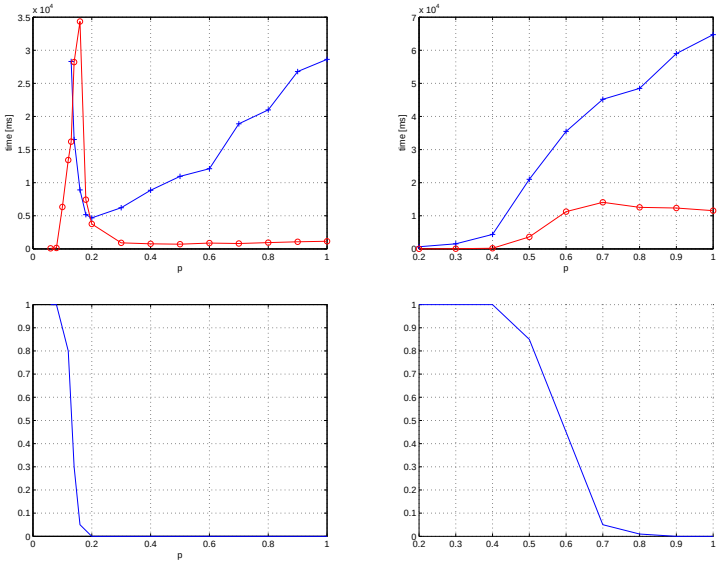


Fig. 2. Top: Comparisons of Django (blue) and ReSumEr (red) runtimes of subsumption checks between patterns and examples generated by Algorithm 3 with connectivity $p = 0.3$ for examples and varying p (horizontal axis) for patterns. In the left panel, patterns have 30 vertices and examples have 100 vertices. In the right panel, patterns have 10 vertices and examples have 200 vertices. All shown points are averages of 50 measurements. **Bottom:** The phase transition landscapes for the respective settings above: the probability that a random pattern with connectivity p (horizontal axis) subsumes a random example with connectivity $p = 0.3$.

observed: in spite of its typical measured runtimes in the order of milliseconds to seconds, occasional runs in satisfiable instances took up to tens of minutes and had to be curtailed. This resulted in Django's excessive runtimes in the top-left panel of Fig. 2 (Fig. 3, respectively) for $p \leq 0.1$ ($k = 3$, respectively). Heavy-tailed behavior is prevented by ReSumEr resulting in its vast superiority in the $p \leq 0.1$ region of Fig. 2, top-left panel. In Fig. 3, however, Resumer's averaged runtimes were also excessive for $k = 3$ and $k = 4$. Unlike for Django, here the reason was not in occasional excessive runs, but rather in the systematic increase of runtime required to complete the unsatisfiable subsumption instances. Fourth, the generally high runtimes of Django in the NO region are surprising. In particular, for large size-differences between the patterns and the examples (right panels in Fig. 2 and 3), Django's runtimes in the NO region were even

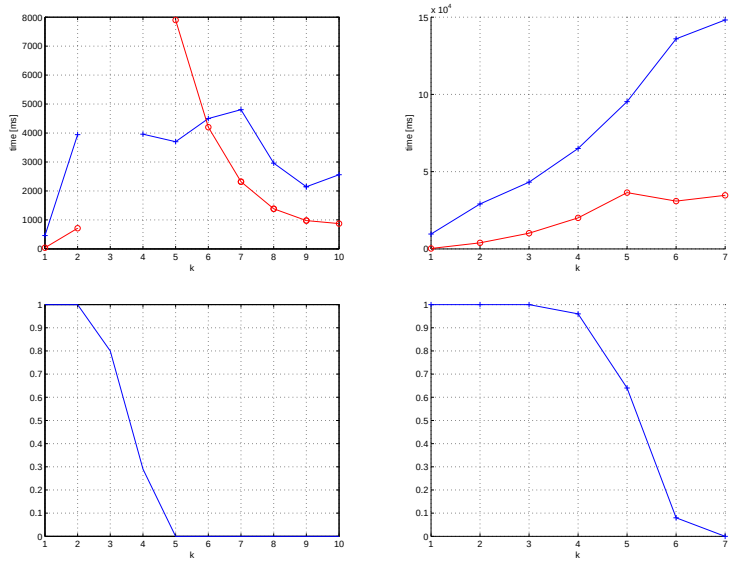


Fig. 3. Top: Comparisons of Django (blue) and ReSumEr (red) runtimes of subsumption checks between patterns and examples generated by Algorithm 4 with connectivity $k = 20$ for examples and varying k (horizontal axis) for patterns. In the left panel, patterns have 30 vertices and examples have 100 vertices; for some k, runtimes were not measurable (see main text). In the right panel, patterns have 20 vertices and examples have 500 vertices. All shown points are averages of 50 measurements. **Bottom:** The phase transition landscapes for the respective settings above: the probability that a random pattern with connectivity k (horizontal axis) subsumes a random example with $k = 20$.

consistently higher than those in the YES/NO (transition) region.[2] Although this phenomenon was also reported in [3] (Table 4 therein) for Django version 1, in general the runtimes reported by [3] for the NO region are much smaller than those in the transition region. Further investigation is thus needed to clarify this discrepancy in light of the differences between our experimental setting and that in [3].

[2] Thus eliminating the usual runtime spikes in the transition area. For ReSumEr, such spikes are also small in the right panels of Fig. 2 and 3, however, here the reason clearly lies in ReSumEr's laborious 'iterative' approach for proving unsatisfiable subsumption instances, as commented earlier.

4 ReCovEr: a restart-based coverage estimator

We will now explain how to exploit a restarted strategy to obtain a maximum likelihood estimate of the proportion of examples subsumed by a pattern, without the need to complete a subsumption with a definitive (YES/NO) answer for any particular example.

We first need to make the assumption that given a pattern P and a set of examples E, the probability that Algorithm 1 finds a solution (i.e. returns YES as its answer) before it explores more than *cutoff* nodes of the search tree, is same for all $e \in E$ such that P subsumes e. Denote this probability by p. We will not directly assess the empirical accuracy of this assumption, but we will later verify it indirectly by testing the accuracy of the algorithm built upon it.

We assume a given pattern P and we fix a constant cutoff value R. In the first step, for each $e \in E$ we run $SubsumptionCheck(P, e)$ (Algorithm 1), stopping it as soon as the number of searched nodes has reached R. Then, after $|E|$ restarts (each time with a different $e \in E$), we can derive the probability that the algorithm has produced exactly m_1 'YES' responses in this first step. In particular, this probability $P(m_1)$ is

$$P(m_1) = \binom{A}{m_1} p^{m_1}(1-p)^{A-m_1} \tag{1}$$

where $A = |\{e \in E | P\theta \subseteq e\}|$. In the next step, all m_1 examples shown to be subsumed in the first step are removed from E and the procedure is repeated with the remaining examples. In general, we can derive the probability that exactly m_i YES answers are generated in the i-th step. Thus for $i = 2$, we obtain

$$P(m_2|m_1) = \binom{A-m_1}{m_2} p^{m_2}(1-p)^{A-m_1-m_2} \tag{2}$$

and similarly for an arbitrary $i \geq 1$, we have

$$P(m_i|m_{i-1}, \ldots, m_1) = \binom{A - \sum_{j=1}^{i-1} m_j}{m_i} p^{m_i}(1-p)^{A-\sum_{j=1}^{i} m_j} \tag{3}$$

The probability of a sequence $(m_1, \ldots, m_k)$, where m_i is the number of examples for which YES was produced in the i-th step, is given by

$$P(m_1, \ldots, m_k) = \prod_{i=1}^{k} P(m_i|m_{i-1}, \ldots, m_1) \tag{4}$$

Substituting for $P(m_i|m_{i-1}, \ldots, m_1)$ from Eq. 3 and taking the logarithm Eq. 4 results in

$$\ln\left(P(m_1, \ldots, m_k)\right) =$$

$$= \sum_{i=1}^{k} \ln \binom{A - \sum_{j=1}^{i-1} m_j}{m_i} + \sum_{i=1}^{k} m_i \ln p + \sum_{i=1}^{k} \left(A - \sum_{j=1}^{i} m_j\right) \ln(1-p) \tag{5}$$

Algorithm 6 *ReCovEr*(P, E, R, M, Δ): Algorithm for coverage estimation

Input: Pattern P and set of examples E, Integers R ('cutoff'), M, Δ;

$tries \leftarrow 0$
$Unknown \leftarrow Examples$
$CoveredInIthTry \leftarrow []$
repeat
  $tries \leftarrow tries + 1$
  $CoveredInThisTry \leftarrow 0$
  for $\forall E \in Unknown$ **do**
    $Answer \leftarrow$ Run $SubsumptionCheck(P, E)$ with number of searched nodes limited to R
    if $Answer = PositiveMatching$ **then**
      $CoveredInThisTry \leftarrow CoveredInThisTry + 1$
      $Unknown \leftarrow Unknown \backslash E$
    end if
  end for
  $CoveredInIthTry[tries] \leftarrow CoveredInThisTry$
until $tries \geq M \wedge \|LikelihoodEstimate(tries - 1) - LikelihoodEstimate(tries)\| \leq \Delta$
return $LikelihoodEstimate(tries)$

To find the parameters A and p for which $P(m_1, \ldots, m_k)$ is maximized, we take the partial derivative of Eq. 5 with respect to p and then find its roots, yielding

$$p = \frac{\sum_{i=1}^{k} m_i}{\sum_{i=1}^{k} m_i + \sum_{i=1}^{k} \left(A - \sum_{j=1}^{i} m_j\right)} \tag{6}$$

Finding the global maximum of $P(m_1, \ldots, m_k)$ from Eq. 4 on the set

$$D = \{(A, p) | A \in \{1, 2, \ldots, |E|\} \wedge p \in [0; 1]\} \tag{7}$$

is now straightforward, since using (6) we can find the maximum on every line

$$L_i = \{(i, p) | p \in [0; 1]\} \tag{8}$$

The maximum on line L_i is located either at the value of p given by (6) or at one of the borders of L_i. It then suffices to evaluate (4) at these three points of L_i for every i ($1 \leq i \leq |E|$). The estimate of A then equals the index i of the L_i on which the maximum is located.

The described estimator is used in ReCovEr (Algorithm 6). The question of how to choose k, i.e. how long a sequence $(m_1, \ldots, m_k)$ should be generated as the input to the estimator, is tackled iteratively: the sequence is being extended until two subsequent estimates differ by less than some Δ, specified as a parameter. A minimum length M of the sequence is however imposed, to avoid premature estimates coinciding by chance.

For purposes of initial empirical assessment of ReCovEr, we generated patterns and examples by Algorithm 3 with $p = 0.4$ for patterns and $p = 0.3$ for examples. Figure 4 demonstrates that ReCovEr produces estimates of acceptable accuracy. Certain imprecision indeed can be accepted: the coverage estimate should be accurate to the extent allowing to establish a reliable ranking of candidate hypotheses. Apart from that, the actual coverage value is seldom of interest.

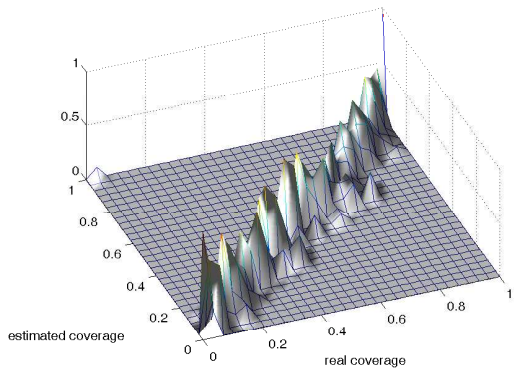


Fig. 4. Precision of ReCovEr (Algorithm 6) estimates reflected as the joint distribution of 1000 pairs [estimated, real], for $R = 100$, $M = 6$ and $\Delta = 1$. Patterns and examples were generated by Algorithm 3 with $p = 0.4$ for patterns and $p = 0.3$ for examples. Patterns have 10 vertices, examples have 100 vertices. The 1000 estimates correspond to 1000 different hypotheses tested on a pre-fixed set of 100 examples.

Algorithm	**Avg. Time [s]**
ReCovEr	20.2
ReSumEr	41.7
Django	45.9

Table 1. Average coverage test runtimes for the configuration from Fig. 4.

Table 1 shows the average runtime of ReCovEr testing one pattern on 100 examples in the same experimental configuration as in Fig. 4. For comparison, the table also shows the analogous runtimes needed to compute the coverage by testing subsumption for each $e \in E$ by ReSumEr or Django. While the runtime benefit provided by ReCovEr does not appear particularly significant from this table, it must be noted that the experimental parameters (detailed in the caption of Fig. 4) chosen for this first assessment correspond to areas where both ReSumEr and Django perform well. It is our expectation that ReCovEr will outperform much more significantly both Django in the YES domain and ReSumEr in the NO domain. In the former case, the expectation is based on the fact that in the YES region, Django exhibits heavy tails which are, by construction, prevented by ReCovEr. In the latter case, ReSumEr conducts an expensive iterative approach for showing unsatisfiability of subsumption, which becomes a crucial factor in the NO region. On the contrary, ReCovEr prevents

this burden because it does not need to complete the subsumption check for any unsatisfiable subsumption instance.

As a last remark, we did not compare ReCovEr to the sampling method from [5]. Although [5] alleviates coverage computation by taking only a small sample of E, subsumption for any single e from that sample is tested in the standard Prolog framework. Such an approach exceeds measurable runtimes for the sizes of e considered in this paper.

5 Conclusions and Future Work

We have demonstrated the benefits of using a complete restarted randomized algorithm ReSumEr for subsumption testing, a procedure at heart of most relational data mining systems. We have further introduced ReCovEr, an algorithm exploiting restarts for a maximum-likelihood based estimation of pattern coverage, eliminating the needed to prove subsumption for any particular example; the set of examples for which subsumption is actually proved during the application of ReCovEr is a result of a random process. ReCovEr prevents heavy tails as well as laborious proving of unsatisfiable subsumption instances. All of our experimental evaluations were constrained to generated data in the form of oriented colored graphs (uniform and scale-free, respectively). However, the principles behind ReSumEr and ReCovEr do not rely on this representation bias. Our next work will therefore concentrate on tests with a more general structure representation language and with real-life relational data mining benchmarks.

Acknowledgements
We thank the anonymous MRDM'07 reviewers for their helpful remarks. The first author is supported by the Czech Academy of Sciences through the project 1ET101210513 Relational Machine Learning for Biomedical Data Analysis. The second author is supported by the European Commission through the project FP6-027473 SEVENPRO.

References

1. H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 408–421. Springer-Verlag, 2001.
2. C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.
3. J. Maloberti and M. Sebag. Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning*, 55(2):137–174, 2004.
4. M. Sebag and C. Rouveirol. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1997.
5. A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.
6. F. Zelezny, A. Srinivasan, and D. Page. Randomised restarted search in ILP. *Machine Learning*, 64(1–2):183–208, 2006.

Relational Transformation-based Tagging for Human Activity Recognition

Niels Landwehr[1], Bernd Gutmann[1], Ingo Thon[1], Matthai Philipose[2], and Luc De Raedt[1]

[1] Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200 A, B-3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be
[2] Intel Research Seattle
1100 NE 45th Street
Seattle, WA 98105, USA
matthai.philipose@intel.com

Abstract. The ability to recognize human activities from sensory information is essential for developing the next generation of smart devices. Many human activity recognition tasks are — from a machine learning perspective — quite similar to tagging tasks in natural language processing. Motivated by this similarity, we develop a relational transformation-based tagging system based on inductive logic programming principles, which is able to cope with expressive relational representations as well as a background theory. The approach is experimentally evaluated on two activity recognition tasks and compared to Hidden Markov Models, one of the most popular and successful approaches for tagging.

1 Introduction

Smart systems that assist humans must be able to recognize the current context of the user and the activity she is performing in order to suggest or take actions in an intelligent manner. To recognize the context and activity, such systems can rely on streams of past activities, context, and sensory information (visual, object-interaction, ...). Recognizing the current activity or context then corresponds to inferring the activity or context from such sequential information. From a machine learning perspective, this task is akin to many tagging tasks pursued in natural language processing. For instance, in part-of-speech tagging, a form of "shallow parsing", the words in a sentence are to be labeled with the corresponding parts-of-speech (word categories). Many techniques have been developed and employed for this purpose. Two popular techniques for part-of-speech tagging are Hidden Markov Models and transformation-based learning [1]. However, whereas Hidden Markov models have been applied in many different areas, ranging from speech-recognition to activity recognition and bio-informatics, to the best of the authors' knowledge, transformation based learning has only seldomly been applied outside the field of natural language processing.

Because the structure of natural language is quite rigid as compared to that of typical activity recognition tasks, the existing transformation-based learners cannot directly

be applied for activity recognition. Therefore, we develop a more flexible *relational* transformation-based tagger within the inductive logic programming paradigm. This does not only provide an *expressive* representation but also allows one to easily incorporate background theory during the learning process. Thus the key contribution of this paper is a relational extension of transformation-based tagging based upon inductive logic programming principles. It also extends earlier work on relational transformation-based learning by [2] in that it focuses on *tagging* rather than *classification*. More specifically, from inductive logic programming (and the work by [2]) our technique inherits its search and refinement techniques (including a branch-and-bound algorithm) and from transformation-based learning the error driven stacking of rules.

The proposed method is evaluated in two activity recognition domains: "Activities of Daily Living" (ADL) recognition from a stream of "object interaction" data [6], and mobile phone profile prediction based on data collected by [8]. Experiments show that obtained tagging accuracies are competitive with those of HMM-based approaches, and it is easy to incorporate human-supplied background knowledge into the learning process. Furthermore, and that is perhaps the key advantage of the relational transformation-based tagger, the method can easily be extended to deal with variants of the tagging problem, for instance the prediction of structured output tags (as in Logical Hidden Markov Models [4]), and to cope with rich background knowledge.

2 Sequence Tagging

Sequence tagging is the task of assigning to each element in a given sequence an appropriate label or *tag*. Let $W = \{w^1, ..., w^k\}$ denote the vocabulary of sequence elements, and $T = \{t^1, ..., t^m\}$ the vocabulary of tags. The most prominent instance of the tagging problem is part-of-speech-tagging in natural language processing, where the task is to assign lexical categories $t \in T$ to words $w \in W$ in a given natural language sentence. Many other interesting sequence analysis problems can be cast in this framework, such as activity recognition in user modeling or gene finding and protein secondary structure prediction in bioinformatics.

In NLP, the two most common tagging approaches are transformation-based taggers (rule-based) and probabilistic methods (hidden Markov models or related techniques). Both of these approaches yield competitive results, and have received much attention. Before discussing our extension to transformation-based learning, we briefly review these two approaches in the next two sections.

2.1 Transformation-based Tagging

Transformation-based learning is a rule-based learning approach which iteratively stacks rules on top of each other to improve performance [1]. The basic transformation-based learning algorithm for the tagging problem is summarized in Algorithm 1. The algorithm takes as input a set S of sequences with known true tags L. During learning, it maintains a set of current tags $\hat{L}$ for all $s \in S$. $\hat{L}$ is initialized with some simple scheme, such as assigning to every element $w \in W$ its most common tag $t \in T$ in the training data (procedure *initial-tags*). The algorithm then tries to improve the current

Algorithm 1 Basic transformation-based tagging algorithm.

tb-tagging(input: sequences S; true sequence tags L)

```
1  L̂ := initial-tags(S, L)
2  initialize R = []
3  repeat
4        r := find-best-rule(S, L̂, L)
5        update L̂ := apply-rule(L̂, r)
6        update R := append(R, r)
7    until (no improvement)
8  return R
```

tagging $\hat{L}$ with respect to the true tagging L by learning a list of *transformation rules* R. Transformation rules can re-tag sequence elements based on the context they appear in. A transformation rule has the form $t' \leftarrow t : context$ and simultaneously replaces all occurrences of tag t in all sequences with t' whenever the constraint $context$ is satisfied.

Example 1. As an example from NLP, the word "move" could be initially tagged as "verb", but would be re-tagged as "noun" if the preceding word was tagged as "article". This can be encoded by the following transformation rule:

$$\textit{noun} \leftarrow \textit{verb} : \text{word} = \textit{move}, \text{preceding tag} = \textit{article}$$

The transformation rule languages employed in traditional transformation-based tagging are mostly simple instantiations of some template—for instance, querying in $context$ the word and tag at the current position and the next or preceeding position(s). We will replace this constraint by a first-order logical expression in Section 3.

In every iteration, the transformation rule which yields the greatest reduction in error between $\hat{L}$ and L is greedily selected (*find-best-rule*), applied to the current tagging $\hat{L}$ and appended to the rule list R. As conditions of rules in R match not only sequence elements but also currently predicted tags $\hat{L}$, rules can effectively bootstrap the current predictions. This makes transformation-based learning strictly more powerful than standard rule learning [1].

2.2 Hidden Markov Model Tagging

Tagging with hidden Markov models is typically performed with a model in which there is a hidden state q_t for every possible tag t, and state emission symbols correspond to symbols $w \in W$. That is, the observed sequence of symbols is seen as being generated by the hidden sequence of tags. Formally, the joint probability of an observation sequence $s = w_1 ... w_n$ with hidden tag sequence $t_1 ... t_n$ is given by

$$P(w_1 ... w_n, t_1 ... t_n) = P(t_1) \prod_{i=1}^{n-1} P(t_{i+1} \mid t_i) P(w_i \mid t_i)$$

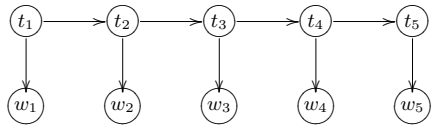


Fig. 1. Example lattice generated by unrolling a tagging HMM to a sequence $w_1, ..., w_5$. Inference in this model is carried out with the Viterbi algorithm, which yields the most likely joint state of the hidden variables $t_1, ..., t_5$ given the observations on $w_1, ..., w_5$.

where $P(t_1)$ is an initial probability for tag t_1 and $P(w_i \mid t_i)$, $P(t_i \mid t_{i-1})$ are conditional probabilities for the emitted word w_i and next tag t_{i+1} given the current tag t_i. When such a model is applied to a sequence $w_1...w_n$, it is unrolled into a lattice as depicted in Figure 1, and the Viterbi algorithm [7] is employed to efficiently compute

$$\begin{aligned}\hat{t}_1...\hat{t}_n &= \arg\max_{t_1...t_n} P(t_1...t_n, w_1...w_n) \\ &= \arg\max_{t_1...t_n} P(t_1...t_n \mid w_1...w_n),\end{aligned}$$

the most likely sequence of tags for the given sequence.

This technique has been used successfully for tagging problems in many domains. For instance, HMM-based approaches are a popular technique for inferring hidden user activities from a stream of object-interaction data in the so-called ADL ("Activities of Daily Living") problem [6, 10], which will be described in more detail below.

3 Relational Transformation-based Tagging

The general motivation for our work on relational transformation-based tagging is to apply the transformation-based tagging methodology to complex datastreams, which are generated for instance by sensors or sensor networks in ubiquitous computing environments. For such complex domains it is not always possible to represent all available information as flat (or *propositional*) symbols from a fixed alphabet. This problem can be overcome by using a more expressive *relational* representation for sequence elements. We will therefore extend the template-based rule language traditionally used in transformation-based learning to a more flexible *relational* rule language, which can take advantage of such richer representations for sequence elements. Furthermore, it is easy in this case to incorporate domain-specific background knowledge into the learning process. Analyzing such relational sequences has received considerable attention recently, for instance with relational extensions of Hidden Markov Models [4] or n-gram models [5].

Example 2. As an example, consider the ADL ("Activities of Daily Living") recognition problem, which is visualized in Figure 2. In ADL recognition, objects which are used in activities of daily living such as making breakfast are equipped with small RFID tags that can be picked up by a wearable reader while a person performs an activity [6].

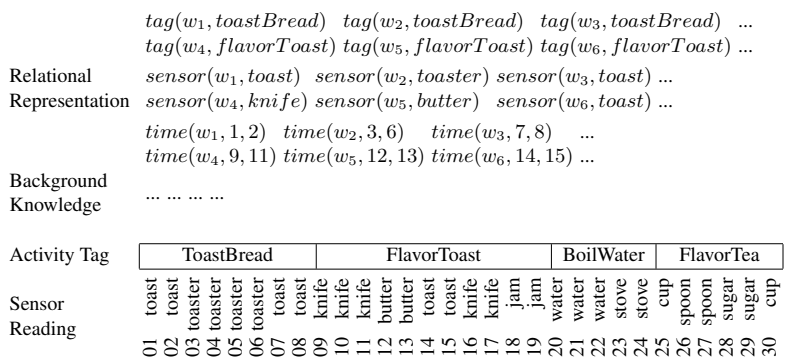


Fig. 2. Relational representation of the ADL recognition problem. While a person is performing activities of daily living (such as preparing breakfast), a stream of object interaction data is generated from a wearable RFID reader ("sensor reading"). This can be represented in a relational form by collapsing identical sensor readings to one sequence element w_i, and encoding the starting point and duration of the observation in another predicate. Furthermore, additional background knowledge can be used to encode prior knowledge about the domain.

The task is to recover the activity currently performed from the stream of sensor data, that is, to tag the sequence of object interactions with activities.

It is obvious that this kind of data is less rigidly structured than natural language data: there are no "grammatical rules" which determine the exact sequence of touching knife, toast, butter and jam when adding flavor to a toast. Nevertheless, context information can help determine the right tag. For instance, using a spoon can indicate activities FlavorTea or EatCereals. This ambiguity can be resolved by looking at the context: the observation of a spoon closely followed by sugar indicates activity FlavorTea, while observation of a spoon after milk and cereals indicates activity EatCereals.

Furthermore, the stream of object data obtained from the sensor has some internal structure, as an object observation has a starting point and duration in time. A representation in first-order logic allows to capture this structure, and to express flexible rule conditions such as *object x has (not) been observed less than t seconds before/after the current time-step* or *the most frequent (currently estimated) tag around the current time-step is t* using manually defined background knowledge.

At the same time, activity recognition can be seen as a *data stream mining* task—the analysis of a continuous, potentially infinite stream of data. In this context, issues such as online learning (with only one pass through the data necessary) are of considerable interest. However, we will not address these issues in the paper, and instead assume that a limited amount of training data is given a priori. Extending the proposed methods to an online-learning scenario is an interesting direction for future work.

The next section will discuss the formal learning setting for relational transformation-based tagging, before discussing learning algorithms and experimental results.

3.1 Learning Setting

The learning setting for relational transformation-based tagging can be formalized as follows:

Given

- a relational language $\mathcal{W}$ for describing sequence elements, i.e., a set of typed first-order logical predicates
- a set of tags T;
- a set of training sequences $S = \{s_1, ..., s_m\}$ with sequence elements described in $\mathcal{W}$ and corresponding true tags L over T;
- a scheme for setting initial tags given by a function $init$;
- a language $\mathcal{L}$ of transformation rules $t' \leftarrow t : q$ where $t, t' \in T$, $q = l_1, ..., l_r$ and the l_i are atoms in $\mathcal{W}$.

Find an ordered lists of transformations $R = [R_1, ..., R_l]$, $R_i \in \mathcal{L}$, such that applying the initial tagging scheme and afterwards transformation rules $R_1, ..., R_l$ minimizes

$$error(\hat{L}) = \sum_{s \in S} \sum_{i=1}^{n_s} \delta(l_{is}, \hat{l}_{is})$$

where n_s is the length of sequence s and l_{is}, $\hat{l}_{is}$ denote the tag assigned to element i in sequence s according to L and $\hat{L}$.

In contrast to standard (propositional) transformation-based tagging approaches, the languages $\mathcal{W}$ (sequence elements) and $\mathcal{L}$ (rules) employed are relational; that is, rule conditions q are first-order queries of the form $l_1, ..., l_k$ where the l_i are first-order logical atoms. Applying a first-order transformation rule $t' \leftarrow t : q$ means simultaneously replacing all tags t in $\hat{L}$ by t' wherever the first-order context constraint q matches the relational description of the corresponding sequence element.

Example 3. As an example for a relational transformation rule in the ADL recognition domain consider

$$\begin{aligned} &FlavorTea \leftarrow EatCereals : \\ &\quad sensor(X, spoon), near(X, sugar, 10), not(near(X, bowl, 5)) \end{aligned}$$

where the variable X is bound to the sequence element under consideration and the background predicate $near/3$ is defined by

$$\begin{aligned} &near(X, O, T) \leftarrow \\ &\quad time(X, S, D), sensor(X', O), time(X', S', D'), dist(S, D, S', D', T'), T' \leq T \end{aligned}$$

and $dist(S, D, S', D', T)$ measures the distance between the intervals $[S, S + D]$ and $[S', S' + D']$. This rule re-tags objects of type spoon from $EatCereals$ to $FlavorTea$ if implied by the context.

3.2 A Branch-and-Bound Learning Algorithm

For learning the list R of relational transformation rules, a large space of possible rules has to be searched. However, structure on the search space can be exploited to make this search more efficient. More specifically, the algorithm we use combines ideas from transformation-based learning (branch-and-bound search based on upper bounds for the error reduction of a transformation rule) and inductive logic programming (refinement search in a generalization/specialization lattice). It is closely related to the algorithm presented in [2].

Recall that the goal of learning is to find a list R of transformation rules which minimize $error(\hat{L})$ on a set of training sequences S with known true labels L. As in propositional transformation-based learning [1], the rule list is learned greedily: starting with an empty list, the algorithm incrementally adds one rule after the other, at every step selecting the rule which yields the greatest reduction in $error(\hat{L})$ and updating the current tagging $\hat{L}$ (cf. Algorithm 1).

When searching for an individual rule with maximum error reduction, a significant part of the search space can be pruned away by computing upper bounds for the error reduction a rule can achieve. One obvious bound for the reduction achievable by a transformation rule $t^i \leftarrow t^j : context$ is given by the number of sequence elements whose true tag (in L) is t^i and which are currently (in $\hat{L}$) assigned tag t^j. Let $\mathcal{M}$ denote the current confusion matrix, i.e., $\mathcal{M}[i,j]$ denote the number of sequence elements with true tag t^i currently tagged as t^j. This can be exploited by considering rules $t^i \leftarrow t^j : context$ in (decreasing) order of their potential $M[i,j]$ for error reduction and keeping track of the best error reduction Δ_{best} found so far. Now, all rules of the form $t^i \leftarrow t^j : context$ for which $\mathcal{M}[i,j] \leq \Delta_{best}$ can be removed from consideration (cf. [1]).

This idea can be taken one step further if it is combined with a general-to-specific search for the first-order constraint $context$ [2]. As a complete search in the space of first-order constraints is infeasible in most cases, we perform a greedy general-to-specific search. To generate the specializations of the current condition q, a so-called refinement operator ρ under θ-subsumption is employed. A conditions q_1 θ-subsumes a condition q_2 if and only if there is a substitution θ such that $q_1\theta \subseteq q_2$. A substitution is a set $\{V_1/t_1, \ldots, V_l/t_l\}$ where the V_i are different variables and the t_i are terms, and the application of the substitution replaces the variables $V_1, \ldots, V_l$ by the corresponding terms $t_1, \ldots, t_l$. $\rho(q)$ typically returns all minimal specializations of q within $\mathcal{L}$. For our purposes, the refinement operator specializes a condition $q = l_1, \cdots, l_n$ simply by adding a new literal l to the clause yielding $h \leftarrow l_1, \cdots, l_n, l$. This operator is monotone in the sense that for $q' \in \rho(q)$ the number of matches in the data can only decrease. Consequently, the maximum gain achievable from specializations of a transformation rule $t^i \leftarrow t^j : q$ can be bounded in terms of the current matches. More specifically, assume that a constraint q matches on a number of sequence elements in the training data S, and that for p_q of these it has a positive effect (current tag is t^j, but true tag is t^i) and for n_q it has a negative effect (current and true tag are t^j). The error reduction of applying the transformation $t^i \leftarrow t^j : q$ is $\Delta_q = p_q - n_q$. It is now obvious that no specialization $t^i \leftarrow t^j : q'$ with $q' \in \rho^*(q)$ can achieve an error reduction greater than $\Gamma_q = p_q$.

Algorithm 2 Branch-and-bound algorithm for relational transformation-based tagging

rtb-tagging(input: sequences S; true sequence tags L; language bias $\mathcal{L}$)

```
1  L̂ := initial-tags(S, L)
2  initialize R := []
3  repeat
4        initialize Δ_best := 0
5        compute M := confusion-matrix(L̂, L)
6        for all i, j ∈ {1, ..., k}, i ≠ j, sorted by M[i, j] descending do
7              initialize Γ := M[i, j]
8              initialize q := true
9              while (Γ > Δ_best) do
10                   for all q' ∈ ρ(q, L) do
11                         compute Δ_q' := error-reduction(t^j ← t^i : q')
12                         compute Γ_q' := max-reduction(t^j ← t^i : q')
13                   end for
14                   let q := argmax_q' Δ_q'
15                   let Δ_best := max(Δ_best, Δ_q)
16                   let Γ := Γ_q
17             end while
18       end for
19       let r := t^i ← t^j : q be a rule with error reduction Δ_best
20       update L̂ := apply-rule(L̂, r)
21       update R := append(R, r)
22   until (no improvement)
23 return R
```

A greedy branch-and-bound algorithm exploiting these two bounds is outlined in Algorithm 2. It takes as input a set of training sequences S, true sequence tags L, and the language bias $\mathcal{L}$. The algorithm starts with an empty rule list R and initial tags assigned in $\hat{L}$. Transformation rules are then greedily added to R, and their effect applied to the current tagging $\hat{L}$ (lines 3–21). Transformations are considered in order of decreasing $\mathcal{M}[i, j]$ (line 6). At every step of the search for a single transformation $t^i \leftarrow t^j : q$ (lines 6–18), the algorithm keeps track of the largest reduction Δ_{best} achieved by a rule so far. During refinements of the context constraint q (lines 9–17) a bound Γ_q for the maximum reduction that any specialization of a rule q can still achieve is computed (max-reduction), and only parts of the search space for which Γ is greater than Δ_{best} are explored.

4 Experiments

The proposed method was implemented in the RETRO (for RElatational TRansfOrmation-based tagging) system and experimentally evaluated in two real-world domains: Activity of Daily Living recognition (**ADL**) and mobile phone profile prediction (**Phone**).

Table 1. Example relations used to describe the activity data. Some relations are directly derived from the data (e.g. $sensor$, $duration$, $close$), others include human-supplied prior knowledge (e.g. $close\_used$).

Relation	Description
$sensor(Id, Object)$	The object observed at sequence element Id is $Object$
$duration(Id, T)$	The object observation at sequence element Id lasted T seconds
$close(Id, Obj, T)$	The object Obj has been observed within T seconds of sequence element Id
$time\_bin(T, Bin)$	The time span T falls into the bin $Bin \in \{short, medium, long\}$
$closest\_tag(Id, Act)$	The closest sequence position to Id for which an activity (i.e., a tag $\neq$ "no activity") is assigned in $\hat{L}$ is tagged with Act
$close\_used(Id, Act, T)$	Less than T seconds away from sequence element Id an object has been observed which is typically used in Act

Relational Representation

$cell(w_1, 6672)\ cell(w_2, 6671)\ cell(w_3, 6673)$ ...
$time(w_1, 1, 15)\ time(w_2, 16, 25)\ time(w_3, 26, 38)$ ...
$usr\_activity(w_1, act)\ usr\_activity(w_2, idle)\ usr\_activity(w_3, act)$ ...
$active\_app(w_1, 101)\ active\_app(w_1, 102)\ active\_app(w_3, 101)$ ...
$comm(125, sms, incoming)\ comm(390, call, outgoing)$ ... ...

Phone profile	normal		silent			normal		meeting	
Cell	6672	6671	6673	7409	6673	6671	7409	7410	6739

Fig. 3. Illustration of the Phone data (predicates for cell location, duration, user activity, active applications, and communication events).

In the ADL recognition domain, object-interaction data for a user having breakfast at home has been gathered by a wearable RFID reader and RFID tags on objects such as milk, cereals, kettle, water tap, cutlery etc. (23 objects in total). The stream of tags picked up by the RFID reader indicates which object is close (approximately 10–15 centimeters) to the wrist of the user at a particular point in time. A single object observation is returned at every second—if several tags are within reach, one is returned randomly. Note that the data is relatively noisy: tags might sometimes be missed, or a tag not related to a particular activity can be reported by the reader because the corresponding object is accidentally close. The task is to predict the current activity performed, out of a set of 24 possible activities such as boiling water, toasting bread, reading a newspaper or "no activity". The sequence data obtained from the RFID reader is represented in a relational form by collapsing identical observations into one observation with a starting point and duration in time (cf. Figure 2 for an illustration). Furthermore, additional background predicates have been defined, see Table 1 for examples.

In the Context Phone domain, data about user communication behavior has been gathered using a software running on Nokia Smartphones. The software automatically logs communication and context data, such as the current provider cell, incoming and outgoing calls and text messages, and other phone status information. The task is to

Table 2. Average F-measure on the ADL Recognition and Phone problems based on a leave-one-sequence-out cross-validation.

Algorithm	**ADL**	**Phone**
Majority tag	19.5 ± 22.3	56.7 ± 13.1
HMM Tagger	74.9 ± 12.5	56.7 ± 13.1
RETRO	75.4 ± 7.8	67.7 ± 10.3

Table 3. Examples for rules learned by RETRO on the ADL dataset.

Learned Rules
$ObtainNewspaper \leftarrow ReadNewspaper$: $close(Id, Obj, T), Obj = door,$ $time\_bin(T, medium)$
$FlavorTea \leftarrow EatCereals$: $closest\_tag(A, FlavorTea)$
$SteepTeaBag \leftarrow DrinkTea$: $close(Id, Obj, T), Obj = stove$
$PourCereal \leftarrow ObtainNewspaper$: $close\_used(Id, PourCereal, T),$ $not(close\_used(Id, ObtainNewspaper, T')), time\_bin(T, short)$
$SteepTeaBag \leftarrow noActivity$: $duration(Id, T), time\_bin(T, long),$ $closest\_tag(ID, SteepTeaBag)$

predict the active profile of the phone (silent, meeting, or normal) at every point in time. See Figure 3 for an illustration of the data and the predicates used.

For comparison, we have also conducted experiments with a (propositional) HMM tagger on the two datasets. As it is not possible to encode all relevant information propositionally, we have selected the most relevant information to be used as the propositional alphabet W. For the ADL recognition problem, this is the sequence of objects observed.

For the phone domain, it is the sequence of cells the phone was located in.

For initializing the tagging $\hat{L}$ in the transformation-based tagger, RETRO simply assigns the most frequent tag given the propositional symbol $w \in W$:

$$\text{init}(w) = \underset{t \in T}{\text{argmax}}\, C(w, t)$$

where $C(w, t)$ is the number of times symbol w was tagged with t in the training data. More elaborate initialization schemes (such as using the HMM tagging as an initialization for the transformation-based tagger) are an interesting direction for future work. Furthermore, instead of a simple greedy search as outlined in Algorithm 2, a beam search with beam size $K = 10$ is used. The main loop of the algorithm is terminated if no rule with a gain of at least $min\_gain = 10$ is found.

Table 2 lists the average F-measure for RETRO and HMM tagging based on a leave-one-sequence-out cross-validation. For the ADL recognition problem, there is no significant difference between the two approaches. In the phone domain, the HMM tagger fails to improve upon the majority tag prediction, while RETRO yields a (borderline) significant increase in F-measure (paired sampled t-test, $p = 0.051$). This shows that transformation-based approaches can be competitive with probabilistic methods in complex tagging domains. However, the presented experiments are still preliminary, and more empirical evaluation is needed to assess the potential of the method in more

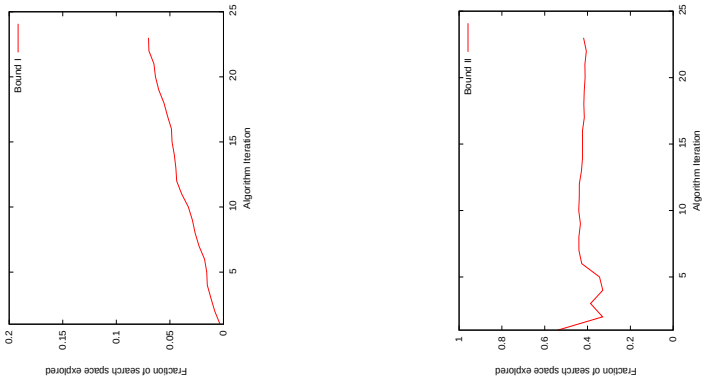


Fig. 4. Effectiveness of the two pruning schemes Bound I (maximum gain attainable from changing a certain tag into a certain other tag) and Bound II (maximum gain attainable from specializing a given rule). Results are averaged over a leave-one-sequence-out cross-validation.

detail. Note furthermore that although HMM tagging is a standard approach in activity recognition, more advanced probabilistic methods have recently been developed which would possibly yield slightly higher accuracy in this domain [9].

Examples for rules learned by RETRO on the ADL recognition task are shown in Table 3. For instance, consider the last rule: it encodes that if a sequence element corresponding to a long object observation is tagged with $noActivity$ and the closest currently predicted activity is $SteepTeaBag$, this sequence element should also be tagged with $SteepTeaBag$. This rule is useful for "filling in gaps" as $SteepTeaBag$ only causes characteristic object observations at the beginning and end of the activity.

Finally, Figure 4 visualizes the effectiveness of the pruning schemes based on the two upper bounds discussed above on the ADL recognition problem. More specifically, Figure 4 (left) shows the fraction of pairs (t^i, t^j) that have to be considered when searching for rules $t^i \leftarrow t^j$ in lines 6–18 of Algorithm 2 as a function of the algorithm iteration. This pruning scheme is very effective, reducing the search space by 93%–99%. It is more effective in earlier iterations as it is easier to find a rule with yields a large reduction in error. Figure 4 (right) shows which fraction of refinements is removed from the beam when rules are refined in lines 10–13 of Algorithm 2 because no further specialization can reach the performance of the best rule found so far. Note that this form of pruning does not affect the computational complexity of the algorithm but rather allows a more thorough search through the space of possible rules (given a limited beam size) by effectively reducing the branching factor of the search. On average, the branching factor is about halfed, this is independent of the algorithm iteration.

5 Conclusions and Related Work

Motivated by the needs of activity recognition problems, we have introduced a relational transformation-based tagging system. It tightly integrates principles of inductive logic programming (especially search, representations, operators, background knowledge) with transformation-based tagging (error-driven search, branch-and-bound idea). The approach has been evaluated on two activity recognition data sets and the results are competitive with those of a Hidden Markov Model approach. Perhaps more important than the experimental results obtained so far is the ease with which one can extend the transformation-based tagging approach beyond the propositional HMM setting. Important directions in this regard include: the use of rich sources of background knowledge (that take not only into account the inputs but also the already available produced tags), the prediction of structured output sequences (predicting sequences of logical atoms, cf. [3], such as *call(anna,10)* denoting the prediction that *anna* will be called in 10 minutes), and relaxing the purely sequential nature of the output (which is important for the ADL dataset where different activities may overlap in time, and therefore ordering them is not always possible).

Acknowledgments We would like to acknowledge support for this work from the Research Foundation-Flanders (FWO-Vlaanderen).

References

1. E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.
2. L. Dehaspe and M. Forrier. Transformation-based learning meets frequent pattern discovery. In J. Cussens, editor, *Proceedings of the 1st Workshop on Learning Language in Logic*, pages 40–51, Bled, Slovenia, 1999.
3. K. Kersting, L. De Raedt, B. Gutmann, A. Karwath, and N. Landwehr. Relational sequence learning. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Application of Probabilistic ILP*. Springer, 2007. to appear.
4. K. Kersting, L. De Raedt, and T. Raiko. Logical hidden markov models. *Journal of Artificial Intelligence Research*, 25:425–456, 2006.
5. N. Landwehr and L. De Raedt. r-grams: Relational grams. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 907–912, Hyderabad, India, 2007.
6. D. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Proceedings of ISWC 2005*, Osaka, 2005.
7. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
8. M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen. ContextPhone - a Prototyping Platform for Context-aware Mobile Applications. *IEEE Pervasive Computing*, 4(2):51–59, 2006.
9. S. Wang, W. Pentney, A.-M. Popescu, T. Choudhury, and M. Philipose. Common sense based joint training of human activity recognizers. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2237–2242, 2007.
10. D. Wilson and M. Philipose. Maximum a posteriori path estimation with input trace perturbation: Algorithms and application to credible rating of human routines. In *Proceedings of IJCAI 2005*, Edinburgh, Scotland, August 2005.

Learning Ground CP-logic Theories by means of Bayesian Network Techniques

Wannes Meert, Jan Struyf and Hendrik Blockeel

Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200A, 3001 Leuven, Belgium,
{wannes.meert, jan.struyf, hendrik.blockeel}@cs.kuleuven.be

Abstract. Causal relationships are present in many application domains. CP-logic is a probabilistic modeling language that is especially designed to express such relationships. This paper investigates the learning of CP-theories from examples, and focusses on structure learning. The proposed approach is based on a transformation between CP-logic theories and Bayesian networks, that is, the method applies Bayesian network learning techniques to learn a CP-theory in the form of an equivalent Bayesian network. We propose a constrained refinement operator for such networks that guarantees equivalence to a valid CP-theory. We experimentally compare our method to a standard method for learning Bayesian networks. This shows that CP-theories can be learned more efficiently than Bayesian networks given that causal relationships are present in the domain.

1 Introduction

Bayesian networks have earned a good reputation for modeling complex problems involving uncertain knowledge. This success seems to be at least in part due to the causal interpretation that can be given to such networks. Although it is theoretically impossible to infer all causal relationships from a typical data set, it still seems useful to try to approximate them by means of learning, mainly because causal information tells you just enough about the (expected) behavior of a process, while allowing irrelevant details to be ignored [13]. This causality is, however, not present in the formal semantics of Bayesian networks (although there is some work on *causal Bayesian networks* [11]).

Causal Probabilistic logic (CP-logic), on the other hand, is a probabilistic modeling language with causality at the heart of its fundamental construct. Because CP-logic focusses on causal information, in a sense, it allows a more fine-grained and flexible representation than Bayesian networks [13]. This results in a smaller number of parameters and easy to read theories. This is not only useful for human experts, but also for automated model building algorithms. Indeed, a lot of effort goes into obtaining the structure of the model and the numerical parameters that are needed to fully quantify it. For example, in a Bayesian network, a binary variable with n parents will require 2^n parameters to be defined. For a large value of n this is not only a heavy task for a human

expert but it also requires sufficiently large data sets to accurately learn all these parameters [9, 3]. A CP-logic theory (CP-theory), on the other hand, has only one parameter for every *cause*.

There is already some work on learning a subset of CP-logic by Riguzzi [12][1]. This subset was extended by Blockeel and Meert [1] by using a conversion to Bayesian networks. The latter method offers the possibility to adapt some of the many methods and optimizations available for Bayesian networks to learn a CP-theory. Where Blockeel and Meert [1] mainly focus on parameter learning, in this paper, we offer a practical method for learning the structure of a large subset of CP-theories by using this conversion to Bayesian networks.

We start this article with a brief introduction to CP-logic in Section 2. In Section 3, we explain the link with Bayesian networks and how to convert a CP-theory to a Bayesian network. Finally, we adapt some Bayesian learning algorithms to learn a CP-theory from data in Section 4 and Section 5.

2 Intuitions about CP-logic

We briefly introduce CP-logic. A more formal description can be found in [13].

A CP-theory can be seen as a set of if-then rules of the following form:

$$(h_1 : \alpha_1) \vee (h_2 : \alpha_2) \vee \ldots \vee (h_n : \alpha_n) \leftarrow b_1, b_2, \ldots, b_m.$$

with h_i atoms and b_i literals in the logical sense, all *causal probabilities* $\alpha_i \in [0, 1]$, and $\sum_{i=1}^{n} \alpha_i \leq 1$. We call the set of all $(h_i : \alpha_i)$ the head of the rule, and the set of all b_i the body. If the head contains only one atom $h : 1$, we may write it as h.

Each of these rules has several possible conclusions (*head*), each of which has a certain probability (α_i) assigned to it. The rule makes exactly one of the conclusions true with the associated probability. For instance,

$$(spaghetti : 0.5) \vee (steak : 0.5) \leftarrow shops(john).$$

expresses that if John goes shopping, he buys spaghetti or steak for dinner each with 50% chance.

Multiple rules in a CP-theory may lead to the same conclusions. Take, for instance, the following CP-theory

$$(spaghetti : 0.5) \vee (steak : 0.5) \leftarrow shops(john).$$
$$(spaghetti : 0.3) \vee (fish : 0.7) \leftarrow shops(mary).$$

which expresses that if John goes shopping he buys spaghetti or steak for dinner and if his girlfriend Mary goes shopping, she buys spaghetti or fish. The numbers associated with the different meals indicate the probability that the event mentioned in the rule body causes that type of dinner to be bought. If both go

[1] Note that CP-logic theories and LPADs (Logic Programs with Annotated Disjuctions) are equivalent. The research about LPADs has evolved into CP-logic.

shopping it is possible that they buy both steak and fish, in that case they can choose what they will have for dinner.

It is part of the semantics of CP-logic that each rule *independently of all other rules* makes one of its head atoms true when triggered. CP-logic is therefore particularly suitable for describing models that contain a number of independent stochastic events or causal processes. Consequently, learning a CP-theory amounts to discovering the causal structure of possibly complex processes.

It may be tempting to interpret the parameters as the conditional probability of a head atom given the body, e.g., $Pr(spaghetti|shops(mary)) = 0.3$, but this is incorrect. The conditional probability that spaghetti is bought, given that Mary went shopping, is higher than 0.3, because there is a second possible cause, namely that John also bought spaghetti. To compute this conditional probability, we need information on the probability that John went shopping. For instance, with

$$\begin{gathered} shops(john) : 0.2. \\ shops(mary) : 0.9. \\ (spaghetti : 0.5) \vee (steak : 0.5) \leftarrow shops(john). \\ (spaghetti : 0.3) \vee (fish : 0.7) \leftarrow shops(mary). \end{gathered}$$

we can say that $Pr(spaghetti|shops(mary)) = 0.3 + 0.2 \cdot 0.5 \cdot 0.7 = 0.37$: Mary buys spaghetti with probability 0.3, but there is also a probability of 0.2 that John went to the shop, and hence a probability of $0.2 \cdot 0.5 \cdot 0.7$ that spaghetti is bought by John ($0.2 \cdot 0.5$) and not by Mary (0.7).

Thus, for head atoms that occur in multiple rules, the mathematical relationship between the CP-theory parameters and conditional probabilities is somewhat complex, but it is not unintuitive. The meaning of the probabilities in the rules is quite simple: they reflect the probability that the body *causes* the head to become true. This is different from the conditional probability that the head is true given the body, but among the two, *the former is the more natural one to express*. Indeed, the former is local knowledge: an expert can estimate the probability that $shops(mary)$ causes $spaghetti$ without considering any other possible causes for $spaghetti$. To infer $Pr(spaghetti|shops(mary))$, we need global knowledge: we need to know all possible causes for $spaghetti$, the probability of them occurring, and how they interact with $shops(mary)$.

The fact that the parameters in a CP-theory are local makes it impossible to estimate them directly from the data as can be done in Bayesian networks. In Section 3, we will see that CP-theory parameters can be mapped to Bayesian network parameters by introducing unobserved nodes.

3 Converting a CP-logic Theory to a Bayesian Network

Blockeel and Meert [1] show that any CP-theory that is non-recursive and has a finite Herbrand universe can be converted into a Bayesian network such that the

CP-theory parameters appear in the network's CPTs[2]. All other CPT entries are either 0.0 or 1.0. In this section, we explain this conversion.

We only consider ground CP-theories, so a non-ground CP-theory must be grounded first. Based on such a CP-theory we construct a Bayesian network. The structure can be created in three steps:

1. For every literal in the CP-theory, a Boolean variable is created in the Bayesian network. This is a so-called *literal variable* and is represented by a *literal node* in the network.
2. For every rule in the CP-theory, a *choice variable* is created in the Bayesian network. This variable can take $k+1$ values, where k is the number of atoms in the head. It is represented by a *choice node* in the network.
3. If an atom is in the head of a rule, an edge is created from its corresponding choice node towards the atom's node. If a literal is in the body of a rule, an edge is created from the literal node towards the rule's choice node.

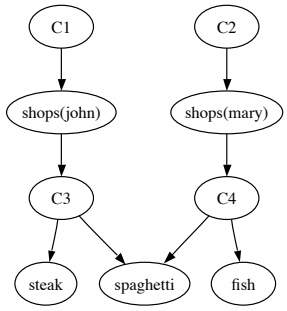
For the CPTs there are two cases:

1. The CPT of a *choice variable* (e.g., Fig. 1 CPT for C3): such a variable can take $k+1$ values with k the number of atoms in the head. The variable has the value i if the i^{th} atom from the head is chosen by the probabilistic process. If none is chosen (in case the probabilities do not sum up to one) then the variable has the value 0. The probability that the variable takes a particular value if the body is true, is the causal probability given in the rule of the CP-theory. If the body is not true, the probability that the choice variable takes the value 0 is 1.0 and all the other values have probability 0.0. Note, that a body may contain negative literals.
2. The CPT of a *literal variable* (e.g., Fig. 1 CPT for spaghetti) is structured differently. If the choice variable of a rule in which the atom is in the head has as value the position of that atom in the head, then it will be true with probability 1.0, otherwise it will be false.

As an example, the conversion of the shopping example to a Bayesian network is depicted in Fig. 1. It can be noted that this structure resembles noisy-or structures as introduced by Pearl [10]. This is partly due to the fact that CP-logic uses the principle of *independent causation*, which is similar to the principle of *independence of causal influence* (ICI). ICI models are a family of Bayesian network models used for classification tasks with large numbers of attributes. So, a simple CP-theory can be used as an ICI classifier.

Since we can convert a CP-theory to a specific type of Bayesian network, we have two different representations for the same CP-theory. To differentiate between them we will name them. All the possible CP-theories expressed in the CP-logic syntax and semantics will be called the *CP-logic space*. The Bayesian networks resulting from the conversion are part of what we will call the *Bayesian network space*. So, this is the space of all the possible Bayesian networks that are equivalent to a valid CP-theory.

[2] CPTs are Conditional Probability Tables as known for Bayesian Networks. Note that Conditional Probability is not the same as Causal Probability (Section 2).



(a) Bayesian network

C1	
0	0.8
1	0.2

john(shops)	C1=1	other
T	1	0
F	0	1

C3	john(shops)	other
0	0	1
1	0.5	0
2	0.5	0

spaghetti	C3=1 or C4=1	other
T	1	0
F	0	1

(b) CPTs for some nodes of (a)

Fig. 1. Conversion from a CP-theory to a Bayesian network.

4 Parameter Learning

Once this equivalent Bayesian network is created we can use known methods for learning Bayesian networks to learn the parameters of the network and thus also the CP-theory parameters. It is, however, necessary to impose constraints on the CPTs in order to keep the network equivalent to a valid CP-theory. As input we use a multi-set of tuples and each tuple indicates the truth value (true or false) of each literal in the domain.

The choice variables in the Bayesian network are not present in our domain, therefore the network is not fully observable. This is solved by using an EM-algorithm for learning the parameters [8]. Because we use an EM-algorithm, the missing values are allowed in the input data.

While the choice variables introduce unobserved variables, which is a disadvantage, the structure of the CP-theory gives extra information about the structure of the CPTs, which can make the learning more efficient. Many of the parameters are 0.0 or 1.0 (Fig. 1). These values are known in advance and don't need to be learned, to the contrary, they can be used to speed up the EM-algorithm. Only the values that are not 0.0 or 1.0 have to be learned; these are the original parameters of the CP-theory. To force the CPTs to have ones and zeros in the right positions after learning, it suffices to initialize these parameters with 0.0 or 1.0. The Bayesian update rule in the expectation-step can only update values strictly between 0 and 1. Since the prior probability is set, e.g., to 0.0 the posterior probability is also 0.0. So, a standard EM parameter learning algorithm for Bayesian networks learns the correct parameters for our CP-theory.

The method outlined above is a simple application of the EM-algorithm. This can, however, be further optimized. The CPTs that only contain probabilities 0.0 or 1.0 are actually not describing a probability distribution but a *functional dependency*. These are rather functions where there is just one result and it is

calculated based on the input, not a set of results with each a probability. Since this network contains many functional dependencies, extra optimizations can be incorporated into the algorithm. For example, an optimization specific for noisy-or implementations is given by Vomlel [14].

5 Structure Learning

Besides learning the parameters, we also need to learn the structure of a CP-theory. This involves a search over possible CP-theories, which can be done either in CP-logic space or in Bayesian network space. In the previous section, we learned the parameters in the Bayesian network space. Here we consider learning the structure also in the Bayesian network space. This avoids the conversion each time the algorithm investigates a new candidate structure. An important constraint when searching the Bayesian network space for possible structures is that every Bayesian network must have a valid mapping to a CP-theory.

The learning algorithm that we propose is based on the *structural EM-algorithm* (SEM) introduced by Friedman [2]. This is a greedy search algorithm, outlined in Table 1 for both the CP-logic space and the Bayesian network space. The *refinement* function returns a Bayesian network in the neighborhood of the current network. The *eval* function evaluates the new network.

Table 1. Greedy SEM algorithm to learn the structure of a CP-theory.

CPtheory := ∅	BN := initial Bayesian net
while CPtheory is not good enough:	**while** BN is not good enough:
S := refinements(CPtheory)	S := refinements(BN)
CPtheory := $\text{argmax}_{L \in S}$ eval(L)	BN := $\text{argmax}_{L \in S}$ eval(L)
return CPtheory	**return** CPtheory(BN)

5.1 Evaluation Function

The evaluation function is typically based on the likelihood of the data given the candidate model after learning the parameters. In this case, the *Bayesian Information Criterion* (BIC) is used [8]. The main advantage of this measure is its modularity. Every node in the network has its own local score and the sum of these scores is the total score. In this way we only have to recalculate the score of the part of the new structure that has changed with respect to the previous structure to see if it is better or not. As the BIC function prefers smaller networks, our algorithm prefers smaller CP-theories.

5.2 Refinement Function

The refinement operators are similar to those in the SEM-algorithm and perform a greedy hill-climbing neighborhood search. More concrete, the SEM operators

add, delete or invert one edge of the current network. Our algorithm takes a similar approach, but takes into account the specific structure of a Bayesian network that represents a CP-theory.

As we have seen previously, the choice nodes represent the rules in the CP-theory and the edges are defined by the literals in the head and the body. Based on this we introduce the following constraints on possible networks.

- Only edges between a literal node and a choice node are allowed.
- A literal node has at least one incoming edge.
- A literal node is a Boolean variable.
- A choice node is a variable that can take values from 0 to k with k the number of atoms in the head.
- The CPT of aliteral node contains only 0 or 1, based on the structure of the CP-theory.
- The CPT of a choice node has in the column where the body is true the CP-theory parameters. In all the other columns the choice node takes the value 0 with probability 1.

These constraints guarantee that the resulting Bayesian networks are equivalent to a CP-theory and all the entries in the CPTs except the CP-theory parameters are 1.0 or 0.0. There are three possible actions to find a network in the neighborhood of the current one: 1) adding a relation between two literals, 2) deleting a relation or 3) inverting a relation. A *relation* between two literals in this context means that there is a rule in the theory that has one of the literals in the head and the other one in the body. Translated to the Bayesian network space, this is the existence of a choice node with an incoming edge from one of the literals and an outgoing edge to the other literal. Fig. 2 gives an overview of the different refinement types, a detailed description follows next.

Deleting a Relation

Deleting an edge corresponds to deleting a literal from a rule. This can be done by removing the edge between a literal node and a choice node. If the choice node has no outgoing edges after the removal of the edge, the choice node itself is also removed. A graphical example can be seen in Fig. 2.a. In the first step in the figure it is shown that a rule can be eliminated by removing a choice node and its edges. The second step removes a literal from the body of a rule by eliminating an incoming edge of the choice node that represents that rule.

Adding a Relation

Adding an edge between a choice node and a literal node is the same as adding a literal to an existing rule in the CP-theory. To add an atom to the head of a rule, an edge is created departing from the choice node representing that rule to the literal node representing the added atom. Adding a literal to the body of a rule is accomplished by adding an edge from the literal node to

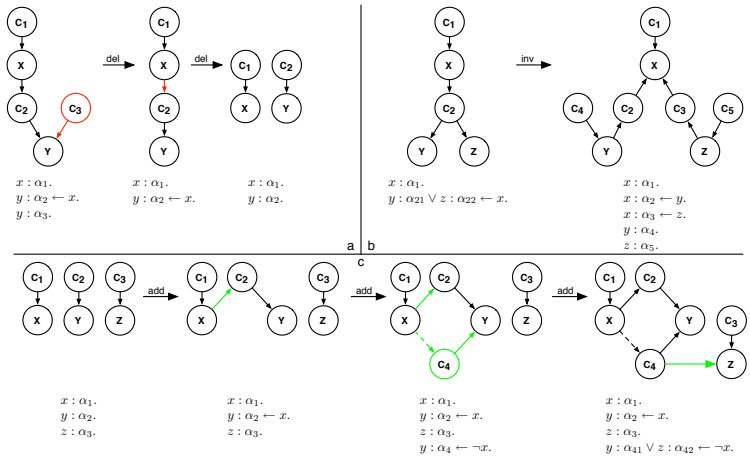


Fig. 2. Examples of the refinement operator. The dotted arrow represents a rule where that particular literal is negated in the body of the rule.

the choice node. To create a new rule, a new choice node must be created with incoming and outgoing edges based on the literals that are present in the new rule.

In the first step of Fig. 2.c, a literal is added to the body of the second rule. The equivalent step in the Bayesian network space is adding the incoming edge to the choice node departing from the literal node, corresponding respectively to the rule and the literal in the CP-theory.

This simple addition of a literal to the body of a rule, is a too simplistic step typically resulting in a CP-theory with a low likelihood. Before introducing an extension that will overcome this problem, we will first explain in more detail why this is necessary. In a CP-theory a literal can become only true if it has a *reason* to become true, i.e., if the body of a rule where it is in the head is true. Suppose that a literal is only present in one rule as for example in the second rule of the CP-theory in Fig. 2.c ($y : \alpha_2$). The literal y can become true independent of the other literals (because the body is always true). After adding a literal to the body of that rule (x after step 1), the head can only become true if the body (x) is true. Suppose that in the target theory, y can also be true if x is false (y has multiple causes), the current theory will then have a low likelihood, because the data will contain cases where y is true and x false. The new body of the rule constraints the head too much, but, on the other hand, we may want to have a relation between the head and the new body as one of the possible causes.

To check if the new body is the cause of the head, but not the only one, we perform a *lookahead* step. When the likelihood is very low after adding a literal

to the body of a rule, we add an additional new rule. This rule is identical to the previous one with the exception that the newly added literal is negated. This new rule covers the causes not yet discovered, be it in a rudimentary way. In a subsequent step the algorithm can find another cause for the head and add this to the rule with the negated literal, possibly even removing this negated literal in a future step. This addition of this new rule is illustrated in step 2 of Fig. 2.c.

Inverting a Rule

With inverting we mean switching the direction of the causation. It is sometimes possible that the algorithm learns the first relationship between literals with the wrong direction of causation. Because the algorithm builds further upon already learned networks, the incorrect direction of the causation may persist when subsequent relationships are added. And although the initial relationship had, by coincidence, a good likelihood, the following steps are not optimal. This operator can detect such situations and reverse them.

The direction of causation is defined in a rule in the CP-theory from the body towards the head. To invert this direction we must somehow switch head and body. Consequently the edges of the corresponding choice node have to be inverted. It is, however, not possible to just invert all the edges. If a rule has multiple consequences, it is necessary to split up the node into a different choice node for every consequence in the rule. This is because the consequents in the head are a disjunction and only reversing the direction of the edges would convert this disjunction into a conjunction, which is incorrect. Therefore, we create a separate rule for each consequent (see Fig. 2.b). The same reasoning applies for rules with multiple conditions in the body.

6 Experiments

As experiment we compare the performance of our algorithm to learn a CP-theory with the SEM algorithm to learn a Bayesian network (no unobserved variables). The SEM algorithm used can be found in the Structure Learning Package [6], which is an extension to the Bayesian Network Toolbox [7] for Matlab.

We construct the input data by sampling interpretations from the CP-theory described in the shopping example (Fig. 1). The test set is fixed and consists of 1000 examples. In each trial, we sample a training set of a given size. The learning algorithms train on this sample, and the resulting models (CP-theory or Bayesian network) are tested on the fixed test set. We report, for each method and training set size, the test set log likelihood averaged over at least 10 trials. Fig. 3 presents the results. The thick lines are the mean log likelihood of both methods and the dashed lines represent 90% confidence intervals for the mean. The graph also plots the minimum and maximum log likelihood measured for both methods over all trials.

The figure shows the results of the CP-logic structure learning algorithm and the SEM structure learning for Bayesian networks. For the two methods, the

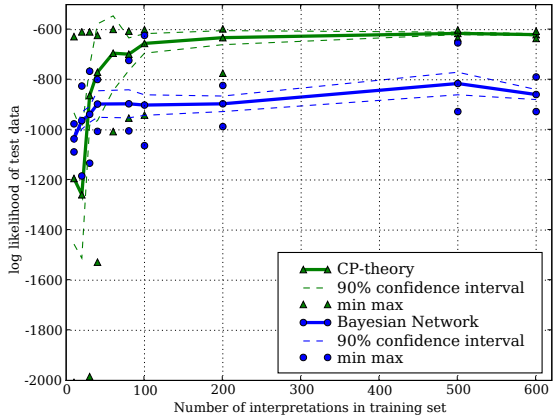


Fig. 3. Comparing structure learning of CP-theories and Bayesian Networks.

average log likelihood converges, but the learned CP-logic theory has on average a better log likelihood than the Bayesian network. This is because the Bayesian network has difficulties to represent a causal structure like the shopping example. It is possible that the log likelihood of the Bayesian network comes close to the log likelihood of the CP-theory as can be seen for the cases where there are 100 and 500 examples in the training set. This is however an exception as the SEM algorithm prefers a small network, and representing the target theory requires many edges (and parameters) if no additional hidden variables are introduced. The SEM algorithm seldom finds such a network, but if it does, it raises the average likelihood a little bit.

Causality imposes a strong connection between the literals' truth values in the data set. In this case the data set is derived from a causal process and in CP-logic this can be described with a small theory. A Bayesian network, on the other hand, requires many edges and parameters to represent such a theory. Therefore, it is more useful to search for a CP-theory in this case. When an incorrect causal relationship is inferred from the data set, this can result in a theory with a very low likelihood because of the same reasons. Suppose that in a given training set a literal y is only true when the literal x is true, this may result in learning this causal relation. If this relationship is not present in the target theory, the test set will also contain cases where y is true and x is false. Such a case is not covered by the learned structure and results in an extremely low likelihood as can be seen from the minima in the graph. These, however, are exceptions and therefore the average is still good. Running the algorithm multiple times starting from different initial theories often solves this.

Because of this strong causal connection, when learning from a small training set, the quality of the training set is important for learning a correct CP-theory. This can be seen when looking at the maxima in the graph; even when learning from small training sets, the algorithm can learn a good theory (the best log likelihoods in our experiments correspond to cases where the algorithm learned the correct structure of the target theory). When the training set is large enough the algorithm finds a good theory for every training set.

7 Conclusions and Future Work

We proposed a method for learning the structure of a ground CP-theory. It is based on existing methods for learning Bayesian networks, but uses refinement operators specific to CP-logic. The main advantage of this approach is the possibility to reuse some of the large amount of the research available in the Bayesian networks literature [4, 5].

We have compared structure learning for CP-theories to structure learning for Bayesian networks. CP-theories better approximate the target theory than Bayesian networks if it contains causal relationships. Compared to CP-theories, standard Bayesian network algorithms need to learn more edges and parameters in the corresponding CPTs to approximate such a theory, and this is detrimental to the efficiency of the learning.

We consider the following directions for further work. We plan to extend the experimental evaluation. First, it would be interesting to compare our method for learning CP-theories to a method for learning Bayesian networks that allows the introduction of hidden variables; our translation of CP-theories also uses these for the choice nodes. Second, there are specific techniques for representing causality in Bayesian networks [11]. It would be interesting to compare to such work. Our current evaluation is on one artificial domain. We plan to evaluate the method on more data sets and in real world applications. Finally, the present paper considers ground CP-theories. Further work will address learning of CP-theories with abstract rules (i.e., rules with variables).

Acknowledgments

Wannes Meert is supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Vlaanderen). Jan Struyf and Hendrik Blockeel are post-doctoral fellows of the Research Foundation - Flanders (FWO-Vlaanderen). Research supported by project GOA/2003/08 (B0516) on Inductive Knowledge Bases.

References

1. H. Blockeel and W. Meert. Towards learning non-recursive LPADs by transforming them into Bayesian networks. In S. Muggleton and R. Otero, editors, *International Conference on Inductive Logic Programming*, 2006.

2. N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. 14th International Conference on Machine Learning*, pages 125–133. Morgan Kaufmann, 1997.
3. N. Friedman and M. Goldszmidt. Building classifiers using Bayesian networks. In *AAAI/IAAI, Vol. 2*, pages 1277–1284, 1996.
4. D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. In *Proc. 10th Conf. Uncertainty in Artificial Intelligence*, pages 293–301, San Francisco, CA, 1994. Morgan Kaufmann Publishers.
5. F. V. Jensen. *Introduction to Bayesian Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
6. P. Leray and O. Francois. BNT structure learning package: Documentation and experiments. Technical report, Laboratoire PSI, 2004.
7. K. P. Murphy. Bayesian network toolbox, http://bnt.sourceforge.net/.
8. R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, Upper Saddle River, NJ, USA, 2003.
9. A. Onisko, M. J. Druzdzel, and H. Wasylu. Learning Bayesian network parameters from small data sets: Application of Noisy-OR gates. In *Working Notes of the Workshop on Bayesian and Causal Networks: From Inference to Data Mining, 12th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, Germany, 22 August 2000.
10. J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, San Francisco, 1988.
11. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
12. F. Riguzzi. Learning logic programs with annotated disjunctions. In A. Srinivasan and R. King, editors, *14th Internation Conference on Inductive Logic Programming (ILP2004), Porto*, pages 270–287, Heidelberg, Germany, September 2004. Springer Verlag.
13. J. Vennekens, M. Denecker, and M. Bruynooghe. Extending the role of causality in probabilistic modeling. In *Proceedings of the 11th International Workshop on Non-monotonic Reasoning*, pages 183–190, 2006.
14. J. Vomlel. Noisy-or classifier. *International Journal of Intelligent Systems*, 21(3):381–398, 2006.

Learning Ground ProbLog Programs from Interpretations

Fabrizio Riguzzi

ENDIF, Università di Ferrara, Via Saragat 1, 44100 Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract. The relations between ProbLog and Logic Programs with Annotated Disjunctions imply that Boolean Bayesian networks can be represented as ground ProbLog programs and acyclic ground ProbLog programs can be represented as Boolean Bayesian networks. This provides a way of learning ground acyclic ProbLog programs from interpretations: first the interpretations are represented in tabular form, then a Bayesian network learning algorithm is applied and the learned network is translated into a ground ProbLog program. The program is then further analyzed in order to identify noisy or relations in it. The paper proposes an algorithm for such identification and presents an experimental analysis of its computational complexity.

Keywords: Probabilistic Logical Models, ProbLog, LPAD, Noisy Or.

1 Introduction

ProbLog [4] is a recent formalism that combines logic and probability. It is interesting for the simplicity of its semantics and for the availability of an efficient top-down interpreter. Logic Programs with Annotated Disjunctions (LPADs) [9] is another formalism for integrating probability and logic in a clear and elegant way.

In this paper, the relations between ProbLog and LPADs are investigated. We show that ground ProbLog programs can be represented as LPADs in a way that preserves the semantics. This allow us to apply results obtained for LPADs to ProbLog programs. In particular, we show that a Bayesian network with binary variables can be represented as a ground ProbLog program and that a ground acyclic [1] ProbLog program can be translated into a Boolean Bayesian network. Representing a Bayesian network as a ground ProbLog program has the advantage of a more compact representation in the case in which noisy or relations are present.

Ground ProbLog programs that encode a Bayesian network take a special form that we call Bayesian in which all the bodies of the rules for an atom contain the same set of atoms and all the possible combinations of signs of literals in the bodies are present. We present a method for transforming a general

ground ProbLog program into a program in Bayesian form. The method involves applying the noisy or law to obtain the probability of clauses.

The possibility of representing a Boolean Bayesian network with a ground ProbLog program provides a method for learning it from interpretations: we first transform the input interpretations into tabular form, then we apply a Bayesian network learning algorithm and we translate the learned network into a ground ProbLog program in Bayesian form. In order to obtain general ground acyclic ProbLog programs, we propose an algorithm that identifies the noisy or relations in the program.

The paper is organized as follows. In section 2 we introduce ProbLog and LPADs. In section 3 we present some properties of ground ProbLog programs. Section 4 discusses the learning problem and the algorithm. Section 5 presents experiments and section 6 discusses related works. Finally, section 7 concludes the paper.

2 Preliminaries

A ProbLog program [4] P is a finite set of clauses of the form

$$\alpha : h \leftarrow b_1, \ldots, b_m \tag{1}$$

where α is a real number between 0 and 1, h and $b_1, \ldots, b_m$ are atoms. We will call $H_B(P)$ the Herbrand base of P.

In this paper we consider an extension of the original ProbLog language where $b_1, \ldots, b_m$ are literals.

The semantics of the extended ProbLog is defined in terms of instances: an instance is a normal logic program obtained by selecting a subset of the clauses. Its probability is given by the product of the α factor for all the clauses that are included in the instance and of $1 - \alpha$ for all the clauses not included. The probability $\pi^P_{PB}(\phi)$ of a query ϕ according to program P is given by the sum of the probabilities of the instances that have the query as a consequence according to a chosen semantics, e.g. Clark's completion [2], stable models [5] or well founded [7]. In this paper we consider only the well founded semantics because it is the one used by LPADs.

A Logic Program with Annotated Disjunctions (LPAD) P [9] consists of a set of formulas of the form $h_1 : \alpha_1 \vee h_2 : \alpha_2 \vee \ldots \vee h_n : \alpha_n \leftarrow b_1, b_2, \ldots b_m$ called *annotated disjunctive clauses*. In such a clause the h_i are logical atoms, the b_i are logical literals and the α_i are real numbers in the interval $[0, 1]$ such that $\sum_{i=1}^{n} \alpha_i = 1$.

The semantics of LPADs is given as well in terms of instances: an instance is a ground normal program obtained by selecting for each clause of the grounding of P one of the heads. The probability of the instance is given by the product of the probabilities associated with the heads selected. The probability $\pi^P_{LP}(\phi)$ of a formula ϕ according to program P is given by the sum of the probabilities of the instances that have the formula as a consequence according to the well founded semantics.

3 Properties of ProbLog

A ground ProbLog program P can be syntactically transformed into an LPAD P' by substituting each clause of the form (1) with the LPAD clause

$$h : \alpha \vee none : (1 - \alpha) \leftarrow b_1, \ldots, b_m$$

where *none* is a special atom that does not appear in the body of any clause. This is similar to the way in which a CP-logic program [8] can be transformed into an LPAD.

Theorem 1. *Given a ground ProbLog program P and a query ϕ, $\pi^P_{PB}(\phi) = \pi^{P'}_{LP}(\phi)$.*

Proof. (Skecth) There is a one to one correspondence between instances of P and instances of P': the clauses excluded from an instance of P are present in the corresponding instance of P' with *none* in the head. A query (that does not involve the special atom *none*) is true in an instance of P if and only if it is true in the corresponding instance of P'.

In fact, for well-founded models, it can be shown that the sequence of interpretations I_α for an instance of P are equal to $I'_\alpha \setminus \{none\}$ where I'_α is the sequence of interpretations for the corresponding instance of P'.

Therefore a ground program defines the same probability distribution when interpreted as a ProbLog program and as an LPAD. For non ground programs the semantics differ, because the instances of ProbLog programs are obtained by selecting clauses from the non-ground program while the instances of LPADs are selected from a grounding of the program.

From the equivalence between ground ProbLog programs and LPADs follows that Bayesian networks with binary variables can be translated into ProbLog programs [9]. For example, the classic burglary Bayesian network (see Figure 2 in [9]) can be translated into the following ProbLog program

$0.1 : burglary \qquad 0.2 : earthquake$
$0.1 : alarm \leftarrow \neg burglary, \neg earthquake$
$0.8 : alarm \leftarrow \neg burglary, earthquake$
$0.8 : alarm \leftarrow burglary, \neg earthquake$
$1.0 : alarm \leftarrow burglary, earthquake$

We will say that a ProbLog program like the one above is in Bayesian form.

Definition 1 (Bayesian form). *A ground ProbLog program P is in Bayesian form if, for every atom a of $H_B(P)$, all the clauses for a have bodies built over the same set of atoms D_a and there is a clause for every possible combination of signs of literals built over D_a.*

It was shown in [9] that a ground, finite and acyclic [1] LPAD can be translated into a Bayesian Logic Program (BLP) preserving the semantics. Since BLP encode Bayesian networks, this provide a way of translating such type of LPADs

into Bayesian networks. Thus we can translate a ground acyclic ProbLog program into a Bayesian network. The class of acyclic programs is an important one because the ProbLog proof procedure may not terminate for cyclic programs.

We present here the technique. Given a ground acyclic ProbLog program P, we build a Bayesian network by associating each atom a in $H_B(P)$ with a binary variable a with values *true* and *false*. Moreover, for each rule r of the form

$\alpha : h \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_l$

we add to the Bayesian network a new variable V_r that has $b_1, \ldots, b_m, c_1, \ldots, c_l$ as parents and has the two values h and *none*, where *none* is a special atom that does not appear anywhere in the body of rules. The conditional probability table (CPT) of V_r is

	$V_r = h$	$V_r = none$
...	0.0	1.0
$b_1 = true, \ldots, b_m = true, c_1 = false, \ldots, c_l = false$	α	$1 - \alpha$
...	0.0	1.0

Moreover, each variable a with $a \in H_B(P)$ has as parents all the variables V_r of rules r that have a in the head. The CPT for a is the following:

	$a = true$	$a = false$
all the parents equal to *none*	0.0	1.0
remaining rows	1.0	0.0

We will now show that every ground ProbLog program can be translated into Bayesian form in a way that preserves the semantics. This is done by collecting, for each atom $a \in H_B(P)$, all the rules R_a with a in the head. From R_a, the atoms D_a that appear in bodies of rules for a are collected. Then a rule c for each combination of signs of literals built on D_a is generated. The probability α_c of c is given by the law of the probability of an or:

$$\alpha_c = 1 - \prod_{r \in R_a | body(c) \models body(r)} (1 - \alpha_r)$$

Thus the bodies of the clauses for the same atom a constitute the causes of a in a noisy or model. For example the program

$0.3 : a \leftarrow b \quad 0.2 : a \leftarrow c$

can be transformed into

$0.0 : a \leftarrow \neg b, \neg c \quad 0.2 : a \leftarrow \neg b, c$

$0.3 : a \leftarrow b, \neg c \quad 0.44 : a \leftarrow b, c$

Theorem 2. *The transformation into Bayesian form preserves the semantics for ground acyclic ProbLog programs.*

Proof. We will prove the theorem by showing that the Bayesian networks encoded by the two programs are equal. Let P and P' be the programs before and after the transformation. Consider an atom a and let $D_a = \{b_1, \ldots, b_n\}$ be the set of atoms on which a depends. Suppose that P contains k rules for a

$R_a = \{r_1, \ldots, r_k\}$ and that rule r_i is annotated with probability α_i. Let d_a be a vector of values for D_a, let V_{R_a} be the vector of variables corresponding to the rules of R_a and let v_{R_a} be a vector of values for V_{R_a}. The probability for $a = true$ given d_a in the network obtained from P is given by

$$P(a = true|d_a) = 1 - P(a = false|d_a) = 1 - \sum_{v_{R_a}} P(a = false, v_{R_a}|d_a) =$$
$$= 1 - \sum_{v_{R_a}} P(a = false|v_{R_a}, d_a)P(v_{R_a}|d_a) =$$
$$= 1 - \sum_{v_{R_a}} P(a = false|v_{R_a})P(v_{R_a}|d_a)$$

Let V_{r_i} be the variable associated to rule r_i and let v_{r_i} be the value for V_{r_i} in v_{R_a}, thus

$$P(a = true|d_a) = 1 - \sum_{v_{R_a}} P(a = false|v_{R_a}) \prod_{i=1}^{k} P(v_{r_i}|d_a)$$

The only value v_{R_a} of V_{R_a} for which $P(a = false|v_{R_a}) \prod_{i=1}^{k} P(v_{r_i}|d_a)$ is different from 0 is the one where every v_{r_i} is equal to *none*. In fact, if $v_{r_i} = a$, then $P(a = false|v_{R_a})$ is 0. Suppose that, given the values of d_a, the set of rules $T_a \subseteq R_a$ has the body true. For the case in which every v_{r_i} is equal to *none*, $P(v_{r_i}|d_a) = 1 - \alpha_i$ if $r_i \in T_a$, $P(v_{r_i}|d_a) = 1$ if $r_i \notin T_a$ and $P(a = false|v_{R_a}) = 1$. Thus we have

$$P(a = true|d_a) = 1 - \prod_{r_i \in T_a} (1 - \alpha_i)$$

which is exactly the law for the probability of an or.

4 Learning Ground Acyclic ProbLog Programs

We consider a learning problem of the following form [6]:
Given:

- a set E of examples that are couples $(I, \pi(I))$ where I is an interpretation and $\pi(I)$ is its associated probability, such that $\sum_{(I,\pi(I))\in E} \pi(I) = 1$
- a space of possible ground ProbLog programs S (described by a language bias LB)

Find: a ground ProbLog program $P \in S$ such that $\forall (I, \pi(I)) \in E \;\; \pi^P_{PB}(I) = \pi(I)$

Instead of a set of couples $(I, \pi(I))$, the input of the learning problem can be a multiset E' of interpretations. From this case we can obtain a learning problem

of the form above by computing a probability for each interpretation in E' by relative frequency.

The approach we propose for learning a ground acyclic ProbLog program consists in transforming the input data into a table, learning a Bayesian network from the table, converting the learned network into a ground ProbLog in Bayesian form and then identifying the noisy or relations in order to obtain a general ground acyclic ProbLog program.

The translation of the input interpretations into a table is done by considering each atom appearing in them as a binary random variable. Each interpretation I is then transformed into a binary vector B_I where the variable corresponding to atom a assumes value 1 if $a \in I$ and value 0 otherwise.

We obtain the table to be given as input to the Bayesian network learning algorithm by fixing the number of rows N of the table and replicating the binary vector B_I a number of times equal to $N \times \pi(I)$.

The translation of the learned Bayesian network into a general ground ProbLog program is performed by the algorithm Identification that analyzes the CPT of each atom and tries to identify the noisy or relations in order to apply the inverse of the transformation into Bayesian form. If it fails in finding such a relation, it returns the CPT converted into ProbLog rules.

Identification, shown in Figure 1, performs exhaustive search in the space of possible bodies of rules. It takes as input, besides the learned Bayesian network, also the parameters $MaxBodySize$, $MaxRules$ and ϵ that define, respectively, the maximum number of literals in the body of rules, the maximum number of rules for an atom and the error allowed. The first two parameters put a limit on the search space in order to contain the computational cost.

Identification analyzes the CPT of each atom in turn. For an atom a the algorithm first builds all possible sets of parents of a from cardinality 1 to cardinality $MaxBodySize$. Then, with function Select (show in Figure 2), it considers all possible combinations of signs for the atoms in each set, thus generating the possible bodies PB. In this phase, the possible bodies that appear in a row of the CPT where the probability of a is close to 0 (smaller than ϵ) are eliminated, because they cannot be possible causes.

Then the combinations of possible bodies are explored with function Explore (shown in Figure 3) by performing a depth first search in the space of subsets of PB. Explore is called with a current set of bodies and the set of bodies not yet added to the current set: it first check if the current set of bodies contains all possible parents of a and if the set of bodies respects the noisy or relation. If so, it returns the current set of bodies. Otherwise, if the current set of bodies has not yet reached cardinality $MaxRules$, it performs a cycle in which, at each iteration, it adds a possible body and calls itself recursively. If no set of bodies respecting the noisy or relation can be found, the empty set is returned and Identification translates the CPT directly into rules.

The test that a set of possible bodies respects the noisy or relation is performed by function Check (shown in Figure 4). The function considers only the rows with $P(a|row) > \epsilon$, because the bodies true in the other rows have already

been removed. The function first identifies the probability of each body considered as a single cause, looking for those rows where a single body is true. The probability of a body is given by the average of the probabilities of such rows. Then it checks that for all the rows the or law is respected with an error smaller than ϵ.

Fig. 1. Algorithm Identification

algorithm Identification(
  inputs : B : Bayesian network,
    ϵ: maximum error,
    $MaxBodySize$: integer,
    $MaxRules$: integer,
  returns : P : ProbLog program)

let B be a set of triples $(Variable, Parents, CPT)$ one for each variable
$P := \emptyset$
for every triple $(Variable, Parents, CPT)$ in B
  $F := \{r | r$ is a row of CPT such that $P(Variable = true|r) < \epsilon\}$
  $G :=$ set of all the possible subsets of $Parents$ from dimension 1 to dimension $MaxBodySize$
  $PB :=$Select(G, F)
  $Bodies :=$Explore$(PB, CPT, \emptyset, MaxRules, \epsilon)$
  if $Bodies \neq \emptyset$ then
    convert $Bodies$ into a set of rules R
  else
    convert CPT into a set of rules R
  $P := P \cup R$
return P

Let us show the behavior of the algorithm with an example. Consider an atom a that has b, c and d as parents and that has the conditional probability table shown below:

row	b	c	d	a	$\neg a$
1	*false*	*false*	*false*	0.00	1.00
2	*false*	*false*	*true*	0.30	0.70
3	*false*	*true*	*false*	0.00	1.00
4	*false*	*true*	*true*	0.30	0.70
5	*true*	*false*	*false*	0.00	1.00
6	*true*	*false*	*true*	0.30	0.70
7	*true*	*true*	*false*	0.40	0.60
8	*true*	*true*	*true*	0.58	0.42

Suppose that $MaxBodySize$ is 2, $MaxRules$ is 3 and that ϵ is 0.01. The set F of rows with $P(a|row) = 0$ is $F = \{\{\neg b, \neg c, \neg d\}, \{\neg b, c, \neg d\}, \{b, \neg c, \neg d\}\}$

Fig. 2. Function Select

```
function Select(
    inputs : G : set of sets of parents,
        F: rows with probability of the child variable < ε,
    returns : PB : set of possible bodies)

PB := ∅
for every parent set Par from G
    let B be the set of all possible assignment of signs to variables of Par
    for every b ∈ B
        if b ∉ F then
            PB := PB ∪ {b}
return PB
```

Fig. 3. Function Explore

```
function Explore(
    inputs : PB : possible bodies,
        CPT: conditional probability table
        Bodies: current set of bodies,
        MaxRules: maximum number of rules,
        ε: maximum error,
    returns : B : valid set of bodies)

if Bodies contains all possible parents and Check(Bodies, CPT, ε) then
    return Bodies
else
    if |Bodies| = MaxRules then
        return ∅
    else
        let PB be {b_1, b_2, ..., b_n}
        for i := 1 to n
            Bodies' :=Explore({b_{i+1}, ..., b_n}, CPT, Bodies ∪ {b_i}, MaxRules)
            if Bodies' ≠ ∅ then
                return Bodies'
        return ∅
```

Fig. 4. Function Check

function Check(
  inputs : B : a set of bodies,
    CPT: conditional probability table,
    ϵ: maximum error,
  returns : *Satisfy* : a Boolean value)

remove from CPT all the rows r with $P(Variable = true|r) < \epsilon$
for every body b in B
  let R_b be the set of rows of CPT that contains only b true
  let $p_b = \sum_{r \in R_b} \frac{P(Variable=true|r)}{|R_b|}$
for each row r of CPT
  let B' be the set of bodies of B that are true in r
  $TP := 1 - \prod_{b \in B'}(1 - p_b)$
  if $|TP - P(Variable|r)| > \epsilon$ then
    return *false*
return *true*

The set G of possible subsets of the set of $Parents$ with maximum size 2 is

$$G = \{\{b\}, \{c\}, \{d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$$

The function Select is called and the set PB of possible bodies is returned

$$PB = \{\{d\}, \{b, c\}, \{b, d\}, \{\neg b, d\}, \{c, d\}, \{\neg c, d\}\}$$

Then Explore is called with $Bodies = \emptyset$. $Bodies$ does not contain all possible parents of a so Explore is called again with $Bodies = \{\{d\}\}$. $Bodies$ still does not contain all possible parents of a so Explore is called again with $Bodies = \{\{d\}, \{b, c\}\}$. Since all the parents are now present in $Bodies$, Check is called.

Rows 1, 3 and 5 are removed form CPT. The probabilities p_d and $p_{b,c}$ are computed: p_d is obtained from rows 2, 4 and 6 and has value 0.3 while $p_{b,c}$ is obtained form row 7 and has value 0.4.

Then the probabilities of a in rows 2, 4, 6, 7 and 8 are checked. In row 2, 4 and 6 only d is true, so it is checked that $|p_d - P(a|row)| < \epsilon$. This is true so Check continues. Row 7 has only b, c true and the test of $|p_{b,c} - P(a|row_7)| < \epsilon$ succeeds. In row 8, both d and b, c are true, so it is checked that $|TP - P(a|row_8)| < \epsilon$. TP is $1 - (1 - p_d)(1 - p_{b,c}) = 1 - (1 - 0.3)(1 - 0.4) = 1 - 0.42 = 0.58$ so the test succeeds and Check returns true.

Therefore b, c and d are recognized as valid bodies and the rules

$0.4 : a \leftarrow b, c \quad 0.3 : a \leftarrow d$

are returned.

With this approach, we may have problem when learning ProbLog programs that are not layered, i.e. when its Herbrand base cannot be divided into subsets (layers) such that each atom directly depends only on atoms from the previous

layer. In fact, in that case the probabilities in the CPT may not be estimated correctly.

For example, consider a dataset generated from the program

$0.6 : c \leftarrow a \quad 0.7 : c \leftarrow b$

$0.2 : d \leftarrow a, b \quad 0.9 : d \leftarrow c$

The case in which a and b are false and c is true never appears in the data because if a and b are false then so is c. Therefore the probabilities in the corresponding row of the CPT for d cannot be estimated. A typical approach used by Bayesian network learning algorithms is to assign probability 0.5 to d true and to d false. In this case the CPT of d does not respect the noisy or law that would assigns probability 0.9 to d true in that row and therefore the rules for d cannot be identified.

This problem does not appear if the program from which the data is generated is layered. For the program above the Herbrand base cannot be divided in layers because d depends on c and on a and b that belong to the layer preceding c.

5 Experiments

A series of experiments were performed for investigating the time complexity of the algorithm. A number of programs consisting of the definition of a single atom a have been generated: each program consists of two clauses, the programs differ for the number of atoms on which a depends that ranges from 4 to 10. The parents of a are distributed among the two clauses in order to have two bodies whose lengths differ by at most 1. Programs of this form were considered because they represent the worst case, since the bodies in PB are ordered from the shortest to the longest. From the programs, the CPT for a is produced with the transformation procedure presented in section 3.

Algorithm Identification is applied with the following parameters: $MaxRules$ is set to 3 and $MaxBodySize$ is set to values ranging from 3 to 10. The execution times in milliseconds are shown in Figure 5. The experiments were performed with Sicstus Prolog 3.12.5 on a Windows machine with a 2.00 GHz Pentium M and 1 Gb of RAM.

The missing points correspond to combinations for which the algorithm terminates because Sicstus gave an insufficient memory error: Sicstus 3.12.5 has a limitation of 256 Mb for the stack on 32 bit machines. For 10 parents the algorithm has successfully terminated only for $MaxBodySize = 3$ in 24.5 seconds while for higher values of $MaxBodySize$ has given an insufficient memory error,

For the points where $MaxBodySize < \lceil Parents/2 \rceil$ the algorithm returned a program in Bayesian form because the solution was outside the search space.

A number of experiments were conducted to test the feasibility of the whole approach: the aim was to learn back a ground ProbLog program. A few ProbLog programs were written, all the possible interpretations for them were generated and assigned a probability according to the semantics. The sets of annotated interpretations were then translated into tables and given as input to a Bayesian

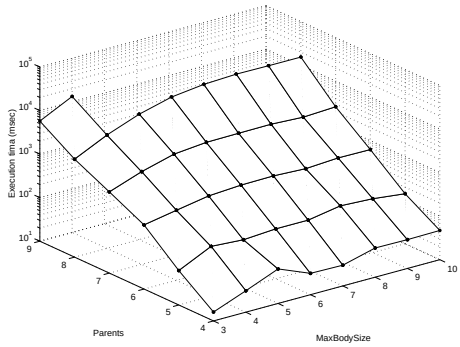


Fig. 5. Execution times

learning algorithm from the suite WEKA. Then the algorithm Identification was applied to the learned networks.

For example, the approach was tested on a program containing 9 atoms and 14 rules. From it, a table containing approximately 30,000 rows was generated and given as input to the implementation of the K2 algorithm available in WEKA. K2 exploits the ordering of the variables so the correct order was supplied to it. The other parameters were left at default values except for initAsNaiveBayes, set to false, and for the maximum number of parents of a node, set to 8.

K2 learned a Bayesian network in 0.3 seconds. Then Identification was applied with $\epsilon = 0.05$, $MaxBodySize = 3$ and $MaxRules = 3$. In 0.42 seconds Identification returned the original ProbLog program. None of the other Bayesian network learning algorithms available in WEKA were able to correctly discover the dependencies encoded by the original program.

Experiments with programs of similar complexity were performed using K2 and in all cases the original programs were returned.

6 Related Works

In [3] the authors propose an approach for revising ProbLog programs. The learning problem they consider consists in finding a subsets of clauses from a given program that maximizes the likelihood of a set of examples in the form of ground goals. Moreover, they set an upper limit to the cardinality of the program to be returned. Thus the approach in [3] is complementary to the one given here, where a set of interpretations is considered as input.

Another related work is [6] where the author proposes the algorithm ALLPAD for learning ground LPADs from interpretations. However, ALLPAD can only

learn LPADs with mutually exclusive bodies and thus it cannot learn ProbLog programs encoding a noisy or.

7 Conclusions

We have presented an approach for learning ground acyclic ProbLog programs from interpretations. The approach consists in translating the input interpretations into tabular form, applying a Bayesian network learning algorithm and then trying to identify noisy or relations in the learned network.

The identification algorithm has been experimentally tested and it was found feasible for a number of parents up to 8. Experiments of the overall approach showed that it was possible to perfectly recover ground programs of around 10 atoms and 14 rules.

8 Acknowledgements

This work has been partially supported by the PRIN 2005 project "Specification and verification of agent interaction protocols".

References

1. K. R. Apt and M. Bezem. Acyclic programs. *New Generation Comput.*, 9(3/4):335–364, 1991.
2. K. L. Clark. Negation as failure. In *Logic and Databases*. Plenum Press, 1978.
3. L. De Raedt, K. Kersting, A. Kimmig, K. Revoredo, and H. Toivonen. Revising probabilistic prolog programs. In *Proceedings of the 16th International Conference on Inductive Logic Programming*, number 4455 in LNAI. Springer, 2007.
4. L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *Proceedings of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.
6. F. Riguzzi. ALLPAD: Approximate learning of logic programs with annotated disjunctions. In *Proceedings of the 16th International Conference on Inductive Logic Programming*, number 4455 in LNAI. Springer, 2007.
7. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
8. J. Vennekens, M. Denecker, and M. Bruynooghe. Representing causal information about a probabilistic process. In *Proc. of the 10th Eur. Conf. on Logics in Artificial Intelligence*, LNAI. Springer, September 2006.
9. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *Proc. of the 20th Int. Conf. on Logic Programming*, 2004.

Towards a Framework for Relational Learning and Propositionalization

Ulrich Rückert and Stefan Kramer

Institut für Informatik/I12, Technische Universität München, Boltzmannstr. 3, D-85748 Garching b. München, Germany, {rueckert,kramer}@in.tum.de

Abstract. We present first steps towards a general framework for propositionalization and relational learning based on sequences of queries and models, and the information effectively needed to generate both. From an abstract point of view, we only consider sequences of queries sent to a database, and sequences of consecutive models that combine those queries in a decision function. On a more detailed and procedural level, we consider how the queries in a sequence are actually generated, and, in particular, which information is taken into account to do that. In this way, the framework can address the question of how well the provided information is used by different learning approaches. While we provide a categorization scheme for existing methods, the framework's main purpose is to address a number of theoretical and practical questions. On the theoretical side, questions concerning model selection and overfitting avoidance can be addressed. More practically, we present a simple visualization scheme comparing the generalization performance of methods. Finally, the framework could provide hints for software design or for combining known building blocks in novel ways.

1 Introduction

Traditional ILP or relational learning approaches generate queries dynamically, interleaving feature construction and model construction.[1] By contrast, these steps are decoupled in propositionalization approaches, and features are generated upfront before the actual learning stage. While this difference is generally acknowledged by many authors, a more formal treatment of this matter is hard to find in the literature. The most elaborate definition of propositionalization by Krogel and Wrobel [3] deals with both existential and aggregate features, but does not formally define the difference between relational learning and propositionalization. In this paper, we propose a formal framework for relational learning and, particularly, propositionalization based on sequences of queries posed to a database (or knowledge base, for that purpose). A formal framework of this type

[1] In the paper, we will use the term "queries" in the sense of "features", i.e., as functions providing information about instances. Moreover, traditional ILP systems such as FOIL [7] or Tilde [1] are called "relational learning" systems here, in contrast to propositionalization systems. Relational learning and propositionalization methods are studied within ILP.

has several advantages: First, it offers clarity in terminology: Which approach can be categorized as propositionalization, and which as relational learning? If we agree on the definitions and terminology, it is safe to categorize a method as one or the other. Second, we can address theoretical questions, for instance, concerning model selection and overfitting avoidance. How much information is effectively used in generating, filtering, and sorting queries, and incorporating them into a classifier? The point of overfitting critically depends on the amount and type of information presented to a learning algorithm. It is certainly an interesting question how well a learning system makes use of the provided information and the framework offers some tools to frame these questions analytically. Third, the framework is useful in the evaluation and comparison of algorithms. Generalization performance is often measured with different feature sets and different numbers of features, making it hard to compare the results of two methods. Based on our framework, we propose a simple scheme allowing a comparison of design decisions for relational learning and propositionalization algorithms. Fourth, the framework could provide hints for the components of a modular architecture, and thus for the implementation of learning systems. In particular, it specifies where and in which way a database is accessed to obtain information about training or test instances. As a first step, the RUMBLE system [9] is designed with a modular query generation, allowing for an easy analysis in the framework. This paper is organized as follows: In Section 2, the framework is described on an abstract level, where we only consider sequences of queries and models. Section 3 zooms in and presents a more detailed view on the way queries are generated, and which information is actually used in the process. In Section 4, we present experimental results with four learning strategies that demonstrate the different ways of incorporating information in the learning process, as described in the framework. Finally, we summarize and conclude the paper in Section 5.

2 Framework: High-Level View

Let $\mathcal{X}$ be a space of arbitrary instances, the *instance space*. We do not make any assumptions about the semantics or representation of instances; they may be graphs, arbitrary objects in the real world or mathematical constructs. Instead of imposing any syntactical or semantical restrictions on the instances, we make use of a *query language* to derive pieces of information about specific instances. For our purposes a *query* is a function from the instance space to the real numbers extended with a "don't know" symbol: $q : \mathcal{X} \to R$, where $R := \mathbb{R} \cup \{\varepsilon\}$ and the abstention symbol ε denotes "no information available". We assume that a query extracts some information from the instances in $\mathcal{X}$, such as the presence or absence of some properties, the number of nodes, or the result of some mathematical function. In typical applications, one would encode nominal values, such as *red*, *blue* and *green* using two-valued indicator variables and numerical (e.g. size) or aggregated quantities (e.g. mean size of referenced objects) as real numbers. We assume a fixed space $\mathcal{Q}$ of possible queries. We

also need a way to encode a query as a string of symbols in a computer. For that purpose, we define a language $\mathcal{L}_Q$, so that every string $s_q \in \mathcal{L}_Q$ represents exactly one query $q \in \mathcal{Q}$.[2] For ease of notation, we do not distinguish between q and s_q explicitly, but simply write q whenever we mean "q as a string", and $q(x)$ whenever we apply the query q as a function to x.

Later on, we will apply queries to whole sequences of instances $X \in \mathcal{X}^n$. Again, we denote the component-wise application of q on such as sequence of instances X by $q(X)$, so that $q : \mathcal{X}^n \to R^n$ can be seen a a function of arbitrary arity n. Finally, let $\mathcal{Y} := \{-1, 1\}$ be the set of *class labels*, so that $\mathcal{X} \times \mathcal{Y}$ denotes the *labeled instance space*. We are concerned with a learning system, that induces classifiers[3] from data. We assume that the system is given a training set $(X, Y) \in (\mathcal{X} \times \mathcal{Y})^n$ of n labeled instances $(x_1, y_1), \ldots, (x_n, y_n)$. A classifier is a function that assigns a class to each instance in $\mathcal{X}$. In our setting we formalize this as follows: a classifier $c := (k, d_c, Q_c)$ of size k contains a *decision function* $d_c : R^k \to \mathcal{Y}$ and a sequence of queries $Q_c \in \mathcal{Q}^k$. A classifier assigns a class label to each instance by evaluating the queries and combing the query results through the decision function: if $Q_c = (q_1, \ldots, q_k)$, then $c(x) := d_c(q_1(x), \ldots, q_k(x))$. Let $\mathcal{C}$ be the space of all classifiers. In particular, we denote the empty classifier by c_ε. A *learning system* L is given a training set (X, Y) as input. After some computations it outputs a classifier c that can be used to make new predictions on new instances. Thus, we can formalize a learning system as a function $L : (X, Y) \to \mathcal{C}$.

Of course, the learning system is implemented as a computer program. In order to gather information about the training set, it poses queries to the database, which contains the training instances. We would like to classify and investigate the various ways a learning algorithm generates new queries and ultimately induces a final classifier. To do so, we observe the queries $q_1, q_2, \ldots, q_l$ that are sent from the system to the database in chronological order. We assume that L is a deterministic algorithm, so that each q_i is uniquely identified by the input to L, that is, the training set and (possibly) some input parameter.[4] Finally, we do not demand that the queries are evaluated on all instances in the database. Some algorithms (e.g. separate-and-conquer approaches) perform the queries only on subsets of instances, others only on single instances (e.g. SVMs). For simplicity of representation we assume that the database always returns a vector $s \in R^n$, where the undesired components of s are simply set to ε.

---

[2] Of course, many query strings might represent the same query. Dealing with semantically equivalent query strings is an interesting problem, but out of the scope of this paper.

[3] All of the following considerations are equally valid if one is concerned with regression rather than classification. Further, generalizations to more complex learning settings such as multi-class, multi-label, or multi-task learning are straightforward.

[4] This is not a severe limitation as most non-deterministic algorithms use a pseudo random number generator in practice. The framework could also be formulated for randomized algorithms, although analytical statements about the use of information would be more difficult, as they depend on random events.

Since we are concerned with deterministic learning machines only, a query q_k is uniquely identified by the list of preceding queries $q_1, \ldots, q_{k-1}$, the corresponding result vectors $(q_1(X), \ldots, q_{k-1}(X))$ and the class label vector Y. Let $\vec{q}_k : \mathcal{L}_Q^k \times R^{n \times k} \times Y \to \mathcal{Q}$ be the function, that decides, which new query is sent to the database, after the system has sent the preceding k queries, and observed the corresponding instantiations and the class label vector Y.[5] Thus, the kth query can be reconstructed by an application of $\vec{q}_k$ on the data that is returned by the database for $\vec{q}_1, \ldots, \vec{q}_{k-1}$.

Some algorithms build the classifier to be output in an incremental fashion. For example, separate-and-conquer rule learning systems iteratively add new rules to an initially empty rule set until a stopping criterion is met. To model this stepwise refinement approach, we peek inside the learning system to observe which (partial) classifiers have been built so far. We denote the current classifier of the algorithm after sending k queries to the database as c_k. When the machine maintains no classifier until query k, we set the $c_1, \ldots, c_k$ to the empty classifier c_ε. By convention, $c_0 = c_\varepsilon$, because there is no current classifier before the system starts and $c_l = L(X, Y)$, because after the last query is posed, the algorithm builds and outputs the final classifier. Similar to the r_i, the system's current classifier after query q_k depends only on the queries so far, their instantiations and the class labels. We define the function $\vec{c}_k : \mathcal{L}_Q^k \times R^{n \times k} \times Y \to \mathcal{C}$ to output the current classifier of the system after seeing the first k queries, the corresponding instantiations and Y.

This allows us to categorize learning systems into certain categories. First of all, we can discriminate between systems that build the classifier incrementally and systems that generate all the queries first, but induce the final classifier only in the last step. We say that a learning system is a *propositionalisation system*, if for all $k < l$: $c_k = c_\varepsilon$ and $c_l = L(X, Y)$. Otherwise, the system is a *stepwise refining system*. In principle, every refinement system can be reduced to a propositionalisation system, because one can modify the $\vec{c}_k(q_1, \ldots, q_k, q_1(X), \ldots, q_k(X), Y)$ to output only the empty classifier for $k < l$ and keep only the last classifier generation function $\vec{c}_l$. However, having access to an initial (partial) classifier can speed up the computation of the $\vec{q}_k$ and $\vec{s}_k$ considerably. Many practical systems are therefore implemented as refining systems. Another way to categorize multi-relational learning systems is by the information that is utilized in the $\vec{q}_k$ and $\vec{s}_k$. One can distinguish between three sources of information:

- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the preceding *query representations*. If the generation of new queries does not depend on the representation of the preceding queries, we call the system *agnostic propositionalization*. For instance, in the case of small molecule data, a learning system could simply process a list of predefined queries that check for certain active functional groups of a small molecule. However, this is rarely practical. Many systems generate the queries according to a "more general than" order imposed on the query

[5] In slight abuse of notation we use the arrow in $\vec{q}$ to indicate that $\vec{q}$ describes the transition from k to $k + 1$.

strings in $\mathcal{L}_Q$. Sometimes, the dependence of $\vec{q}_k$ on the preceding queries can by configured explicitly by a user, often by specifying *refinement operators*.
- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the preceding *instantiations*. The main motivation to use this information is to find a preferably diverse and informative set of queries. On graph data it is common to prune away queries that are satisfied or violated on less instances than a predefined threshold, because such queries introduce little discriminative power and are therefore less desirable. If the queries in $\mathcal{L}_Q$ are ordered by generality, threshold-based pruning can be implemented efficiently, because it depends only on the instantiation of the least general generalizations of a query.
- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the *class labels* Y. This is the most prominently used information for $\vec{c}_k$, because every good classifier is based on queries that are as informative as possible about the target. The influence of the class labels on $\vec{q}_k$ is less clear, because it is hard to predict if a new feature will offer valuable information about the target before evaluating it. We will see later that one can do remarkably well with $\vec{q}_k$ that do not depend on the class labels.

The categorization of learning algorithms according to the utilization of available information can be used as a foundation for theoretical analyses of existing and novel systems. For instance, a system that makes use of the class labels Y to build the queries is more susceptible to overfitting than a learner that ignores this information. An analytical investigation could quantify this phenomenon. A similar effect takes place when a learner generates highly correlated features. A simple application of the PAC-Bayesian bound can be used to investigate how query inter-correlation influences overfitting behavior. Clearly, analyses of how well certain learning systems make use of the provided information can lead to interesting and relevant insights.

In most learning systems, $\vec{q}_k$ and $\vec{c}_k$ depend on all three sources of information, although the actual dependencies vary considerably. Often, performance considerations lead to systems that use certain types of information only implicitly, so that their influence on the output is only marginal. Sometimes there are complicated interdependencies that make it hard to assess the influence of each source of information. However, there are also many learning systems which proceed in a way that can easily be formalized. For example, a very common approach to relational learning is to extend an initially empty classifier step by step until it reaches a sufficient level of proficiency. Often, such a system iterates in a loop: First, it generates a new *batch* of queries and sends it to the database. Then, it filters those queries that are likely to be relevant and adds them to the current (partial) classifier. In the next section we give a formal way to describe this (or similar) approaches to query and classifier construction.

3 Framework: Low-Level View

The level of the framework described in the preceding section focuses on the sequence of queries and models from an abstract point of view. In particular,

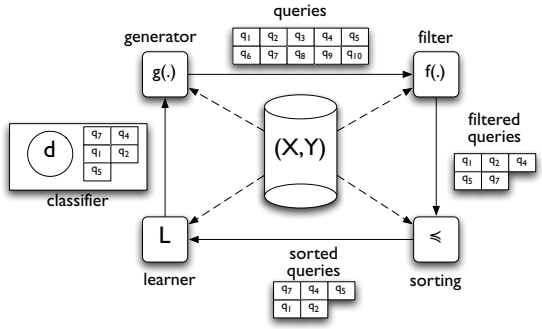


Fig. 1. The low-level view of a relational learning system in the framework: each batch of queries is generated by $g(.)$, filtered through $f(.)$, then sorted according to $\preceq$ and finally handed over to the classifier construction, where the queries are augmented with the decision function d to form an (intermediate) classifier. Each component can base its actions on previous queries, the training instances, the labels and/or previous classifiers. This procedure is repeated iteratively until a final classifier is output.

it does not consider the way the queries in a sequence are generated. In the following, we take a look inside the learning system's "black box" and take a more procedural view of the query and classifier generation process. On this level, we deal with the information effectively used to generate individual queries or batches of queries. For instance, we can express formally whether information about the target class is used directly (by looking it up in the database) or only indirectly (via looking at a partially induced classifier) in this process. To do so, we abstract from single queries and instead deal with batches of queries, where each batch contains the queries generated between two classifier updates. More formally, recall that c_k denotes the current classifier of the system after sending the kth query q_k. If a system does not update the classifier after q_k, then $c_k = c_{k-1}$. For each current classifier c let $Q(c) := \{q_i | c_i = c\}$ be the set of queries that do not lead to an update of c. It is clear that the $Q(.)$ can be used to partition the query sequence $q_0, \ldots, q_l$ into a sequence of *query batches* $Q_1 = Q(q_0), \ldots, Q_{l'} = Q(q_l)$, so that the Q_k contain exactly those queries that are sent to the database between two classifier updates. We can now investigate, how a system generates each batch of queries. For instance, when modeling a stepwise refinement system (e.g., Tilde [1]), the algorithm obtains the next batch of queries or the next query by maximizing some criterion (e.g. with respect to the target class). Another possibility is to generate the next queries depending on the parameters of the decision function d_c of the current classifier. To allow for a more fine-grained formalization of the classifier update process, we frame the construction of a query batch Q_k as follows: First, the queries in Q_k are (syntactically) generated depending on some information available at the time of generation, then they are filtered, sorted, and finally handed over to a classifier

Table 1. Sources of information for generator function, filter function, and sorting order.

	generator	filter	sorting predicate
agnostic	$g()$	$f(Q')$	$\preceq$
syntax-dependent	$g(q_1, ..., q_k)$	$f(Q')$	$\preceq$
instantiation-dependent	$g(X)$	$f_X(Q')$	$\preceq_X$
interaction-depdendent	$g(q_1, ..., q_k, X)$	$f_{q_1,...,q_k,X}(Q')$	$\preceq_{q_1,...,q_k,X}$
class-dependent	$g(q_1, ..., q_k, X, Y)$	$f_{X,Y}(Q')$	$\preceq_{X,Y}$
model-dependent	$g(q_1, ..., q_k, d_c)$	$f_{q_1,...,q_k,d_c}(Q')$	$\preceq_{q_1,...,q_k,d_c}$

construction procedure, which derives the decision function d_c. For each of these steps, we consider the information that is necessary to perform them: information about the instances, their class labels, the previously generated queries, or the partially learned model.

To formally define the process of query generation and the information used therein, we have to introduce a few functions and predicates. First, we assume that the system contains a procedure b that generates the next batch of queries. The resulting batch of queries is possibly sorted according to some criterion and presented to the classifier construction procedure. Different implementations of b are conceivable: For instance, b could use information about the queries q_1 to q_k generated so far, X resp. Y, or the current decision function d_c. Zooming in on b, we may find it practical for some systems to distinguish between the process of generating and filtering queries. In other words, we assume the batch of queries is first generated using a so-called generator procedure g and then filtered according to a filter procedure f. Thus, b consists of two parts g and f where each query generated by g is given as input to f. This does not necessarily need to happen in a chronological way (first call g, then f), but it can also happen in a more complex fashion.

Typical generator functions g generate queries syntactically based on a declarative language bias. The filter function f can be thought of as applying an interestingness predicate p, as known from the data mining literature [6], to a set of generated queries $f(Q) = \{q \in Q | p(q)\}$. In some instantiations of the framework, f may pick a single query, for instance, by optimizing a scoring function. Splitting the process into generating and filtering, it is possible to reconstruct pattern mining approaches like Warmr [2] or classical ILP systems like FOIL [7] in some detail (see below). Optionally, we can sort the resulting queries for the batch according to some order $\preceq$, which in turn may depend on various types of information. Finally, the newly generated, filtered, and sorted batch of queries has to be instantiated with respect to the instances of a database (if it has not already been instantiated by g, f or the sorting procedure), before it can be processed by the classifier construction procedure, which induces d_c.

The generator/filter functions as well as the sorting predicate may be based on different sources of information (see Table 1). If the data without the class are accessed, we have X as an argument or as a subscript. If the queries from the sequence up to index k are taken as input, we have q_1 to q_k either as ar-

gument or as subscript. Analogously, we have Y or d_c as an argument of g, or as a subscript of f or $\preceq$, if the target class or the current model is required. A schematic illustration of the classifier induction loop in this framework is given in figure 1. Modelling a relational learning system as an iterated query generation, filtering, sorting and learning procedure might appear arbitrary or ad-hoc. However, we feel that the presented framework is not only modular enough to enable theoretical and empirical investigations (c.f. the next section), but also flexible and generic enough to apply to existing systems. In the appendix we give a short survey of how a selection of existing relational learning systems can be described and classified in the framework.

4 Experimental Evaluation

The framework outlined in section 2 and 3 allows to rate a relational learning system according to the information it is using to generate queries and classifiers. With this, we have a convenient method to compare different learning systems and make justified statements about the contributions of certain design statements. For example, we could answer questions such as: On this particular kind of data, does it make more sense to generate queries of form A or form B? Is it worthwhile to keep all the features or will filtering those features, that meet condition C, help overfitting avoidance? If the goal is to pass the informative queries earlier to the learning system than non-informative queries, should one sort the generated queries according to sorting order E or F? Each of these questions could be answered by keeping parts of the learning system fixed while varying only the part under investigation. Other experiments could shed light on the interdependence between certain design decisions. For instance, one could ask: What is the best sorting order for each of three different feature generation methods? Which learning algorithms works particularly well with certain filters? Those questions are often hard to answer when comparing existing methods, because most existing systems are built in an integrated fashion so that it is difficult to rate the contribution of single design decisions.

In the following we perform two studies. In the first, we primarily compare different query generation procedures while keeping the filter and sorting order fixed. In the second study, we keep the query generation and filter stage fixed and use different sorting criteria to investigate which sorting order works best. We deal with two data sets where the goal is to predict the biological activity of small molecules, given as molecular graphs. The Yoshida dataset [11] consists of 265 molecules classified according to their bio-availability. The second dataset classifies 415 molecules according to the degree to which they can cross the blood-brain barrier [5].

For the first study, we chose the following four query generating procedures, sorted by the amount of information that is used:

- *Agnostic.* Here we simply generate all subgraphs with up to ten edges ($g()$).
- *Based on a minimum frequency constraint.* Here, we apply a frequent subgraph mining tool to identify all subgraphs that appear in at least 6% of the

Table 2. Training and test set accuracies on the yoshida (top) and bloodbarr dataset (bottom) for a varying number of queries. The left part gives the results for agnostic, minimum frequency, dispersion-based and dispersion with class correlation based feature generation, the right part gives the results for sorting by size, by balance, by class correlation and by a χ^2 test with the class.

Num. of Queries	Agnostic		Min. Freq.		Disp.		Disp.+ Class Corr.		By Size		By Balance		By Class Corr.		By χ^2 Test	
	Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst
20	45.2	45.7	56.4	53.2	66.2	58.9	72.9	64.9	65.2	58.9	65.2	59.6	71.0	61.1	72.5	62.3
40	46.6	45.7	61.8	55.5	75.8	66.4	78.3	67.5	68.6	57.0	68.5	60.0	72.7	64.5	74.1	65.3
60	51.1	47.9	63.3	55.8	78.2	68.3	80.4	66.0	73.5	63.0	75.7	66.4	73.5	64.2	74.5	66.4
80	53.2	50.2	64.6	57.4	79.8	68.7	80.0	67.5	74.3	61.5	78.9	65.3	75.1	67.2	75.4	67.5
100	53.7	49.8	66.4	56.2	81.9	68.3	81.4	67.5	80.6	67.5	81.6	68.3	75.8	66.8	76.4	67.5
150	53.8	48.7	70.5	57.0	83.6	68.3	82.5	63.4	84.9	68.7	82.8	67.5	77.1	67.5	78.7	65.3
200	54.9	50.2	73.4	60.0	85.1	66.0	82.8	63.4	86.1	69.1	83.7	68.7	77.9	66.4	80.3	66.0
20	61.1	63.6	63.9	62.7	71.7	67.7	76.6	71.6	73.0	70.8	73.3	71.1	77.3	74.9	76.3	74.5
40	64.4	61.2	61.2	55.7	76.8	70.8	78.4	73.5	75.3	69.9	73.7	72.5	77.4	75.9	77.3	74.7
60	65.2	62.4	66.7	60.2	78.0	71.8	79.5	72.3	77.5	72.0	76.2	72.8	77.8	74.7	77.8	74.2
80	59.4	56.9	66.2	61.4	78.4	71.3	80.6	71.8	80.7	74.7	78.9	73.3	78.3	72.3	78.1	73.0
100	62.4	60.7	66.1	59.0	80.7	73.5	82.0	74.0	82.0	74.7	79.6	72.5	79.4	73.5	78.3	72.8
150	68.8	65.5	67.6	59.8	81.8	73.0	82.0	69.6	83.9	74.9	80.0	71.8	80.1	72.8	80.0	72.3
200	70.7	66.0	74.1	64.8	84.4	72.8	83.9	73.0	84.8	75.2	81.1	72.0	80.9	73.0	82.2	73.7

graphs in the database ($g(X)$). The implementation is based on a depth-first search.

- *Based on dispersion.* Dispersion based query generation [9, 10] aims at a diverse set of queries, so that each query's instantiation is as different as possible from the instantiations of the other queries. To this end, we devised a scoring function that measures the dissimilarity (dispersion) of a query set and apply a stochastic local search algorithm to find subgraph queries whose instantiation optimizes this score ($g(q_1, ..., q_k, X)$).
- *Based on dispersion and correlation to the target class.* This is the same as the preceding strategy, except that the scoring function is modified so that dispersion and correlation with the target contribute in equal parts ($g(q_1, ..., q_k, X, Y)$).

In each of the four cases we use a simple filter that discards all queries whose instantiations are duplicates of already existing features ($f(X)$). We do not sort the generated features, but hand them to the learning algorithm in the order they were generated. A good strategy finds relevant queries first and less informative queries only later so that the system can stop early without compromising predictive accuracy. Stopping as early as possible is desirable because it leads to fast learning systems that induce small and compact classifiers. Thus, to investigate the performance of each strategy, we generate a fixed number of queries and use those as features in a linear classifier. We apply Margin Minus Variance (MMV) [8] with $b = 0$ and $p = 2$ to learn the linear classifier. We give the results

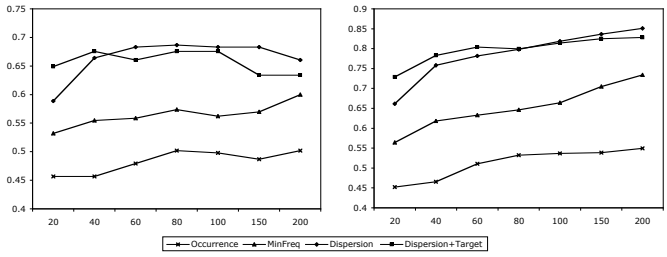


Fig. 2. Predictive accuracy (left) and training set accuracy (right) for linear classifiers on the Yoshida data set plotted against on the number of queries posed to the database.

in the left part of table 2. It can be seen that the first two strategies do not make efficient use of the information provided by the database. For both strategies, predictive accuracy and training set accuracy increase only slowly and are significantly lower than the corresponding quantities for the second two strategies. For the two dispersion based strategies there is surprisingly little difference. The correlation to the target, which is used in the fourth strategy, causes the predictive and training set accuracy to be better for small numbers of queries. However, after a certain number of queries are generated, the "dispersion only" strategy not only catches up, but also does not overfit as strongly as the fourth strategy. This indicates that – at least for the two investigated data sets – finding queries that complement each other is more important than finding queries that are informative about the target on their own. The visualization of training and test accuracy in figure 2 illustrates this phenomenon on the yoshida dataset.

For the second study, we keep the feature generation and filtering fixed, but investigate different sorting criteria. Sorting is particularly interesting, if one is aiming at small and comprehensible feature sets and wishes to stop query generation as early as possible. We generate all queries by mining for subgraphs that are contained in at least 6% of the database graphs ($g(X)$). The queries are then filtered by the same filter as above, i.e. by discarding all features that give rise to the same instantiation as an already existing query. We investigate the following four sorting criteria:

- *By size.* Here we sort the subgraph queries according to the number of edges in the subgraph ($\preceq$). The sorting order is from few edges to many edges.
- *By balance.* In this case, queries are sorted according to how evenly the +1 and -1 are assigned in a query instantiation ($\preceq_X$). Queries, which assign +1 to the same number of instances than -1 are ranked first, while queries that assign +1 (or -1) to only a single instance are ranked last. The idea is to prefer queries whose instantiations split the training set into parts of preferably equal size, because those provide more information and can be better used to scatter the instance space.

- *By class correlation.* This simply sorts the queries according to the Pearson correlation coefficient between query instantiation and target class vector ($\preceq_{X,Y}$).
- *By* χ^2. This is similar to the preceding sorting criterium, except for the use of a χ^2 test on the 2x2 contingency table instead of a simple correlation coefficient ($\preceq_{X,Y}$).

We give training and test accuracies (estimated by tenfold cross validation) in the right part of table 2. Again, sorting dependent on the class information works better than the first two criteria only for a small number of features. For 150 or 200 queries, sorting by subgraph size outperforms the other methods on both datasets. It is surprising that an agnostic sorting criterium using neither information about the training set nor the target class performs best.

5 Conclusion

We presented a general framework for propositionalization and relational learning, which allows for the categorization and experimental comparison of existing and novel approaches. The main idea of the framework is to abstract from everything except the sequence of queries and (partial) models, and the way they are generated. In particular, we center on the precise information needed to create sequences of queries presented to a learning algorithm. By doing so, we can find interesting answers to the question of how well different approaches make use of the provided information. The framework has two levels. On a higher level, we consider sequences of queries and models. On the lower level, we identify functions and predicates necessary for the generation of the queries in this sequence, and the type of information they can be based on. The benefits of such a framework are as follows: First, it allows for the clarification of terminological ambiguities. Second, theoretical as well as practical questions, from overfitting avoidance to the evaluation of learning algorithms, can be addressed. To demonstrate this, we performed two experiments on graph data, which indicated that, contrary to common belief, information about the target class is not necessary to generate informative queries for well performing classifiers. In future work, the model presented in this paper should be refined further and complemented by quantitative information, e.g., by taking into account the information (in bit) used as input to the various components.

References

1. H. Blockeel and L. D. Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
2. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, 1999.
3. M.-A. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In C. Rouveirol and M. Sebag, editors, *Inductive Logic Programming, 11th International Conference*, volume 2157 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 2001.

4. N. Landwehr, A. Passerini, L. D. Raedt, and P. Frasconi. kfoil: Learning simple relational kernels. In *21st National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2006.
5. H. Li, C. W. Yap, C. Y. Ung, Y. Xue, Z. W. Cao, and Y. Z. Chen. Effect of selection of molecular descriptors on the prediction of blood-brain barrier penetrating and nonpenetrating agents by statistical learning methods. *Journal of Chemical Information and Modeling*, 45(5):1376–1384, 2005.
6. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
7. J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
8. U. Rückert and S. Kramer. A statistical approach to rule learning. In *Machine Learning, Proceedings of the 23rd International Conf.*, pages 785–792. ACM Press, 2006.
9. U. Rückert and S. Kramer. Margin-based rule learning, to appear, 2007.
10. U. Rückert and S. Kramer. Optimizing feature sets for structured data. In S. Matwin and D. Mladenic, editors, *18th ECML*. Springer, 2007.
11. F. Yoshida and J. Topliss. QSAR model for drug human oral bioavailability. *J. Med. Chem.*, 43:2575–2585, 2000.
12. F. Zelezný and N. Lavrac. Propositionalization-based relational subgroup discovery with rsd. *Machine Learning*, 62(1-2):33–63, 2006.

Appendix: Instantiations of the Framework

In this appendix, we briefly discuss known and conceivable instantiations of the above framework. First, many traditional ILP algorithms proceed top-down, iterating query generation and testing. Typically, models consist of queries that are combined conjunctively or disjunctively. Queries that are already part of a model are syntactally refined according to a declarative language bias specification. This scheme applies to many classical ILP algorithms, from FOIL [7] to Tilde [1]. In terms of the framework, the queries are generated depending on previous queries ($g(q_1, ..., q_k)$) and evaluated with respect to the class ($f_{X,Y}(Q')$). The state of the classifier d_c is not included in the generation function in this case, because the queries are combined conjunctively and no additional information has to be considered for the purpose of query generation. Propositionalization methods proceed either purely syntactically in the generation of new queries (e.g., [12], or in a bottom-up, data-driven manner (e.g., [3], or in a top-down search manner (e.g., [2]). In the RSD approach [12], queries are generated syntactically ($g()$) taking into account non-decomposability and subsequentely filtered to avoid redundancy ($f_X(Q')$). Aggregate functions for propositionalization were proposed by Krogel and Wrobel in their RELAGGS system ($g(X)$ and $f(Q') = Q'$). In his PhD thesis, Krogel also discusses the elimination of features involved in functional dependencies ($f_X(Q')$). Top-down propositionalization methods like Warmr [2] generate or refine existing queries syntactically ($g(q_1, ..., q_k)$) and filter according to frequency constraints ($f_X(Q')$). According to the scheme proposed in this paper, kFOIL [4] is categorized as relational learning and not as propositionalization ($g(q_1, ..., q_k)$ and $f_{X,Y}(Q')$).

Distributed Relational State Representations for Complex Stochastic Processes *

Ingo Thon[1] and Kristian Kersting[2]

[1] Katholieke Universiteit Leuven, Department of Computer Science
Celistijnenlaan 200A, 3001 Heverlee, Belgium
ingo.thon@cs.kuleuven.be
[2] Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, 32 Vassar St, Cambridge, MA 02139, USA
kersting@csail.mit.edu

Abstract. Several promising variants of hidden Markov models (HMMs) have recently been developed to efficiently deal with large state and observation spaces and relational structure. Many application domains, however, have an apriori componential structure such as parts in musical scores. In this case, exact inference within relational HMMs still grows exponentially in the number of components. In this paper, we propose to approximate the complex joint relational HMM with a simpler, distributed one: k relational hidden chains over n states, one for each component. Then, we iteratively perform inference for each chain given fixed values for the other chains until convergence. Due to this structured mean field approximation, the effective size of the hidden state space collapses from $O(n^k)$ to $O(n)$.

1 Introduction

In recent years, Statistical Relational Learning (SRL) has emerged as an active research subfield of Machine Learning. It is a relatively young research field that deals with machine learning and data mining in relational domains where observations may be missing, partially observed, and/or noisy. So far, however, surprisingly few SRL approaches have been developed for modeling dynamic domains, i.e., domains with temporal and/or sequential aspects. One reason might be that time is not simply another relation. The algorithmic complexity for general purpose, dynamic SRL approaches easily explodes and becomes intractable in practice if quite strong assumptions are not made such as low branching factors to keep tractability [7]. Another alternative way to keep dynamic SRL approaches tractable is to lift simple dynamic probabilistic models, which naturally restrict the dynamics of the domain, to relational models. This approach

* An earlier version of this work appeared as 4-pages extended abstract in the electronic working notes of the 5th International Workshop on Mining and Learning with Graphs (MLG'07), August 1–3, 2007, Universita degli Studi di Firenze, Florence, Tuscany, Italy. In the present paper, we report for the first time on experimental results.

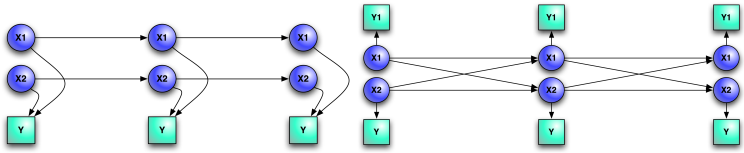


Fig. 1. Factored HMMs. (Left) a factorial HMM: independent processes X_i (ovals) are coupled through a single, joint output sequence Y_i (boxes). (Right) Weakly coupled HMMs: processes X_i (ovals) weakly interact to generate independent output sequences Y_i (boxes), one for each process.

has been followed by [1] and by [3], who lifted (hidden) Markov models to the relational case. Hidden Markov models (HMMs) [5] itself are extremely popular for modeling dynamic domains. Application areas include user modeling, speech recognition, empirical natural language processing, and robotics, but also sequential domains like computational biology.

Many application domains, however, have an apriori componential structure such as parts in musical scores. In this case, exact inference within relational HMMs still grows exponentially in the number of components due to the combinatorial nature of the state space. In the propositional case, this 'curse of compositionality' has been successfully addressed by a number of *factored* HMMs such as *factorial* HMMs [2] and *mixed-memory* Markov models [9]. Here, the (hidden) state is factored into multiple state variables and is therefore represented in a distributed manner. Moreover, the distributed nature allows to devise an efficient variational approximation by (weakly) decoupling the state variables. The main contribution of the present work is to show how to lift this idea to the relational case.

We proceed as follows. After briefly reviewing factored HMMs in the next Section, we will introduce weakly-couple relational HMMs (wcrHMMs) in Section 3. Section 4 then presents a structured mean field approximation for efficient inference within wcrHMMs. Before concluding, we will experimentally evaluate this approach.

2 Factored Hidden Markov Models

Consider modeling string quartets. A violin has a pitch range from g until $a4$ this corresponds to four octaves denoted by the number, each with 12 semi tones denoted by a letter and an optional modifier. For example, g corresponds to the 8th note of the zeros octave. Therefore, a string quartet can play $(4 \cdot 12)^4 \approx 5 \cdot 10^6$ combinations of notes (even more including double stops and flageolets). This number also corresponds to the required number of hidden state in an HMM modeling a string quartet. This is clearly an intractable state space. An alternative is to decompose the string quartet state space into four separate

state variables, namely one for each instrument. This results in a much smaller number of states per state variable, namely, only 48 values.

This decomposition is exactly the idea underlying factored HMMs. Figures 1 (left and right) show two instances of factored HMMs, which represent extreme points of the factored HMM spectrum: factorial HMMs and coupled HMMs. Unfortunately, only decomposing the state variables does not make exact inference and learning algorithms tractable [2]. The decomposition, however, paves the way for an approximative inference algorithm, which is cubic in the number of hidden state variables. The basic idea is that *each object (instrument) represented by a (hidden) state variable chooses its next state only based on the current joint state, i.e., independent of the next state of the other state variables.* This assumption together with making a structured mean field approximation allows us to show in the remainder of this text that the exponential runtime complexity drops from $O(n^{2k})$ to $O(k^3n^2)$ for one transition, where k is the number of random variables and n is the domain size of the random variables *even in the relational case.*

Why are we interested in the relational case? Reconsider our string quartet examples. The number of states for each (hidden) state variable is still very high compared to the number of state variables: 48 vs. 4. So, why not factorizing even further? Well, decomposing the state for one instrument into two random variables – one for the note and one for the octave – we would encode that changing the semitone and the octave are independent of each other. Now assume that one instrument transitions from the note 12^{th} one halftone up. The next note will be the first note but also one octave higher, which is wrong. Nevertheless, as we will argue in the next section, there are (context-specific) independencies among the state variables, which we would like to employ for fast inference.

3 Weakly-Coupled Relational Hidden Markov Models

In a factored HMM, each hidden state consist of a vector of unstructured symbols. These symbols are the joint state of a set $\mathbf{X_t}$ of random variables $\mathbf{X_t} = X_{1,t}, \ldots, X_{n,t}$ at each time t. With the term *chain* we refer to the set $\bigcup_t X_{i,t}$ representing the same object over time. The random variables $\mathbf{X_t}$ are carrying the information of the history over to the next state at time point $t+1$. As an example for a state consider:

$$\underbrace{basso\_1\_0}_{X_{1,t}}, \underbrace{alto\_1\_1}_{X_{2,t}}, \underbrace{tenor\_1\_1}_{X_{3,t}}, \underbrace{soprano\_2\_1}_{X_{4,t}}$$

The state says that the instrument *basso* plays the first note of the octave zero at time point t represented by $X_{1,t}$. We will call such a statement *ground state* and the combination of ground states for each $X_{i,t}$ at time t a *joint* ground state. Using ground states only, a traditional *factorial* HMM requires to specify the conditional probability distribution (CPD) $P(X_{i,t+1}|X_{i,t})$ for each possible state value combination. Even in our simple examples this CPD consists of 2304

$$\text{abstract state} \begin{cases} \text{body:} & note(Voice, Note, Octave) \\ \text{guard:} & note(Other, Note, Octave2) \wedge Other \neq Voice. \end{cases}$$
$$\text{head:} \quad \rightarrow note(Voice, Note, Octave2)$$

Fig. 2. An abstract transition (probability value omitted) of a weakly-coupled relational HMM. Capitalized words denote placeholders (for ground properties of the state) to share knowledge across set of states by means of unification.

entries. Additionally the hidden state values can only depend via the output. For coupled HMMs, things get even more worse. There the number of parameters also grows exponential in the number of chains. Our string quartet example, would require to specify roughly $2.5 \cdot 10^8$ parameters. This is clearly intractable.

In contrast, relational HMMs allow to aggregate sets of ground states together by using logical atoms. The above example rewritten in logical notation would be

$$\underbrace{note(basso, 1, 0)}_{X_{1,t}}, \underbrace{note(alto, 1, 1)}_{X_{2,t}}, \underbrace{note(tenor, 1, 1)}_{X_{3,t}}, \underbrace{note(soprano, 2, 1)}_{X_{4,t}}$$

Now for instance, $note(Voice, Note, Octave)$ refers to all ground states, in which an instrument $Voice$ plays any note $Note$ in any octave $Octave$. Where capitalized words denote variables and ground states are states where every variable is replaced by a constant value. This abstraction in turn allows to compactly encode the probabilistic information. In the following, we will extend relational HMMs to the weakly-coupled case.

Weakly-coupled relational HMMs are the factored variant of logical HMMs [3]. Consequently, the state of the system at each time step is a set of ground atoms (one for each chain) and not only a single ground atom. An abstract state consists of two components: a body (the state of a single chain) and a guard.

Definition 1. *An* **abstract state** $\{B, \varphi\}$ *consists of a body* B *and a guard* φ*. A body is a logical atom and specifies the set of all subsumed ground states for a chain. A mapping* θ_B *of the variables (placeholders) in* B *to objects in the domain (constants) instantiates the abstract state* B *to a ground state. The guard is a conjunction of logical atoms. It describes how one object is related to other objects in a state. The guard applied to a joint state also induces one or more mappings* $\theta_{\varphi,i}$*.*

Thus, whereas the body corresponds to an abstract state in the sense of relational HMMs [3] and in turn specifies the properties of states of a *single* random variable, the guard defines properties and relations, among all random variables. As we will see below, an abstract transition fires only if the guard is true. This can always be checked as the systems is at each time in exactly one state, i.e., one ground atom per chain. To break ties among matching abstract states, we assume the set of abstract states to be totally ordered according to some arbitrary order.

As an example, consider the abstract state shown in Fig. 2. Its meaning is that two different instruments ($Voices$) play the same note. First, the body says that there is a voice, which is playing some note. Then, the guard makes sure that there is another voice playing the same note. Note that we assume that the system is at each point in time in a particular joint ground state, i.e., we can match each placeholder (such as $Voice$, $Other$, etc.) to a domain element (constant). This variable mapping can in turn be used to specify a probability distribution over the next states, i.e., over the states the system can transits to. Following [3], we specify a distribution over possible successor states as follows.

Definition 2. *An* **abstract transition** *is an expression of the following form:*

$$p_i \ :: \ \{B, \varphi\} \rightarrow H_i$$

where $p_i \in [0, 1]$ is a probability value, $\{B, \varphi\}$ denotes an abstract state, and H is a logical atom. An abstract transition belongs to exactly one abstract state. Hence the p_i of the abstract transitions associated to the same abstract state have to sum up to one. Note that the variables appearing in the body and the guard can be used in the head. In this way, we can share knowledge across individual chains.

Figure 2 shows an example for an abstract transition. It states that the instrument playing $Voice$ takes over the octave of the another instrument (if the guard is true in the current joint state). If there are multiple true groundings of the guard, as $Other = soprano$ and $Other = alto$ when determining the abstract state for X_1 in the example, we select uniformly among them. Multiple successor states, i.e., free variables in the head are dealt with in the same way as for logical HMMs, namely by assuming a selection distribution $\mu(a|A)$ mapping atoms A to ground atoms a.

Definition 3. *A* **selection distribution** $\mu(a|A)$ *defines for every logical atom A and every ground atom a the probability that a will be a ground instance of A.*

Additionally to the transition distribution there has to be a way to define the prior distribution π over the joint ground states. To this end, we assume a finite set of expressions of the form $p :: \{H_1, \ldots, H_n\}$, i.e., one atom H_i per chain. Then, using the selection distribution, we define

$$\pi(\{h_1, \ldots, h_n\}) = P(\mathbf{X_0} = \{h_1, \ldots, h_n\}) = \alpha \sum\nolimits_{p::\{H_1,\ldots,H_n\}\in\pi} p \cdot \prod\nolimits_{i=1}^{n} \mu(h_i|H_i)$$

where α is a normalization constant. Note, however, that this is not the only way one can imagine to specify a prior over joint ground states and any of them will work fine with our inference procedure we will introduce below.

The only thing left is the definition of the *sensor* model, i.e., the probability model for making observations.

Definition 4. *A sensor model is a set of expressions of the form*

$$p \ :: S_1, \ldots, S_m \rightarrow O$$

where the S_i and O are logical atoms.

Each time an observation rule fires (assuming the same conflict resolution rule as for abstract transition rules) in a joint ground state, we make the corresponding observation (grouding free variables using the selection distribution μ).

Putting everything together results into the definition of a weakly-coupled relational HMMs.

Definition 5. *A* **weakly-coupled relational HMM** *(*WCRHMM*) consists of a set of totally ordered abstract states, sets of abstract transitions for every abstract state, a selection distribution μ, a initial state distribution π, a set of observations.*

A long the lines of [3], one can prove that every WCRHMM defines a unique probability distribution.

Theorem 1. *A weakly-coupled relational HMM defines a time discrete stochastic process $\langle \mathbf{X}_t \rangle_t$. The induced probability measure over the Cartesian product over all random variables exists and is unique for each $t > 0$ and in the limit $t \to \infty$.*

To see this, note that every WCRHMM with a finite number of chains can be translated into a logical HMM: one basically computes the Cartesian product of all abstract states and the resulting abstract transitions.
The proof of Theorem 1 provides us with a general way to do inference and learning within WCRHMMs: compile the WCRHMM into a logical HMMs and use the inference techniques developed for logical HMMs [4, 3]. This approach, however, typically scales as n^2, where n is the number of hidden state. In practice, exact inference is therefore limited to relational HMMs with relative small state spaces.

4 Structured Mean Field Approximation

Mean field theory provides an alternative perspective on inference. The intuition behind mean field is that in dense graphs each node is subject to influences from many other nodes. Assuming that each influence is rather weak and that the total influence is roughly additive, the law of large number suggest that each node should be roughly characterized by its mean value. Indeed, the mean value is unknown, but it is related to the mean values of the other nodes. For Bayesian networks and HMMs, it has been found that the mean value of a given node is obtained additively from the mean values of the nodes in its Markov blanket [8]. For weakly-coupled HMMs, however, we can do even better. Each chain individually is tractable. Thus, we can improve the mean field approximation by decoupling only the variables across the chains. This is called a *structured mean field* approach. Whenever the chains are only loosely coupled, we would expect this approximation to be quite accurate.

This basically leads to relational variants of [9]'s *chain-wise* inference procedures for mixed-memory Markov models, which all follow the same principle and are akin to the hard EM. Even though it is possible to use this procedure

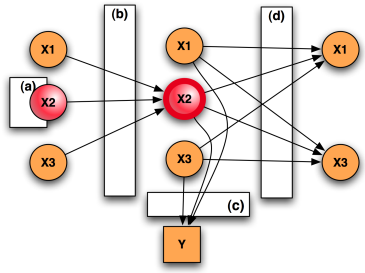


Fig. 3. Probabilistic information employed by the chainwise Viterbi algorithm to compute a transition probability for chain having all other chain fixed: (a) the probability to reach the last state, (b) the transition probability of chain i, (c) the observation probability, (d) the transition probability of the other chain from t to $t+1$ given that chain i is at t in x_i.

although for parameter learning and the estimation of a lower bound of the probability of one output sequence, let us illustrate this for the Viterbi algorithm, i.e., for computing the most-likely joint state sequence $\overline{x}_{i,0:T}$ given a sequence of observations $o_{1:T}$. First, an initial guess is made for the Viterbi path $\overline{x}^{(0)}_{i,0:T}$ of each component relational HMM i, for instance by running the Viterbi algorithm for logical HMMs for each chains separately ignoring the inter-chain dependencies. This is done by ignoring the guard. Then, a *chainwise* Viterbi algorithm is applied, in turn, to each of the relational HMMs. The chainwise Viterbi computes the optimal path of hidden $\overline{x}^{(l)}_{i,0:t}$ states through the ith chain given fixed values $\overline{x}^{(l-1)}_{j,0:t}$ of the last iteration for the hidden states of the other chains. This is essentially again the Viterbi algorithm for logical HMMs but it uses a modified transition probability:

$$\delta(x^{(l)}_{i,t}|o_{1:t}) = \max_{x_{i,t-1}} \delta(x^{(l)}_{i,t-1}|o_{1:t-1}) \quad \text{(a)}$$

$$P(x^{(l)}_{i,t}|\overline{x}^{(l-1)}_{1:i-1,t-1}, x^{(l)}_{i,t-1}, \overline{x}^{(l-1)}_{i+1:n,t-1}) \quad \text{(b)}$$

$$P(o_t|\overline{x}^{(l-1)}_{1:i-1,t}, x^{(l)}_{i,t}, \overline{x}^{(l-1)}_{i+1:n,t}) \quad \text{(c)}$$

$$\prod_{j=1:n\setminus i} P(\overline{x}^{(l-1)}_{j,t+1}|\overline{x}^{(l-1)}_{1:i-1,t}, x^{(l)}_{i,t}, \overline{x}^{(l-1)}_{i+1:n,t}) \quad \text{(d)}$$

where, cf. Figure 3, (a) is the probability to reach the last state, (b) is the transition probability of chain i, (c) is the observation probability, and (d) is the transition probability of the other chain from t to $t+1$ given that chain i is at t in $x^{(l)}_{i,t}$. After the chainwise Viterbi has been applied once to each chain, we iterate the cycle until convergence. The complete procedure RCVITERBI is given in Algorithm 1. One complete cycle of Algorithm 1 can be computed in time $O(k^3n^2)$ instead of the original $O(n^{2k})$.

Algorithm 1 RCVITERBI: Relational chainwise Viterbi

1: **procedure** UPDATE-PATH($x_{0:T}$, $o_{1:T}$, $\overline{x}_{0:T}$, i)
2: $p_{x,0} \leftarrow \pi(\overline{x}_{1,0} \dots \overline{x}_{1,i-1}, x, \overline{x}_{0,i+1} \dots \overline{x}_{0,n})$ ▷ init $p_{x,0}$. In general, $p_{x,i}$ stores the probability of the most likely path for $o_{1:t}$, which ends in x.
3: **for all** $t \in [1 \dots T]$ **do**
4: **for all** x **do**
5: $p_{x,t} \leftarrow 0$ ▷ init $p_{x,t}$, i.e., the probability of being in x at t
6: **end for**
7: **for all** x' with $p_{x',t-1} > 0$ **do** ▷ Consider only states reachable at $t-1$
8: $\{B, \varphi\} \leftarrow$ abstract state matching $\overline{x}_{1,t-1} \dots \overline{x}_{1,i-1}, x', \overline{x}_{t-1,i+1} \dots \overline{x}_{t-1,n}$
9: $\theta_B \leftarrow$ mgu of x' and B ▷ Ground the variables in the body
10: **for all** θ_φ s.t. $\varphi\theta_B\theta_H$ contains no free variable and is true in state $\overline{x}_{1,t-1} \dots \overline{x}_{1,i-1}, x', \overline{x}_{t-1,i+1} \dots \overline{x}_{t-1,n}$ **do**
11: **for all** $p :: \{B, \varphi\} \rightarrow H$ **do** ▷ For all abstract successors of x'
12: $p_{new} \leftarrow 0$
13: **for all** groundings x of $H\theta_B\theta_\varphi$ **do** ▷ For all ground successors, compute modified transition probilities (lines 13 – 21)
14: $p_a \leftarrow p_{x',t-1}$
15: $p_b \leftarrow p \cdot \mu(x|A)$
16: $p_c \leftarrow 0$
17: **for all** $p_O :: S_1, \dots, S_m \rightarrow O$ s.t. $S_1 \dots, S_m$ is true in $\overline{x}_{1,t-1} \dots \overline{x}_{1,i-1}, x', \overline{x}_{t-1,i+1} \dots \overline{x}_{t-1,n}$ **do**
18: $p_c \leftarrow p_c + p_O\mu(o, O)$
19: **end for**
20: $p_d \leftarrow 1$
21: **for all** $j < n$ and $j \neq i$ **do**
22: $p_d \leftarrow p_d \cdot P(x_{t+1,j}|\overline{x}_{1,t} \dots \overline{x}_{1,i-1}, x, \overline{x}_{t,i+1} \dots \overline{x}_{t,n})$
23: **end for**
24: $p_{new} \leftarrow p_{new} \cdot p_a \cdot p_b \cdot p_c \cdot p_d$
25: **if** $p_{new} > p_{x,t}$ **then** ▷ If more likely, set as current best path
26: $p_{x,t} \leftarrow p_{new}$
27: $pred(x, t) \leftarrow x'$
28: **end if**
29: **end for**
30: **end for**
31: **end for**
32: **end for**
33: **end for**
34: $x \leftarrow \arg\max_x p_{x,T}$ ▷ Extract the computed Viterbi path.
35: **for** t=T-1…0 **do**
36: $\overline{x}_{i,t} \leftarrow pred(x, t)$
37: $x \leftarrow \overline{x}_{i,t}$
38: **end for**
39: **end procedure**

5 Experimental Demonstration

To demonstrate the relational chainwise Viterbi algorithm, consider the vacuum world of [6] as depicted in Figure 4 for the case of 4 rooms.

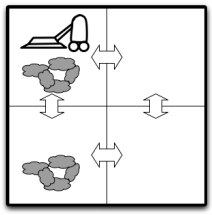


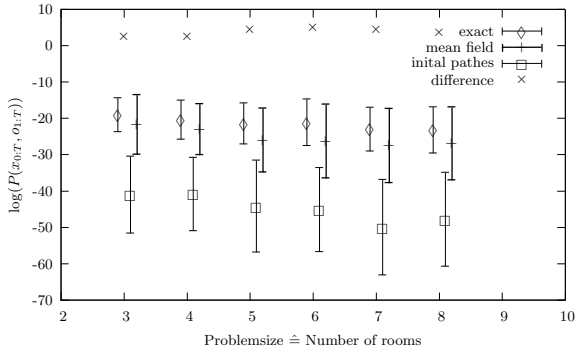
Fig. 4. Illustration of the Vacuum world we used to demonstrate RCVITERBI. Here we assume 4 rooms which are arranged in a circle. The robot is in the upper-left room.

Example 1. In the Vacuum world, there are n rooms and a single robot. The robot has two actions to choose from: walking (w) and cleaning (c). Rooms `X` and `Y` are connected via `door(X,Y)`, which the robot can use to walk from `X` to `Y`. If the robot is in a `Room` and cleaning, the room will be clean (`clean(Room)`) with a chance of 90% after the cleaning action. A clean room will stay clean in any case and a dirty room (`dirty(Room)`) will also stay dirty by default if not performing the cleaning action. The action the robot chooses correspond with probability 0.8 to the true state of the current room. In other words, the robot is not always able to determine the current state of the room correctly. The observation at every time consists of the position of the robot and dirt level of the room. This information is only with a probability 0.75 correct. In the other cases either the position of the robot is wrong or the dirt level or both.

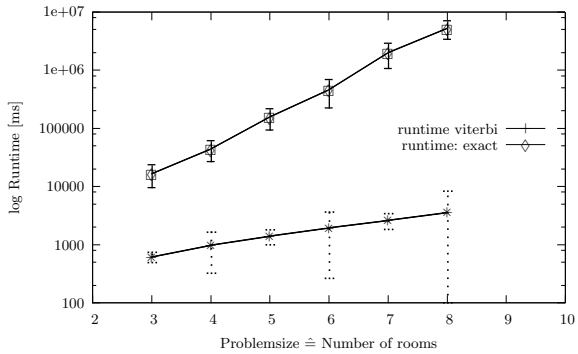
To model the Vacuum world as a WCRHMM, we treated `robot(Room,Action)`, `clean(Room)`, and `dirty(Room)` as abstract chain. The rooms as well as the topological information among them, i.e., `door(X,Y)`, is provided apriori as deterministic background knowledge. Then the Vacuum world can be modeled as follows:

$$
\begin{array}{lll}
0.9 :: & clean(X) \leftarrow dirty(X) & \{robot(X,c)\} \\
0.1 :: & dirty(X) \leftarrow dirty(X) & \{robot(X,c)\} \\
1.0 :: & dirty(X) \leftarrow dirty(X) & \{\} \\
1.0 :: & clean(X) \leftarrow clean(X) & \{\} \\
0.8 :: & robot(X,c) \leftarrow robot(X,\_) & \{door(X,Y) \wedge dirty(X)\} \\
0.2 :: & robot(Y,w) \leftarrow robot(X,\_) & \{door(X,Y) \wedge dirty(X)\} \\
0.2 :: & robot(X,c) \leftarrow robot(X,\_) & \{door(X,Y) \wedge \neg dirty(X)\} \\
0.8 :: & robot(Y,w) \leftarrow robot(X,\_) & \{door(X,Y) \wedge \neg dirty(X)\}
\end{array}
$$

Based on this model, we compared the exact and the chainwise relational Viterbi algorithms. More precisely, for an increasing number of rooms $(3, 4, \ldots, 8)$, we randomly sampled 40 observation sequences of length 10. For each sequence we then ran both algorithms to compute the most-likely hidden state sequence. The mean field approach was set to spend 4 iterations per element in the interpretation.



(a) Qualitative comparison: Number of rooms (x axis) vs. log joint probability $log(P(x_{0:T}, o_{1:T}))$ of computed Viterbi path and observation sequence (y axis), which is bascially maximized by the Viterbi algorithm. Finally, we show the probabilities of the paths, which had been used to initialize the mean field approximation. The mean of the latter probabilities roughly corresponds to the expected probability of a randomly selected state sequence.



(b) Running time comparison: Number of rooms (x axis) vs. runtime (y axis) in ms on a logarithmic scale. The skewness of error bars are due to the logarithmic scale

Fig. 5. Experimental results on the vacuum world domain averaged over 40 sequences: (a) qualitative comparison, (b) running time comparison. The results show that the structured mean field approximation achieves a performance, which is competitive with the exact inference approach, but it is several orders of magnitude faster.

The experimental results are summarized in Figures 5(a) and 5(b). As one can see in Figure 5(a), the chainwise Viterbi approach yields close approximations

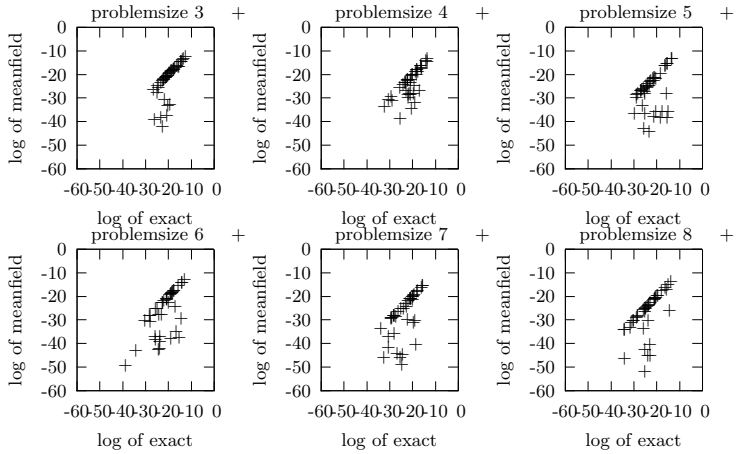


Fig. 6. Scatter-plots of the 40 experiment made for every problemsize. This shows that the results are in most cases equally good. In the other cases the solutions are still reasonable.

of the true probabilities. It is, however, an order of magnitude faster, cf. Figure 5(b), as predicted by the theory: the exact viterbi algorithm is exponential in the number of rooms as the number of possible hidden states grows exponential in the number of rooms. The quantitative results were as follows:

	#rooms					
# of paths with	**3**	**4**	**5**	**6**	**7**	**8**
same probabilities	28	18	17	20	22	27
$0\% <$ log range $\leq 5\%$	1	3	3	3	3	3
$5\% <$ log range $\leq 10\%$	3	6	7	1	3	1
subtotal (absolute/relative)	**32/.8**	**27/.67**	**27/.67**	**24/.6**	**28/.7**	**31/.77**
$> 10\%$ log range	8	13	13	16	12	9
total	**40**	**40**	**40**	**40**	**40**	**40**

Thus, in most cases both algorithms output a path with the same probability. In the cases in which the estimated path RCVITERBI is suboptimal, the solution is still reasonable. This is also illustrated in the scatter-plots in Figure 6.

To summarize the experiments demonstrate that RCVITERBI achieves comparable performance as the exact approach but is several orders of magnitudes faster.

6 Conclusions

We introduced weakly coupled relational HMMs (wcrHMMs). Based on a distributed, abstract state representation, we then developed a structured mean field approximation for efficient, approximative inference. First experiments have shown that the approximation works well in practice. This experiments have also shown, that the exact algorithm is intractable even in the simple cases, because of the exponential growth of the runtime in the size of the interpretations.
More experiment have to be conducted, particular on real world data.
To the best of our knowledge, the inference procedure is the first application of a variational method within SRL. Researchers like Pedro Domingos or Taisuke Sato have started research in the same direction[3]. Investigating this connection for other SRL approaches is an interesting direction for future research as it paves the way towards general *relational, variational Bayes* methods.

Acknowledgments
The authors thank Luc De Raedt for his support. We also would like to thank the anonymous reviewers for their helpful comments. The research was partly supported by the Research Foundation-Flanders (FWO-Vlaanderen).

References

[1] C. Anderson, P. Domingos, and D. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In *Proc. of the 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, 2002.
[2] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning Journal*, 29:245–273, 1997.
[3] K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.
[4] K. Kersting and T. Raiko. 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence (UAI-05)*, pages 300–307, 2005.
[5] L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
[6] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 1995.
[7] S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In *Proc. of the 8th Int. Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 992–997, 2003.
[8] L. K. Saul and M. I. Jordan. Exploiting tractable substructures in intractable networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 486–492. The MIT Press, 1996.
[9] L. K. Saul and M. I. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Mach. Learn.*, 37(1):75–87, 1999.

[3] personal communication

Author Index