Filip Železný
Nada Lavrač (Eds.)

LNAI 5194

Inductive Logic Programming

18th International Conference, ILP 2008
Prague, Czech Republic, September 2008
Proceedings

Springer

Lecture Notes in Artificial Intelligence 5194

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Filip Železný   Nada Lavrač (Eds.)

Inductive Logic Programming

18th International Conference, ILP 2008
Prague, Czech Republic, September 10-12, 2008
Proceedings

Springer

Volume Editors

Filip Železný
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Technická 2, 16627 Praha 6, Czech Republic
E-mail: zelezny@fel.cvut.cz

Nada Lavrač
Jožef Stefan Institute
Department of Knowledge Technologies
Jamova 39, 1000 Ljubljana, Slovenia
E-mail: Nada.Lavrac@ijs.si

Library of Congress Control Number: 2008933735

CR Subject Classification (1998): I.2.3, I.2.6, I.2, D.1.6, F.4.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN	0302-9743
ISBN-10	3-540-85927-6 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-85927-7 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12512553 06/3180 5 4 3 2 1 0

Preface

The 18th International Conference on Inductive Logic Programming was held in Prague, September 10–12, 2008. ILP returned to Prague after 11 years, and it is tempting to look at how the topics of interest have evolved during that time. The ILP community clearly continues to cherish its beloved first-order logic representation framework. This is legitimate, as the work presented at ILP 2008 demonstrated that there is still room for both extending established ILP approaches (such as inverse entailment) and exploring novel logic induction frameworks (such as *brave induction*). Besides the topics lending ILP research its unique focus, we were glad to see in this year's proceedings a good number of papers contributing to areas such as statistical relational learning, graph mining, or the semantic web. To help open ILP to more mainstream research areas, the conference featured three excellent invited talks from the domains of the semantic web (Frank van Harmelen), bioinformatics (Mark Craven) and cognitive sciences (Josh Tenenbaum). We deliberately looked for speakers who are not directly involved in ILP research. We further invited a tutorial on statistical relational learning (Kristian Kersting) to meet the strong demand to have the topic presented from the ILP perspective. Lastly, Stefano Bertolo from the European Commission was invited to give a talk on the ideal niches for ILP in the current EU-supported research on intelligent content and semantics.

For the main technical track, we received 46 abstracts followed by 36 full-paper submissions. Eight of them were accepted and sixteen rejected after reviews. The remaining 12 papers were given a conditionally-accepted status and their authors were given an additional three weeks to revise them. The revised papers were then re-reviewed by the program chairs and found acceptable for publication. It is our belief that the extra work demands we laid on the authors this year were an effective means of ensuring the quality of the conference. ILP 2008 also received 22 short submissions for the late-breaking papers track, which were reviewed separately. The accepted short papers appear in a separate proceedings book. Extended versions of selected papers from both conference tracks will appear in a special issue of the Machine Learning Journal.

Organizing ILP 2008 was a great experience, thanks to the excellent help we received on several fronts. We are indebted to our generous sponsors who made possible the trips of the invited speakers (sponsored by the US Air Force European Office of Aerospace Research and Development, the PASCAL2 Network of Excellence, the Czech Society for Cybernetics and Informatics and the European Commission) and the best student paper prize (sponsored by the Machine Learning Journal). We are equally grateful to Springer for collaborating so flexibly and pro-actively in preparing these proceedings and the Machine Learning Journal special issue. Thanks go as well to the diligent program committee for their reviews of the submitted papers. Their review activity was supported by the

MyReview System, which proved to be a powerful, yet easy-to-use, conference management software.

ILP would be just three letters were it not for the authors of the submitted papers. To them goes our foremost gratitude. Keep up the good work and submit to ILP 2009!

July 2008

Filip Železný
Nada Lavrač

ILP 2008 Organization

Program Chairs

Filip Železný	Czech Technical University, Prague
Nada Lavrač	Jožef Stefan Institute, Ljubljana

Local Organization Chairs

Jiří Kléma	Czech Technical University, Prague
Peter Vojtáš	Charles University, Prague
Milena Zeithamlová	Action M Agency, Prague

Program Committee

Annalisa Appice, Italy
Hendrik Blockeel, Belgium
Rui Camacho, Portugal
James Cussens, UK
Luc De Raedt, Belgium
Sašo Džeroski, Slovenia
Floriana Esposito, Italy
Alan Fern, USA
Peter Flach, UK
Tamás Horváth, Germany
Katsumi Inoue, Japan
Andreas Karwath, Germany
Kristian Kersting, Germany
Stefan Kramer, Germany
John Lloyd, Australia
Francesca Lisi, Italy
Donato Malerba, Italy
Stan Matwin, Canada
Stephen Muggleton, UK
Ramon Otero, Spain
C. David Page, USA
Bernhard Pfahringer, New Zealand
Jan Ramon, Belgium
Céline Rouveirol, France
Vitor S. Costa, Portugal
Jude Shavlik, USA
Takayoshi Shoudai, Japan
Ashwin Srinivasan, India
Prasad Tadepalli, USA
Tomoyuki Uchida, Japan
Christel Vrain, France
Stefan Wrobel, Germany
Akihiro Yamamoto, Japan
Mohammed J. Zaki, USA
Gerson Zaverucha, Brazil

Local Organization Committee

Petr Buryan
Matěj Holec
Jiří Kubalík
Onřej Kuželka
Karel Moulík
Petr Pošík
Olga Štěpánková
Monika Žáková

Invited Speakers

Josh Tenenbaum	Massachusetts Institute of Technology
Kristian Kersting	Fraunhofer IAIS
Frank van Harmelen	Vrije Universiteit Amsterdam
Mark Craven	University of Wisconsin in Madison
Stefano Bertolo	European Commission

Additional Reviewers

Claudia d'Amato
Michelangelo Ceci
Jiří Kléma
Aline Paes
Ganesh Ramakrishnan
Ulrich Rückert
Chiaki Sakama

Sponsors

Czech Technical University in Prague
Jožef Stefan Institute, Ljubljana
Charles University in Prague
US Air Force, European Office of Aerospace Research and Development
PASCAL2 European Network of Excellence
Czech Society for Cybernetics and Informatics
Machine Learning Journal
European Commission

Table of Contents

Invited Talks

Building Theories of the World: Human and Machine Learning Perspectives 1
Joshua B. Tenenbaum

SRL without Tears: An ILP Perspective 2
Kristian Kersting

Semantic Web Meets ILP: Unconsumated Love, or No Love Lost? 3
Frank van Harmelen

Learning Expressive Models of Gene Regulation 4
Mark Craven

Information Overload and FP7 Funding Opportunities in 2009-10 5
Stefano Bertolo

Research Papers

A Model to Study Phase Transition and Plateaus in Relational Learning 6
Erick Alphonse and Aomar Osmani

Top-Down Induction of Relational Model Trees in Multi-instance Learning 24
Annalisa Appice, Michelangelo Ceci, and Donato Malerba

Challenges in Relational Learning for Real-Time Systems Applications 42
Mark Bartlett, Iain Bate, and Dimitar Kazakov

Discriminative Structure Learning of Markov Logic Networks 59
Marenglen Biba, Stefano Ferilli, and Floriana Esposito

An Experiment in Robot Discovery with ILP 77
Gregor Leban, Jure Žabkar, and Ivan Bratko

Using the Bottom Clause and Mode Declarations on FOL Theory Revision from Examples 91
Ana Luísa Duboc, Aline Paes, and Gerson Zaverucha

DL-FOIL: Concept Learning in Description Logics 107
Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

Feature Discovery with Type Extension Trees 122
Paolo Frasconi, Manfred Jaeger, and Andrea Passerini

Feature Construction Using Theory-Guided Sampling and Randomised Search 140
Sachindra Joshi, Ganesh Ramakrishnan, and Ashwin Srinivasan

Foundations of Onto-Relational Learning 158
Francesca A. Lisi and Floriana Esposito

L-Modified ILP Evaluation Functions for Positive-Only Biological Grammar Learning 176
Thierry Mamer, Christopher H. Bryant, and John McCall

Logical Hierarchical Hidden Markov Models for Modeling User Activities 192
Sriraam Natarajan, Hung H. Bui, Prasad Tadepalli, Kristian Kersting, and Weng-Keen Wong

Learning with Kernels in Description Logics 210
Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

Querying and Merging Heterogeneous Data by Approximate Joins on Higher-Order Terms 226
Simon Price and Peter Flach

A Comparison between Two Statistical Relational Models 244
Lorenza Saitta and Christel Vrain

Brave Induction 261
Chiaki Sakama and Katsumi Inoue

A Statistical Approach to Incremental Induction of First-Order Hierarchical Knowledge Bases 279
David J. Stracuzzi and Tolga Könik

A Note on Refinement Operators for IE-Based ILP Systems 297
Alireza Tamaddoni-Nezhad and Stephen Muggleton

Learning Aggregate Functions with Neural Networks Using a Cascade-Correlation Approach 315
Werner Uwents and Hendrik Blockeel

Learning Block-Preserving Outerplanar Graph Patterns and Its Application to Data Mining 330
Hitoshi Yamasaki, Yosuke Sasaki, Takayoshi Shoudai, Tomoyuki Uchida, and Yusuke Suzuki

Author Index 349

Building Theories of the World: Human and Machine Learning Perspectives

Joshua B. Tenenbaum

Massachusetts Institute of Technology
jbt@mit.edu

Abstract. Human knowledge of the world is often expressed in the form of intuitive theories: systems of abstract concepts that organize, predict and explain our observations of the world. How are these powerful knowledge structures represented and acquired? I will describe computational frameworks for modeling people's intuitive theories and theory-building processes, and some ways of testing these models experimentally with human learners. Our models of human learning and inference build on core approaches in Bayesian artificial intelligence, statistical relational learning and inductive logic programming, but also suggest new ways to extend these machine learning and reasoning approaches to more human-like capacities.

[This talk describes joint work with Charles Kemp, Noah Goodman, Yarden Katz, Kristian Kersting and Tom Griffiths.]

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, p. 1, 2008.

SRL without Tears: An ILP Perspective

Kristian Kersting

Fraunhofer IAIS
kristian.kersting@iais.fraunhofer.de

Abstract. Statistical relational learning addresses one of the central questions of artificial intelligence: the integration of probabilistic reasoning with first order logic representations and machine learning. A rich variety of different formalisms and learning techniques have been developed. This tutorial provides an gentle introduction to and an overview of the state-of-the-art in statistical relational learning. It starts from classical settings for inductive logic programming and shows how they can be extended with probabilistic methods. It touches upon lifted inference and recent developments in nonparametric approaches to statistical relational learning. While doing so, it reviews state-of-the-art statistical relational learning approaches.

[The tutorial is partially based on material prepared earlier jointly with James Cussens, Luc De Raedt, Brian Milch, Leslie Pack Kaelbling, Josh Tenebaum, Kurt Driessens, and many more].

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, p. 2, 2008.

Semantic Web Meets ILP: Unconsumated Love, or No Love Lost?

Frank van Harmelen

Vrije Universiteit Amsterdam
Frank.van.Harmelen@cs.vu.nl

Abstract. Even at first glance, ILP and the Semantic Web have much in common. Both are about large volumes of data, both make use of background knowledge , and both use computationally tractable forms of logic. Nevertheless, the actual intersection of the two research areas is very small. In this talk, I will first give a birds eye overview of the Semantic Web programme (its goals, its methods, its achievements to date, and the important open challenges). I will then consider the most obvious use of ILP for the Semantic Web: could ILP be used to learn the ontologies that are such a crucial ingredient in the Semantic Web story? However, as with any result from Machine Learning, such ontologies will not be fully correct and complete. This will require that, from its side, the Semantic Web community must learn how to deal with such partially incomplete and incorrect ontologies. I will present the most recent work in this direction, the efforts to build LarKC, the Large Knowledge, a platform for infinitely scaleable distributed incomplete Semantic Web reasoning. Could the Large Knowledge Collider be the place where ILP and the Semantic Web finally meet?

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, p. 3, 2008.

Learning Expressive Models of Gene Regulation

Mark Craven

University of Wisconsin
craven@biostat.wisc.edu

Abstract. A central challenge in computational biology is to uncover the mechanisms and cellular circuits that govern how the expression of various genes is controlled in response to a cell's environment. This challenge presents many interesting opportunities for machine-learning methods, especially those that employ expressive representations. In this talk, I will discuss recent research in using machine-learning approaches to (i) recognize regulatory elements in genomic sequences, (ii) uncover networks of interactions among genes, and (iii) characterize the cellular responses induced by various stimuli. I will highlight tasks that call for models that use expressive representations, and discuss lessons learned about what types of representational attributes are important for these tasks.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, p. 4, 2008.

Information Overload and FP7 Funding Opportunities in 2009-10

Stefano Bertolo

European Commission
stefano.bertolo@ec.europa.eu

Abstract. Physical tools like the lever and the loom amplify human strength and dexterity and are often introduced, as in the case of the agricultural harvester, because a sudden abundance of one quantity of interest (wheat fields) introduces scarcity in other related quantities (the human labor needed to harvest them). The rate at which digital information is becoming available (both to organisations and individuals) is creating a similar scarcity in our ability to interpret it to make decisions that benefit us. In this talk I will describe funding opportunities that the EU will make available through its Framework Programme 7 to tackle this scarcity. I will discuss several trends that make the ILP community ideally placed to contribute to these efforts, together with some general patterns that have proved very effective in the engineering of successful proposals in the recent past.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, p. 5, 2008.

A Model to Study Phase Transition and Plateaus in Relational Learning

Erick Alphonse and Aomar Osmani

LIPN-CNRS UMR 7030, Université Paris 13, France
erick.alphonse@lipn.univ-paris13.fr,
aomar.osmani@lipn.univ-paris13.fr

Abstract. The feasibility of symbolic learning strongly relies on the efficiency of heuristic search in the hypothesis space. However, recent works in relational learning claimed that the phase transition phenomenon which may occur in the subsumption test during search acts as a plateau for the heuristic search, strongly hindering its efficiency. We further develop this point by proposing a learning problem generator where it is shown that top-down and bottom-up learning strategies face a plateau during search before reaching a solution. This property is ensured by the underlying CSP generator, the RB model, that we use to exhibit a phase transition of the subsumption test. In this model, the size of the current hypothesis maintained by the learner is an order parameter of the phase transition and, as it is also the control parameter of heuristic search, the learner has to face a plateau during the problem resolution. One advantage of this model is that small relational learning problems with interesting properties can be constructed and therefore can serve as a benchmark model for complete search algorithms used in learning. We use the generator to study complete informed and non-informed search algorithms for relational learning and compare their behaviour when facing a phase transition of the subsumption test. We show that this generator exhibits the pathological case where informed learners degenerate into non-informed ones.

1 Introduction

According to [Mit82], symbolic learning is defined as search: given a hypothesis space defined a priori, identified by its representation language, find a hypothesis consistent with the learning data. This paper, relating symbolic learning to search in a space state, has enabled machine learning to integrate techniques from problem solving, operational research and combinatorics. The search is NP-complete for a large variety of languages of interest (e.g. [Hau89, KV94]) and heuristic search is crucial for efficiency. Whereas heuristic search has been showed to be effective in attribute-value languages, it appeared early that learning in relational languages, also known as Inductive Logic Programming (ILP), had to face important *plateau phenomena* (see e.g. [Qui91, SP91, RM92]): the evaluation function, used to prioritise nodes in the refinement graph is constant in parts of the search space, and the search goes blind. These plateau phenomena are the pathological case of heuristic search, being complete or not [Pea85].

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 6–23, 2008.

[SGS01, Alp04, AO08] pointed out that an explanation for these plateaus is the phase transition behaviour of the NP-complete subsumption test, as shown by [GBS99, GS00]. When one studies the probability of covering a random example of a fixed size by a hypothesis given the hypothesis' size, one distinguishes three well-identified regions: a under-constrained region for small hypothesis size, named "yes", where the probability of covering an example is close to 1, an over-constrained region for large hypothesis size, named "no", where the probability is close to 0, and finally in between the phase transition or the "pt" region, where an example may or may not be covered. As the heuristic value of a hypothesis depends on the number of examples covered (positive or negative), we see that the two regions "yes" and "no" represent plateaus that need to be crossed during search without an informative heuristic value.

We think that a systematic study of the impact of plateaus on heuristic search used in learning is a necessary step for the development of scaling-up relational learners, much in the line of recent advances in combinatorics through the phase transition framework.

In this paper, we propose a consistency problem generator in relational learning, RLPG, that guarantees the existence of plateaus during search, based on model RB proposed for CSP. Using its properties, it is proved that the current hypothesis' size evaluated during learning is an order parameter of the phase transition of the subsumption test. This result asymptotically guarantees the existence of a plateau for the heuristic search. Moreover, it is shown that the size of the plateau grows sub-quadratically with the problem size. In practice, we will show that problems of very small size can be generated while still guaranteeing plateaus, which makes it suitable as a benchmark model for relational learning. This is empirically validated by running several complete search learners on problems generated by RLPG that exhibit the pathological case where informed search learners degenerate into non-informed ones.

In section 2, we present the necessary background on relational learning and Constraint Satisfaction Problems (CSP). In the next section, we discuss another model proposed to import the phase transition framework into relational learning [BGSS03]. However, they tackle the different problem of studying the link between the localisation of the target concept with respect to the phase transition and the generalisation performance on a test set. This model cannot be lifted to our problem as we will discuss it. The section 4 presents the model RLPG (Relational Learning Problem Generator). Then, this generator is empirically validated in section 5 on several complete search strategies for learning, available in the learning systems Aleph [Sri99], Progol [Mug95] and Propal [AR06]. Finally, we conclude on further developments of the model RLPG.

2 Background

2.1 Relational Learning (RL)

In this article, we study what has been termed the ILP-consistency problem for function-free Horn clauses by [GLS97]. Given a set of positive examples

E^+ and a set of negative examples E^- of function-free ground Horn clauses and an integer k polynomial in $|E^+ \cup E^-|$, does there exist a non-recursive function-free Horn clause h with no more than k literals such that h logically implies each element in E^+ and h does not implies any element in E^-.

In such hypothesis space, the logical implication is equivalent to θ-subsumption which is NP-complete and therefore decidable [Got87].

Definition 1 (θ-subsumption). *Let C, D two clauses. C θ-subsumes D, noted $C \geq_\theta D$ iff there exists a substitution θ such that $C\theta \subseteq D$*

The consistency problem is fundamental in learning as it is the core of the Statistical Learning Theory, notably studied in the PAC framework (see [KV94] for details). This *a fortiori* is true in Relational Learning where almost all noise-resistant learners are relaxation of this problem [Für97].

A central idea in symbolic learning is the use of a generality partial order between hypotheses, in the hypotheses space denoted $\mathcal{L}_h$, to guide the resolution of the consistency problem (see Mitchell [Mit82] for more details). Mitchell defined top-down search and bottom-up search strategies. Without loss of generality, we restrict ourselves to top-down search. The search strategies are further refined into generate-and-test (GT) and data-driven (DD) strategies. In the GT paradigm, the top-down refinement operator, noted ρ, is only based on the structure of the hypothesis space, independently of the learning data: Let $h \in \mathcal{L}_h : \rho(h) = \{h' \in \mathcal{L}_h | h \geq h'\}$. Therefore, generate-and-test algorithms have to deal with many refinements that are not relevant with respect to the discrimination task. On the contrary, the DD strategy searches the space of hypotheses that are more general than or equal to a given positive example and uses negative examples to prune irrelevant branches in the refinement graph. It is defined as a binary operator: Let $h \in \mathcal{L}_h, e^- \in E^- :$ $\rho(h, e^-) = \{h' \in \mathcal{L}_h | h \geq h' \textit{ and } h' \not\geq e^-\}$. Relying on the negative examples allows a TDD strategy to have a branching factor that is smaller than the branching factor of a generate-and-test strategy, and can therefore compensate for a poor evaluation function by using the learning data [AR06, AO08]. The so-called *near-misses* are negative examples that reduce the branching factor to one.

2.2 Constraint Satisfaction Problems and Random Problem Generators

A Constraint Satisfaction Problem (CSP) is defined by a finite set of variables $\{X_1, \ldots, X_n\}$, a set of finite domains $\{D_1, \ldots, D_n\}$, each variable X_i taking its value from its corresponding domain D_i, and a set of constraints $\{C_1, C_2, C_3, \ldots, C_m\}$. Each constraint C_i is defined over a subset of k variables called its scope and denoted by $\mathit{scope}(C_i)$. An extensional definition of a constraint C_i is the set of tuples of values allowed for the variables in $\mathit{scope}(C_i)$. Instantiating a variable is affecting to it a value from its domain. A solution of a CSP is an assignment of all variables that satisfies all constraints. When $(\forall i)$ $|\mathit{scope}(C_i)| = 2$, the CSP is called binary.

Studies on CSP assume a model of random instance generation [HW94, SD96]. Randomly generated CSP have widely been used experimentally and theoretically to study the phase transition between the regions of under-constrained and over-constrained CSP. Most studies use one of the known models A, B, C, D (see [Smi01] for details). In each of these models, sets of randomly generated CSP were used. Each set of problems is characterised by four parameters [SD96]: a set of n variables; the number of values d in each variable domain; p_1, the proportion between the number of generated constraints and the number of possible constraints, defining the CSP density, and p_2, the proportion between the number of incompatibles tuples and the possible ones in each constraint, defining the constraint's tightness. p_1 and p_2 are order parameters to exhibit phase transitions in CSP. For instance, by fixing one of the order parameters and varying the other one from 0 to 1, one wanders from an under-constraint region, the "yes" region, where the probability of solubility is close to 1, to an over-constrained region, the "no" region, where the probability is close to 0, with the phase transition localisation depending on the parameters' value.

We detail the standard stochastic model B [SD96] and its extension model RB [XL00, XBHL07] we are going to use in the following. Model B is defined by the tuple $B(k, n, d, p_1, p_2)$, where $k \geq 2$ denotes the arity of each constraint, $n \geq 2$ the number of variables, d the domain size for all constraints, p_1 constraint density and p_2 constraint tightness. We note that in model B, the number of constraints is $m = p_1.\binom{n}{k}$, the number of disallowed tuples of each constraint is $t = p_2.d^k$. Some limitations of model B regarding the asymptotic complexity have been pointed out by [AKK+97]. They prove that random problems generated with B model suffer from trivial insoluble instances as problem size increases. Model RB, which share the same generation procedure as model B, avoids its limitations by adding constraints of the parameters' values. Model RB is denoted by $RB(k, n, \alpha, r, p)$, where k, n, p are respectively the same as k, n, p_2 in model B, α defines the domain size $d = n^\alpha$ and r defines the number of constraints $m = r.n.ln(n)$.

To generate a problem in each model, we have to build m constraints, each one formed by randomly selecting, uniformly and without replacement, a scope of k (distinct) variables and randomly selecting, uniformly and without replacement, a relation of t distinct disallowed tuples. [XL00] prove that model RB, under some conditions, avoids trivial asymptotic behaviours and provides exact phase transition thresholds for random CSP. Model RB guarantees the phase transition by varying one of the two defined parameters r and p. Note that the main difference between B and RB is that the domain size of each variable in RB grows polynomially with the number of variables. We will use the following theorem to show that the hypothesis size in relational learning is an order parameter of the phase transition.

Theorem 1 ([XL00]). *Let P_{sat} denotes a probability distribution, if k, $\alpha > \frac{1}{k}$ and $p \leq \frac{k-1}{k}$ be constants and $r_{cr} = -\alpha/ln(1-p)$ then*

$$\lim_{n\to\infty} P_{sat}[P \in RB(k, n, \alpha, r, p) \ is \ sat] = \begin{cases} 1 \, if \ r < r_{cr} \\ 0 \, if \ r > r_{cr} \end{cases}$$

This theorem indicates that a phase transition is guaranteed when the domain is not too small and the constraint tightness not too large. In the case of binary constraints ($k = 2$), the domain size is required to be greater that the squared root of the number of variables.

2.3 Reduction of Extensional CSP to θ-Subsumption

As θ-subsumption is NP-complete, various models used to study phase transition phenomena from other NP-complete problems can be imported to relational learning via reduction. Models for random CSP are easy to import in relational learning because of their trivial reduction to the subsumption problem (time and space complexity linear in the problem size), that is to decide if a variabilised function-free horn clause, C, θ-subsumes a ground function-free horn clause, D.

Following the presentation of CSP in section 2.2, C encodes the scope of the m constraints with m literals built on different predicate symbols : each constraint C_i is associated with a literal l_i such that $scope(C_i) = vars(l_i)$. By definition, C cannot have more literals than $\binom{n}{k}$. The extensional definitions of the constraints is given by D : for each constraint we define as many ground literals as there are allowed tuples in it, built from its associated predicate symbol. The size of D is then $\Sigma_i^m |C_i|$, with $|C_i|$ the cardinality of the set of allowed tuples by C_i.

We illustrate the reduction on an example. Let be a CSP defined over 3 variables with $D_1 = D_2 = \{a, b\}$ and $D_3 = \{a, b, c\}$, and 2 constraints $C_1(X_1, X_2) = \{(a, b)\}$ and $C_2(X_1, X_2, X_3) = \{(a, b, c), (b, a, a)\}$. We define C as: $c \leftarrow c1(X_1, X_2), c2(X_1, X_2, X_3)$ and D as $c \leftarrow c1(a, b), c2(a, b, c), c2(b, a, a)$. Note that the positive literal is only relevant for the learning task as it doesn't encode anything CSP specific. It is easy to see that a substitution θ, solution to the subsumption problem, is the solution tuple of the CSP and conversely. In the example $\theta = \{X_1/a, X_2/b, X_3/c\}$.

3 Related Work

A model to study the phase transition of the subsumption test has been proposed in [GS00] and the study of its possible impact on relational learning efficiency has been proposed in [GBS99, BGSS03]. We are going to discuss them and show their limitations to study the impact of the phase transition of the subsumption test on plateaus during search.

3.1 Model to Study the Phase Transition of the Subsumption Test

In [GS00], they propose a model, inspired by model B, to study the phase transition of the subsumption test. Hypotheses are function-free horn clauses from the hypothesis space $\mathcal{L}_h^m$ built as follow:

$$\mathcal{L}_h^m = \{c \leftarrow \bigwedge_{k=1}^{n-1} p_{l_k}(X_k, X_{k+1}) \wedge \bigwedge_{k=n}^{m} p_{l_k}(X_{i_k}, X_{j_k})\}$$

where c is the clause head without variables, $i_k < j_k \in \{1, \ldots, n\}, l_k \in \{1, \ldots, m\}$ and such that all literals in the clause body are built on distinct binary predicate symbols. The first $n-1$ literals ensure that all variables are linked, and therefore, that the set of variables cannot be decomposed into sets of independent variables that can break the subsumption test into easier sub-problems [GS00]. That is to say, p_1, the constraint density, can not be fewer than $p_{1min} = 2/n$. Examples are represented as ground clauses as described earlier to encode the constraints' domains. They showed that m, the hypothesis size, and L, the domain size, were order parameters of the phase transition in their settings. However, there is no guarantee that a phase transition will occur as their model differs from random CSP models. p_1 and p_2, the order parameters in most CSP models, are random variables here, depending on m and L. Indeed, the variables pairs are randomly drawn with replacement and several literals can be built on the same pair of variables. The constraint on the variables is no longer a set of tuples built on a unique predicate symbol, but the intersection of all sets of tuples related to the literals. For instance, let $n = 4$, then the maximal number of constraints in a binary CSP is $m_{max} = 6$. Let $m = m_{max}$ and the two following hypotheses:

$$h_1 : c \leftarrow p_1(X,Y), p_2(Y,Z), p_3(Z,T), p_4(X,Y), p_5(X,Y), p_6(X,Y)$$

$$h_2 : c \leftarrow p_1(X,Y), p_2(Y,Z), p_3(Z,T), p_4(X,Z), p_5(X,T), p_6(Y,T)$$

We remark that in h_1, the order parameter $p_1 = p_{1min}$ and $p2$ is undefined, and in h_2, the order parameter $p_1 = 1$ and $p_2 = 1 - N/L^2$.
To exhibit the phase transition of subsumption, we propose to use model RB, where phase transition is proved to occur asymptotically and can be precisely located.

3.2 Phase Transition of Subsumption and Relational Learning

Bringing the phase transition framework to the realm of Relational Learning has been first done by Giordana et al. [GBS99, BGSS03] where they proposed to study the link between the localisation of the target concept with respect to the phase transition of the subsumption test and the generalisation performance on a test set. They tackle the learning of a function-free horn clause from the hypothesis space $\mathcal{L}_h^m$ described above. A learning problem is parametrised with the pair (m, L), the number of variables n being fixed to 4 and the number of allowed tuples N is fixed to 100 in their experiments. In a (m, L) problem, m is the size of the target concept drawn from $\mathcal{L}_h^m$ and L the number of constants in the examples. For each problem, a learning set and a test set are built to evaluate generalisation performance of learning algorithms. Both are balanced sets of examples with 100 positive examples and 100 negative examples. It has to be noted that if (m, L) lies in the "yes" (resp. "no") region, by construction the concept description will almost surely cover (resp. reject) any randomly constructed example. For those problems, the example generator is modified and relies on a repair mechanism to ensure a balanced distribution of positive and negative examples [BGSS03].

It was shown in [AO08], however, that this localisation of the concept was not a reliable indication of the learning problem difficulty, and that plateaus generated by the phase transition behaviour of the subsumption test prevented FOIL from solving any problem.

Their model cannot be lifted to our study of the link between plateaus and heuristic search efficiency. Notably, we can note that the hypothesis space is very large in their settings and prohibits the study of average heuristic search behaviour of complete learners and even incomplete learners. There is also no guarantee that plateaus, through the occurrence of a phase transition, will occur with smaller parameter values, as their model differs from random CSP models as stated above, but also as a balanced distribution of the examples is ensured by a repair mechanism and is no longer random.
Also, the proposed problem generator does not translate into lattice-like hypothesis space (the target concept being one of the most specific elements) and the link between variables is a hidden structure. This point is important to be able to easily run standard learning approaches on generated problems. For instance, this prevents the use of popular approaches based on the existence of a bottom-most element in the search space, like top-down seed-based approaches (Aleph, Progol, Propal) and bottom-up approaches, like lgg-based approaches [Plo70].

We introduce in the next section a model to analyse phase transition and plateau phenomena in relational learning. It guarantees plateaus during search in problems of very small size, suitable to evaluate the average performance of learners, and defines the hypothesis space as a boolean lattice to ease the implementation of various learning strategies.

4 Model for Exhibiting Plateaus in Random RL Problems

A learning problem instance in our model is denoted $RLPG(k, n, \alpha, N, Pos, Neg)$. The parameters k, n, α are the same parameters as in RB. N is defined as in [GBS99] as the number of allowed tuples by each constraint and we have $N = (1-p).d^k$, with p the constraint tightness. As they argued, this is more meaningful for learning as it is the number of literals built on the predicate symbol associated to a given constraint. Pos and Neg are the number of positive and negative examples in the learning dataset, respectively.

Given k and n, the maximum number of constraints is $\binom{n}{k}$. All these constraints are encoded in a clause which is set as the bottom clause of the hypothesis space $\mathcal{L}_h$. $\mathcal{L}_h$ is then defined as the power set of the bottom clause, which is isomorphic to a boolean lattice. As said previously, this property is interesting for learning because it eases the implementation of various learning strategies like bottom-up generate-and-test and data-driven (i.e. lgg based [Plo70]) strategies. Also, this restriction is often used in top-down learning systems like Aleph, Progol or Propal to define complete and efficient refinement operators. In that space, it is guaranteed that each hypothesis evaluated by the learning algorithms encodes a valid constraint network of the underlying model RB. The refinement operator is to add a literal from the bottom clause that is not in the hypothesis, hence the number of literals in the hypothesis is exactly m, the number of constraints in the underlying CSP.

Learning examples are randomly drawn, independently and identically distributed, given n, α and N, as explained in section 2.2. Their size is $N.\binom{n}{k}$. Each example defines the set of allowed tuples of size N for possible constraint networks ranging from 0 to $\binom{n}{k}$ constraints.

In the next section, we detail how this model exhibits a phase transition of the subsumption test when varying hypothesis sizes, and in section 4.2 how this phase transition translates into a plateau for the heuristic search, during the resolution of the consistency problem. From now on, we restrict ourselves to binary CSP, that is all logical predicates are binary. As in [BGSS03], this restriction is for the sake of simplicity while still being representative of typical relational learning problems.

4.1 The Phase Transition of the Subsumption Problem

Given a randomly drawn example according to model RLPG, a hypothesis of size m defines m constraints over n variables, each constraint being extensionally defined in the example. As the hypothesis size m varies during search from 1 to $n(n-1)/2$ ($k = 2$ here), $r = m/(nln(n))$, the order parameter of the phase transition, varies (or equivalently in model B, $p1$ varies from 0 to 1). In model RB, there exists an exact localisation of the phase transition for $r = r_{cr}$. As we use the same model, varying m, the hypothesis size of the current hypothesis asymptotically exhibits a phase transition.

[XL00] give an asymptotic value of the cross-over point of the phase transition as $r_{cr} = -\alpha/ln(1-p)$ (theorem 1). This critical value is the point where the expected number of solutions of the problem $E(N) = 1$. In practice, this is often

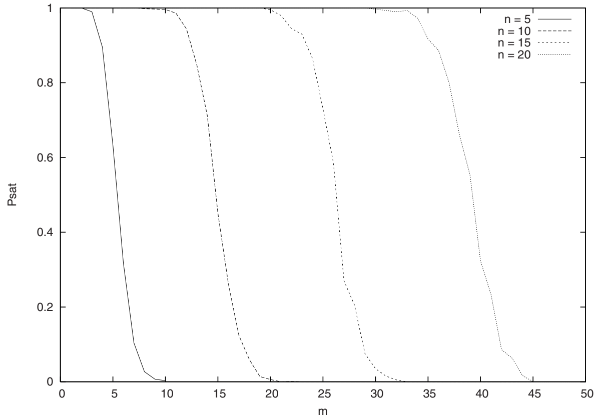


Fig. 1. P_{sat} drawn for different values of n ($d = n^{\alpha}$ with $\alpha = 1$, $p = 0.75$) as m varies, averaged over 300 drawn (hypothesis,example) pairs

used to localise the critical value of the order parameter where $P_{sat} = 0.5$ (see e.g. [SD96]). In our case, the critical hypothesis size is $m_{cr} = -\frac{\alpha n ln(n)}{ln(1-p)}$. We can see that as n increases, the value of m_{cr} grows in $nln(n)$. Interestingly, even when n is small (starting from 5), experiments corroborate the theoretical results as shown in figure 1. Note that for each plot, the maximum number of possible literals is $n(n-1)/2$.

Empirical validation of the localisation of the crossover point regarding to the order parameter m is summarised in table 1. We observe that for all values of n, greater than 5, the empirical value of the crossover point m_{cr} is close to the theoretical asymptotic value. Even when the conditions of theorem 1 are not respected ($p = 0.75$ in Figure 1), we observe that $P_{sat} = 1$ almost surely for $m < m_{cr}$. It was noted already in [Smi01] that the conditions attached to α and p are strong and it should be possible to relax them by using other methods to compute P_{sat} lower bounds. This robustness allows us to define small problems where the phase transition is exhibited, which translates into small hypothesis spaces as we will see in the next section.

Table 1. Comparison between empirical and theoretical values of m for $\alpha = 1, p = .75$

n	m_{TP}	empirical value of m
5	5,79 ± 1	between 5 and 6
10	16,61 ± 1	between 14 and 15
15	29,30 ± 1	between 26 and 27
20	43,21 ± 1	between 39 and 40

We show the phase transition along the second order parameter p of model RB in figure 2, with $\alpha = 1.4$ where the contour plots correspond to $P_{sat} = 0.99, 0.5$ and 0.01, and in figure 3, where the contour plot corresponds to $P_{sat} = 0.5$, with different values of α. Each point is averaged over 1000 subsumption tests. p is controlled with N in RLPG and we observe that as N increases, p decreases which gives smaller values for m_{cr}.

p can also be controlled by varying α, keeping N constant, as in [GS00] who used the domain size as order parameter, which is pictured in figure 4. We are going to use this control parameter in the next section to change the localisation of the "pt" region and therefore to change the plateau length of a problem in the next section. Although this is not strictly model RB, the advantage of $d = n^\alpha$ here to control p instead of N is that the examples' size does not change and it keeps learning problem size constant. However, p changes faster, quadratically in d instead of linearly in N, as it can be seen in the figure.

4.2 The Plateaus of Heuristic Search

A relational learning problem is defined by drawing, independently and identically distributed, Pos positive examples and Neg negative examples. By construction, a hypothesis, solution of the consistency problem, can only be found

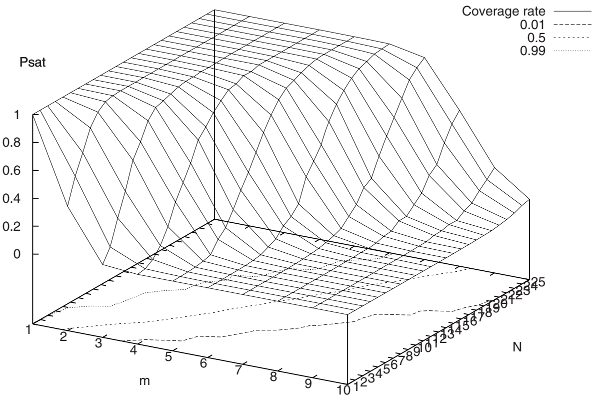


Fig. 2. Probability of solubility of the subsumption test in the (m, N) plane for $n = 5$ and $\alpha = 1.4$

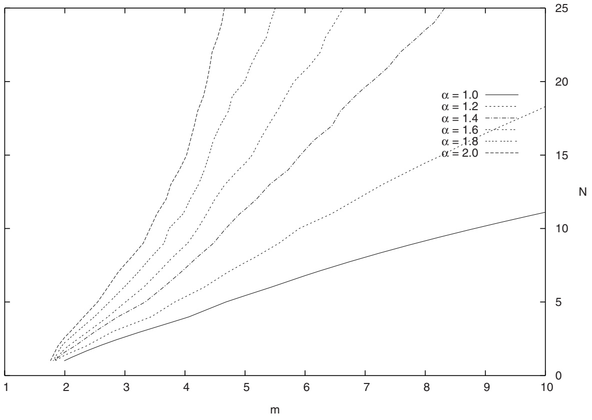


Fig. 3. Contour plot corresponding to $P_{sat} = 0.5$ in the (m, N) plane for $n = 5$ and various values of α

in the phase transition region, as hypotheses in the "yes" (resp. the "no") will almost surely subsume (resp. not subsume) all learning examples. The top-down search, which starts from the top-most hypothesis in the hypothesis space lattice, will have to specialise hypotheses by adding a literal at a time to cross the "yes" region before reaching the phase transition region where a solution can be expected. Dually, a bottom-up search will have to cross the "no" region before reaching the "pt" region.

The "yes" and "no" regions implies a plateau to cross during a heuristic search in these regions. Indeed, if we study the state of the art on evaluation functions used in learning (see for instance [FF03]), it shows that all of them are based, without loss of generality, on three parameters that are the coverage rate of positive examples, the coverage rate of negative examples and possibly

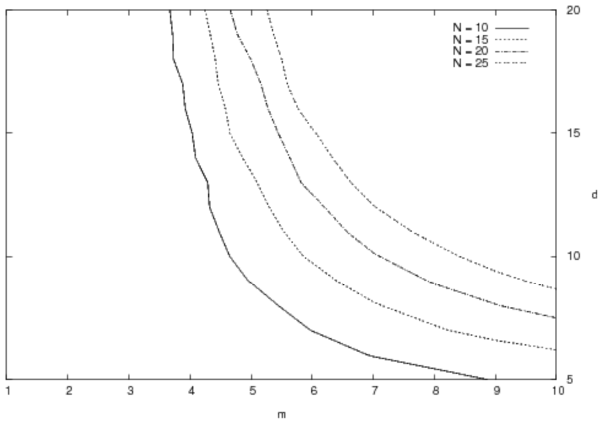


Fig. 4. Contour plot corresponding to $P_{sat} = 0.5$ in the (m, d) plane for $n = 5$ and various values of N

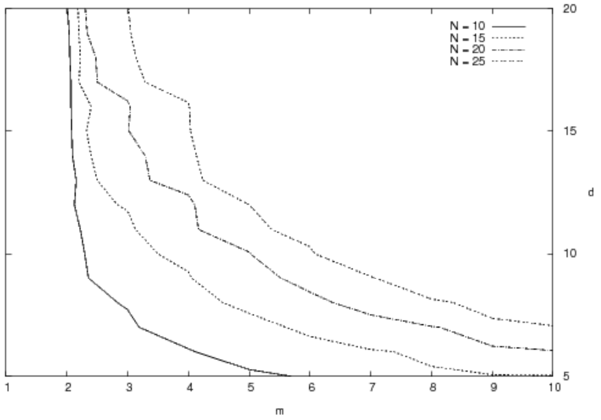


Fig. 5. Contour plot corresponding to plateaus size: $P_{sat} \geq 0.99$ in the (m, d) plane for n=5 and various values of N

a complexity measure of the hypothesis under consideration. In the "yes" or "no" regions, the coverage rate of positive and negative examples is constant and implies that all evaluation functions are constant, defining a plateau. It has to be noted that, as the first two parameters are inherited from the learning task definition, it is unlikely that a solution for solving the plateau problem consists in designing new evaluations functions.

We show the plateaus for a top-down approach in figure 5. It shows the contour plot corresponding to $P_{sat} = 0.99$[1] for problems defined with $n = 5$ variables and different values of N. P_{sat} is evaluated on average at each point in the (m, d) plan

[1] Any value arbitrarily closer to 1.0 gives the same result.

by running 1000 subsumption tests. These contour plots indicate the beginning of the "pt" region to the right, the left part, for low m, being the plateau in the "yes" region. We see that varying d the number of constants in the example, we change the tightness of the constraints which varies the plateau length.

Various experiments have been conducted with different parameter values which show all similar plateau profiles. It is interesting to note that the smallest value for n where plateaus were exhibited is $n = 5$. In this case, the bottom-most hypothesis in the lattice has a size of $n(n-1)/2 = 10$, and therefore the plateaus are exhibited for a hypothesis space of 2^{10} hypotheses only. This is a very small problem size which makes it very useful for studying average performances of complete search learners in reasonable time. We are going to use it in the next section to compare the behaviours of different complete informed and non-informed search learners.

5 Experimental Results

Complete search learners, available in the learning systems Aleph, Progol and Propal are run on a collection of problems denoted by $RLPG(2, 5, d, 15, Pos, Neg)$, with d varying from 5 to 20, and $Pos = Neg$ varying from 1 to 5. We evaluate the impact of the plateaus on their heuristic search cost by recording the number of evaluated hypotheses to answer the consistency problem. Every plot is averaged over 1000 randomly drawn learning problems. As said previously, we limit ourselves to top-down approaches. For a detailed description of the various strategies we discuss below, we refer to [Pea85, Sri99, Mug95, AR06] because of the space requirements of the paper.

As non-informed searches, we use the breadth-first TGT search (BF-TGT) and the depth-first TGT search (DF-TGT). As informed searches, we use the A TGT search (A-TGT) and the best-first TGT search (BESTF-TGT). Informed

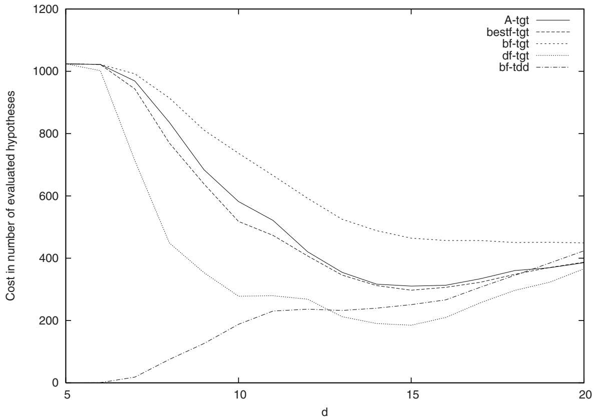


Fig. 6. Median number of evaluated hypotheses during search for learning problems with Pos = Neg = 3 and d varying from 5 to 20

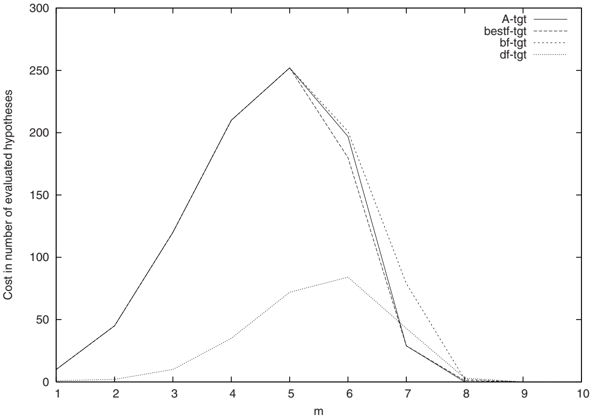


Fig. 7. Median number of evaluated hypotheses during search according to their size, with $d = 8$

search makes use of an evaluation function to minimise, whose general form is $f = g + h$. g is defined as the cost from the start to the current hypothesis and h as an estimation of the distance from the current hypothesis to the goal. We define A-TGT according to the Progol system: g is defined as the length of the current hypothesis and h has the difference between the number of negative examples and the number of positive examples. In our context, as all positive examples must be subsumed, it simplifies to the number of negative examples. BESTF-TGT is not biased towards shorter hypotheses and defines $g = 0$.

In all these complete strategies, the implemented refinement operator structures the hypothesis space as a tree in order to avoid redundancy during search. This is done classically with boolean lattices by fixing a total order on the possible refinements.

The last learning strategy we study is the one used in the TDD learner Propal. This is an incomplete learner as it performs a beam search guided by the Laplace function. So we set Propal with a beam of unlimited size, which basically turns down to a non-informed breadth-first search (BF-TDD). The only difference is that when the solution is reached at a level of the search, it will be the first picked up at the next level. Note also that, as an incomplete learner, it does not have an optimal refinement operator, like the other learners, and may evaluate the same hypothesis several times.

We only show results obtained when $Pos = Neg = 1, 3$ and 5, as extensive experiments varying the number of examples exhibited the same patterns. Figure 6 shows the median cost of the different learning strategies for $Pos = Neg = 3$ with various plateau lengths defined by d (as illustrated in figure 5). We can notice several patterns for the learners depending on the plateau lengths.

We first compare BF-TGT and BF-TDD. BF-TDD outperforms its generate-and-test counter-part, as it has been shown in [AR06] that the TDD approach necessarily has a branching factor smaller or equal to that of a TGT strategy.

When $d = 5$, the plateau is the largest and no consistent hypothesis exists. BF-TGT has to evaluate all the hypotheses (1024) before answering the problem. As d increases, the plateau length decreases and BF-TGT will develop all hypotheses up to the phase transition region where a hypothesis can be found. As for BF-TDD, the computation of a near-miss with the bottom of the search space in the case of $d = 5$ yields an inconsistency as the near-miss is equal to the bottom-most hypothesis and the search halts with no evaluated hypotheses. As d increases, near-misses will be farther from the bottom of the search space and the branching factor will increase. This is pictured by an increasing number of evaluated hypotheses, which converge towards the cost of BF-TGT. The other non-informed learner, DF-TGT, shows the same behaviour as BF-TGT for low values of d as it cannot detect trivial inconsistency. When d is above 13, it performs best as there are several solutions in the "pt" region. DF-TGT directly crosses the plateau and ends up doing few backtracks before finding a consistent hypothesis.

We discuss now the informed strategies. We can see that they both have a similar behaviour, and mimic BF-TGT, although they evaluate fewer hypotheses than it for smaller plateaus. However, they systematically evaluate more hypotheses than DF-TGT, and than BF-TDD, except for the largest values of d. This behaviour is the pathological case of an informed search. As an illustration, we plot the number of hypotheses evaluated according to their size, for $d = 8$ in figure 7, where the plateau is large, and for $d = 15$ in figure 8, where the plateau is smaller. We can see that they all develop all hypotheses up to $m = 5$ for $d = 8$, given a plateau size of 4 in figure 5, and up to $m = 3$ for $d = 15$, given a plateau size of 2. It is only in the "pt" region that the heuristic becomes useful to guide the search and differentiates the different approaches. The fact that each time m is one literal bigger than the plateau size is not clear. We can note that BESTF-TGT systematically outperforms A-TGT, even slightly, as A-TGT will

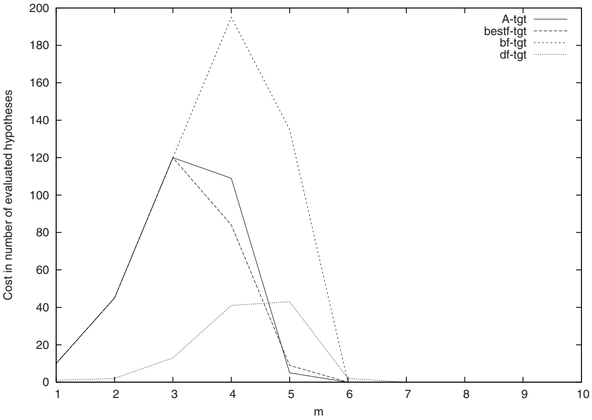


Fig. 8. Median number of evaluated hypotheses during search according to their size, with $d = 15$

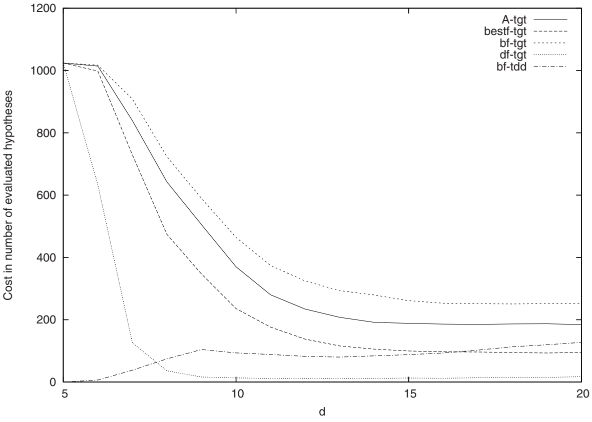


Fig. 9. Median number of evaluated hypotheses during search for learning problems with Pos = Neg = 1 and d varying from 5 to 20

give preference to a hypothesis that subsumes 2 negative examples compared to a hypothesis one literal longer that subsumes only 1 negative example.

We can note that the cost of resolution, after decreasing up to $d = 15$ for the GT learners, starts increasing after this point. The problems seem harder for BF-TDD too. This point corresponds to problems that have a probability of solubility close to 0.5 and would correspond to the phase transition region of the consistency problem. As noted in [AO08], learners solving the ILP-consistency problem potentially have to face two phase transitions: the phase transition of the NP-complete subsumption test and the phase transition of NP-complete search. It is a very interesting follow-up to study how the parameters of RLPG can exhibit this second phase transition as the probability of solubility of the

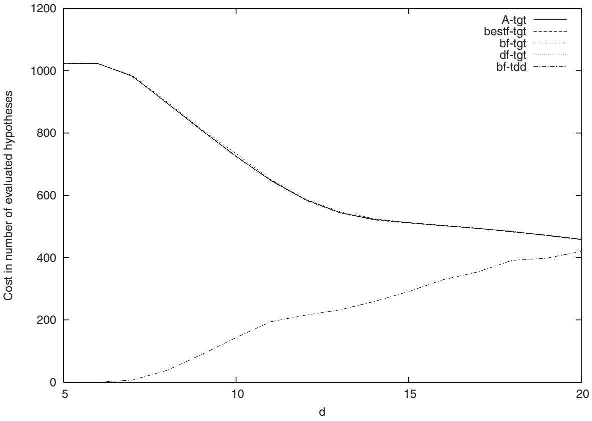


Fig. 10. Median number of evaluated hypotheses during search for learning problems with Pos = Neg = 5 and d varying from 5 to 20

consistency problem may impact the behaviour of learning strategies. Figure 9 shows problems generated with $Pos = Neg = 1$, where the number of solutions is high. In this case, we observe the same behaviour of algorithms as in figure 6, with a notable difference for DF-TGT which quickly reaches the "pt" region and finds a solution with almost no backtrack.

Figure 10 shows the median cost of the different learning strategies in problems with $Pos = Neg = 5$ where almost no solution exists and all learners have to search the entire space. We observe that all informed and non-informed TGT approaches have the same high median cost as they cannot efficiently detect inconsistency as opposed to the TDD approach for all values of d.

6 Conclusion

Recent works in relational learning pointed out that the phase transition phenomenon, which may occur in the subsumption test during search, acts as a plateau for the heuristic search, strongly hindering its efficiency [AO08]. We proposed to systematically investigate this issue by designing a relational learning problem generator where it is shown that top-down and bottom-up learning strategies face a plateau during search before reaching a solution. This property is ensured by using the generative model RB defined for CSP to exhibit the phase transition of the subsumption test, with the hypothesis size as an order parameter. The size of the plateau grows sub-quadratically with the problem size and it is guaranteed asymptotically. Intensive experiments show that even for small problems the asymptotic model of RLPG still holds. This feature allows to study a wide range of algorithms in reasonable time and is therefore suitable as a benchmark model. At the end of the paper, we have presented preliminary results with various complete learners and interesting behaviours have been pointed out. Notably, it has been shown, as a validation, that RLPG exhibited the pathological case where informed search degenerates into non-informed one when facing plateaus.

Finally, this model points out interesting follow-ups. We plan to further study the properties of generated problems depending on RLPG's parameters; implement and compare other learning strategies, notably bottom-up, to exhibit characteristic behaviours to help design better heuristic approaches for relation learning.

References

[AKK$^+$97] Achlioptas, D., Kirousis, L.M., Kranakis, E., Krizanc, D., Molloy, M.S.O., Stamatiou, Y.C.: Random constraint satisfaction: A more accurate picture. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 107–120. Springer, Heidelberg (1997)

[Alp04] Alphonse, E.: Macro-operators revisited in inductive logic programming. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 8–25. Springer, Heidelberg (2004)

[AO08] Alphonse, E., Osmani, A.: On the connection between the phase transition of the covering test and the learning success rate. Machine Learning Journal 70(2-3), 135–150 (2008)

[AR06] Alphonse, E., Rouveirol, C.: Extension of the top-down data-driven strategy to ILP. In: Muggleton, S., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 49–63. Springer, Heidelberg (2007)

[BGSS03] Botta, M., Giordana, A., Saitta, L., Sebag, M.: Relational learning as search in a critical region. Journal of Machine Learning Research 4, 431–463 (2003)

[FF03] Fürnkranz, J., Flach, P.: An analysis of rule evaluation metrics. In: Proc. 20th International Conference on Machine Learning, pp. 202–209. AAAI Press, Menlo Park (2003)

[Für97] Fürnkranz, J.: Pruning algorithms for rule learning. Mach. Learn. 27(2), 139–172 (1997)

[GBS99] Giordana, A., Botta, M., Saitta, L.: An experimental study of phase transitions in matching. In: Proc. of the 16th International Joint Conference on Artificial Intelligence, pp. 1198–1203

[GLS97] Gottlob, G., Leone, N., Scarcello, F.: On the complexity of some inductive logic programming problems. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 17–32. Springer, Heidelberg (1997)

[Got87] Gottlob, G.: Subsumption and implication. Information Processing Letters 24(2), 109–111 (1987)

[GS00] Giordana, A., Saitta, L.: Phase transitions in learning relations. Machine Learning 41, 217–225 (2000)

[Hau89] Haussler, D.: Learning conjunctive concepts in structural domains. Machine Learning 4(1), 7–40 (1989)

[HW94] Hogg, T., Williams, C.P.: The hardest constraint problems: A double phase transition. Artificial Intelligence 69(1–2), 359–377 (1994)

[KV94] Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. The MIT Press, Cambridge (1994)

[Mit82] Mitchell, T.M.: Generalization as search. Artificial Intelligence 18, 203–226 (1982)

[Mug95] Muggleton, S.: Inverse entailment and PROGOL. New Generation Computing 13, 245–286 (1995)

[Pea85] Pearl, J.: Heuristics. Addison-Wesley, Reading (1985)

[Plo70] Plotkin, G.: A note on inductive generalization. In: Machine Intelligence, pp. 153–163. Edinburgh University Press (1970)

[Qui91] Quinlan, J.R.: Determining literals in inductive logic programming. In: Proc. of the 12th International Joint Conference on Artificial Intelligence, Sydney, New South Wales, Australia, pp. 746–750. Springer, Heidelberg (1991)

[RM92] Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: Proc. of the Tenth National Conference on Artificial Intelligence, San Jose, California, pp. 723–738. AAAI Press/MIT Press (1992)

[SD96] Smith, B.M., Dyer, M.E.: Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence 81(1-2), 155–181 (1996)

[SGS01] Serra, A., Giordana, A., Saitta, L.: Learning on the phase transition edge. In: Nebel, B. (ed.) Proc. of the 7th Int. Conf. on Artificial Intelligence, pp. 921–926

[Smi01] Smith, B.M.: Constructing an asymptotic phase transition in random binary constraint satisfaction problems. Theoretical Computer Science 265(1–2), 265–283 (2001)

[SP91] Silverstein, G., Pazzani, M.J.: Relational cliches: Constraining constructive induction during relational learning. In: Proc. of the 8th Int. Workshop on Machine Learning, pp. 203–207. Morgan Kaufmann, San Francisco (1991)

[Sri99] Srinivasan, A.: A learning engine for proposing hypotheses (Aleph) (1999), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph

[XBHL07] Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random constraint satisfaction: Easy generation of hard (satisfiable) instances. Artif. Intell. 171(8-9), 514–534 (2007)

[XL00] Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. J. Artif. Intell. Res (JAIR) 12, 93–103 (2000)

Top-Down Induction of Relational Model Trees in Multi-instance Learning

Annalisa Appice, Michelangelo Ceci, and Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari
via Orabona, 4 - 70126 Bari - Italy
{appice,ceci,malerba}@di.uniba.it

Abstract. (Multi-)relational regression consists of predicting continuous response of target objects called reference objects by taking into account interactions with other objects called task-relevant objects. In relational databases, reference objects and task-relevant objects are stored in distinct data relations. Interactions between objects are expressed by means of (many-to-one) foreign key constraints which may allow linking explanatory variables of a task-relevant object in several alternative ways to the response variable. By materializing multiple assignments in distinct attribute-value vectors, a reference object is represented as a bag of multiple instances, although there is only one response value for the entire bag. This works points out the same assumption of multi-instance learning that is a primary instance is responsible for the observed response value of a reference object. We propose a top-down induction multi-relational model tree system which navigates foreign key constraints according to a divide-and-conquer strategy, derives a representation of reference objects as bags of attribute-value vectors and then, for each bag, constructs a primary instance as main responsible of the response value. Coefficients of local hyperplane are estimated in an EM implementation of the stepwise least square regression. Experiments confirm the improved accuracy of our proposal with respect to traditional attribute-value and relational model tree learners.

1 Introduction

Regression has received significant attention in supervised learning, where training data consist of observations of an unknown continuous function f, and the learning task is to learn a general model g that is close to f on training data and can be subsequently used to reliably predict on new unlabeled observations. Although regression task is most popular with attribute-value learning, several extensions have already been investigated in multi-relational data mining [11], where data is expected to be spread in several database relations.

Handling relational data adds significant difficulties to the regression task since data stored in distinct database relations describe objects of different type. These objects play different roles, and it is necessary to distinguish between the *reference* (or target) *objects*, which are the main subject of analysis, and the *task-relevant objects* (or non-target objects), which are related to the former and can

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 24–41, 2008.

contribute to account for the variation. The response variable of a regression model is a continuous property of the reference objects, while the explanatory variables of the model can be associated to both reference and task-relevant objects. Foreign key constraints model "many-to-one" relationships according to which several task-relevant objects may be associated to the same reference object (non-determinacy) [15]. Since explanatory variables of task-relevant objects, namely non-determinate variables, can be linked to the response variable in several alternative ways, it is necessary to establish how the response value should be estimated due to the multiple instances that are possible for one reference object when taking into account the task-relevant objects. Deriving multi-instance data for a reference object boils down multi-relational regression to a generalization of the *multi-instance* learning.

Multi-instance learning was firstly defined in [6] to deal with predictive tasks where training data consist of bags of several instances and the response value is assigned to the entire bag. In the original proposal, all instances, which are represented by attribute-value vectors, are used for training. However, in many cases not all instances are responsible for the observed value, thus aggravating problems due to noisy and irrelevant observations. To overcome these limitations, several regression methods developed in multi-instance settings base their prediction on only one (or few) instance(s) of the bag. The validity of these methods has been empirically proved in several applications ranging from drug design [6] to invitro fertilization [19] and image analysis [16].

In this work, we follow the same approach to solve the multi-instance learning problem generated from multi-relational data, and we base the prediction on the primary instance (or prototype) selected for each reference object. In particular, we propose a novel multi-relational learner, called MIRT (*M*ulti *I*nstance *R*elation model *T*ree *I*nduction), which follows a multi-instance approach to learning regression models from multi-relational data. MIRT works under the common hypothesis that the underlying function f is a linear model with Gaussian noise on the value of the response variable. To avoid the *a priori* definition of a global form for the regression function [13], the function f is approximated by means of a tree-structured regression model, called *model tree*. This is built by recursively partitioning the set of reference objects according to a computationally efficient divide-and-conquer search strategy. An internal node can either perform a binary test on some explanatory variable or instantiate a new relation on the basis of some foreign key constraint, while a leaf is tagged with a multiple linear function (or hyperplane) g built over a multi-instance representation of the reference objects at the leaf.

Our proposal permits to compute simultaneously both primary instances and regression coefficients together by means of Expectation Maximization (EM) algorithm which minimizes the least square error of a multiple linear function learned in a stepwise fashion [8]. In this way, the final relational model tree reveals local linear dependences between response variable and explanatory ones without suffering of the presence of noise and outliers. The tree can then be used to predict the unknown response of any new reference object (test), whose primary instance is constructed by choosing the binding of the non-determinate variables involved in the local g that minimizes the Euclidean distance from the training primary instances stored in the leaf.

The paper is organized as follows. In the next Section, we discuss background and motivations of this work. The learning problem is formally defined in Section 3. In Section 4, we present a novel method that enriches the top-down induction of a relational model tree with an EM iterative approach to compute primary instances and regression coefficients. Lastly, experimental results are reported in Section 5 and some conclusions are drawn.

2 Background and Motivation

Related research on regression in multi-relational data mining and multi-instance learning are reported below.

2.1 Background in Multi-relational Data Mining

Multi-relational data mining methods for regression can be classified into two alternatives: propositional and structural (or relational).

The propositional approach constructs features which capture relational properties of data and transforms original relational data into a single attribute-value representation. The non-determinacy issue is dealt with by deriving boolean features (e.g. there exists at least a molecule conformation that includes an atom with charge greater than 2.5) or aggregate features (the average charge of atoms involved in the conformations of the same molecule). The resulting representation can then be input to a wide range of robust and well-known conventional regression methods which operate on an attribute-value single instance representation [12].

The structural approach provides functionalities to navigate relational structures in its original format and to generate potentially new forms of evidence not readily available in a flattened single table representation. The whole hypothesis space is directly explored by the mining method. Structural regression methods [14,2,1,22,9] are generally obtained by upgrading propositional learners, e.g. regression trees and model trees, to the multi-relational setting. Although several of these methods assume that the function underlying data is a (local) hyperplane, only few of them [1,22] allow regression functions which include non-determinate explanatory variables. These methods use the several instances of a reference object in training and employ aggregate functions after or before learning coefficients of local hyperplanes. In [1], multiple instances are dealt as separate instances and coefficients of hyperplane are computed by solving least square regression in the derived single instance learning. Multiple predictions derived for the same reference object are aggregated by resorting to the average function. In [22], user-defined aggregate functions allow to aggregate a non-determinate variable before adding the variable to the linear model, thus avoiding the problem of multiple predictions. Anyway the use of aggregates may suffer from problems of information loss and statistical significance when in presence of noise or outliers.

2.2 Background in Multi-instance Learning

Although multi-instance learning was initially defined for classification [6], some regression methods have been developed for the multi-instance setting.

A seminal work is in [18], where Ray and Page have assumed that the function f underlying multi-instance data is a linear model and there is one instance for each bag that is the main responsible of the real-valued response value. They have proposed an EM-based iterative method to determine primary instances and regression coefficients of the hyperplane that minimizes the least square error. Regression coefficient are computed for all explanatory variables. The hyperplane can then be used to predict the unknown response of a new object represented by a bag of multiple instances. The problem of determining the test primary instance is naively solved by selecting the first instance stored in the bag. Zhang and GoldmanIn [23] have combined the EM algorithm with a diverse density based algorithm. A two-step gradient ascent search of the standard diverse density algorithm is used to select the primary instances which maximize diverse density value. Dooly et al. [7] have proposed a multiple-instance variant of unweighted k-NN in which the distance between bags is defined as the minimum Euclidean distance between points in the two bags. The regression method investigated in [5] bypasses the problem of choosing a primary instance by computing a nonlinear weighted average of all response values. However, this is justified by the specific task of predicting the molecule activity. A gradient-based optimization procedure is used to determine the best linear model.

The multi-instance approach to relational regression task has also been considered in multi-relational data mining. Srinivasan and Camacho [20] have firstly recognized the multi-instance nature of a relational regression task when learning relational clauses. However no solution was proposed for the issue of binding those non-determinate variables which are the responsible for a response value. Several bindings are either treated as data points for regression analysis or are grouped in a single aggregated value. An important contribution to interpreting multi-relational data mining problems as multi-instance learning problems comes from Blockeel et al. [3], who have upgraded decision tree learning to the multi-instance framework in ILP.

3 Problem Statement: Relational vs. Multi-instance

The relational regression task can be formally defined as follows:

Given:

1. a set S of reference objects, that is, the target objects of the analysis;
2. some sets R_i of task-relevant objects;

3. a set I of interactions between reference objects and/or task-relevant objects;
4. a response value $y \in \mathbb{R}$ which tags each reference object in S and assumes value according to an unknown function $f : \langle S, \bigcup R_i, I \rangle \rightarrow \mathbb{R}$.

Find a function g that is hopefully close to f on the domain $\langle S, \bigcup R_i, I \rangle$.

The set S is stored in one data relation (namely target table) of a relational database D, i.e. one tuple for each reference object. Similarly, each set R_i is stored in a distinct data relation of D (one table for each object type). The interactions I are expressed by the foreign key constraints (FK) which state how tuples in one data relation relate to tuples in another relation. Foreign key constraints FK allow navigating D and retrieving all the task-relevant objects which are *related* to a reference object and thus potentially relevant to predict the value of Y. The definition of the task-relevant objects which are *related* to a reference object according to a *foreign key path* is formally provided in the following.

Definition 1. *A task-relevant object $tro \in R_i$ is related to a reference object $ro \in S$ if and only if:*

1. *there exists a foreign key constraint $fk \in FK$ from R_i to S (or vice-versa) such that the foreign key of tro assumes the same value of the primary key of ro (or vice-versa), or*
2. *there exists a task-relevant object $newTro \in R_j$ such that $newTro$ is related to ro and there exists a foreign key constraint fk from R_j to R_i (or vice-versa) such that the foreign key of $newTro$ assumes the same value of the primary key of tro (or vice-versa).*

The sequence of foreign key constraints $fkp = \langle fk_1, \ldots, fk_n \rangle$ according to a task-relevant object is related to a reference object is called foreign key path.

A foreign key path provides the schema of the attribute-value vectors which can be constructed by performing the natural join between data relations involved in the foreign key path and projecting over the explanatory variables. Grouping the attribute-value vectors referring to the same reference object allows to represent a reference object as a bag of multiple attribute-value vectors. In this case, the difference with the original formulation of a multi-instance task is that a relational bag may include attribute-value vectors with different attribute schema (one schema for each foreign key path). Independently from this difference, the multi-instance form of relational data poses the same difficulties due to noise and presence of irrelevant instances as traditional multi-instance data. This motivates our focus on the problem of selecting the main responsible of the response values. This attempt corresponds to reformulate the relational regression goal as follows:

Find (i) the primary instance of each reference object by choosing the best binding for non-determinate variables of possibly related task-relevant object, and (ii) a regression function g that is hopefully close to f on the retrieved primary instances.

4 Multi-instance Induction of Relational Model Trees

MIRT induces a multi-relational model tree by recursively partitioning the reference objects stored in the target table of a relational database and associating different hyperplanes to the separate partitions. Partitioning takes into account the relational arrangement of task-relevant objects stored in the same relational database. Each hyperplane is a linear combination of a subset of the explanatory variables belonging to several data relations in the splitting tests along the unique tree path connecting the root to the leaf.

The approach adopted to estimate the regression coefficients is where MIRT differs from the state-of-art relational model tree learners. MIRT neither resorts to aggregate nor considers multiple binding values as separate instances, but it minimizes the least square error over the primary instances constructed from the training attribute-value vectors falling in the leaf. Under the restriction that only the attribute-value vectors which satisfy the intensional description of the partition at hand are constructed, the primary instances are obtained in an EM-based formulation of the stepwise regression that i) chooses at each step the best explanatory variable to be introduced in the hyperplane, ii) identifies the best binding for this variable among the possible ones in the bag iii) uses these best bindings to compute an estimate of the regression coefficient of this variable in the hyperplane. The primary instances constructed in the training are stored in each leaf in order to provide a baseline to construct the primary instance of an unknown reference object and then to predict its unknown response value.

Details of the relational tree induction, the EM based implementation of multi-instance stepwise regression and the prediction of unknown reference objects are reported in the next sub Sections.

4.1 Relational Tree Induction

The construction of the tree proceeds top-down. It starts with a root node t_0 with is associated with the entire set of training reference objects and recursively proceeds by choosing from either:

- growing the tree by performing a splitting test on the current node t and introducing the nodes t_L (left child of t) and t_R (right child of t) or
- stopping the tree's growth at the current node t and then associating an hyperplane at t.

At each node, the splitting test is chosen by minimizing the average standard deviation of response value [4]. Coherently with the semantics of a relational tree defined in [2], a variable that is introduced in the splitting test (i.e. this variable does not occur in the higher splitting tests of the tree) must not occur in the negation of the splitting test. This restriction is required to guarantee the *mutually exclusivity* when partitioning reference objects according to the test conditions which may involve several data relations.

A splitting test may either be a *foreign key* test or an *attribute* test. A foreign key test is a binary test to partition reference objects according to the existence of a non empty relationship between data relations involved in one or more foreign key constraints. This corresponds to perform natural joins between relations involved in the foreign key path associated to the unique tree path connecting the root to the node and relations introduced with the foreign key test. Due to the complexity of computing natural joins, MIRT imposes that a foreign key between two data relations can be introduced at most once in this unique path. The foreign key constraints are selected from the relational database under the restriction that the resulting foreign key path must satisfy the linkedness. This corresponds to consider only foreign key constraints, where either the foreign key or the primary key belong to one data relation already involved in a test along the corresponding tree path from the root to the node.

Example 1. An example of splitting test which employs a foreign key condition (from atom to molecule) to partition the entire set of molecules:
molecule(I, L, M)
| − −(yes)[molecule(I, L, **M**),atom(A,**M**,C)]
| | − − ...
| − −(no) [molecule(I,L,M), **not(molecule(I,L,M),atom(A,M,C))**] ...

An attribute test is a test involving a boolean condition on an attribute X. X is neither a primary key nor a foreign key and X belongs to one of the data relations added to the tree by means of a foreign key condition. if X is continuous, the binary test is in the form $X \leq \alpha$ with α one of the points found on range of X in partition at hand, while if X is a discrete variable, the binary test is in the form $X \in \{\alpha_1, \ldots, \alpha_s\}$, where $\{\alpha_1, \ldots, \alpha_s\} \subset S_X$ and S_X is the set of distinct values of X in the partition in hand. MIRT starts with an empty set $Left_X = \oslash$ and a full set $Right_X = S_X$. It moves one element from $Right_X$ to $Left_X$, such that the move results in a better split.

Example 2. An example of splitting test which employs an attribute condition (Charge) to partition the set of molecules which have at least one atom related according to the foreign key constraint from atom to molecule:
molecule(I, L, M)
| − −(yes)[molecule(I, L, M),atom(A,M,C)]
| | − − (yes)[molecule(I,L,M),atom(A,M,C), **C≤2.3**]
| | − − ...
| | − − (no)[molecule(I,L,M),atom(A,M,C),**not(**
   molecule(I,L,M),atom(A,M,C), C≤2.3)]
| | − − ...
| − −(no) [molecule(I,L,M), not(molecule(I,L, M),atom(A,M,C))] ...

In addition to foreign key test and attribute test, MIRT can perform a test which combines one or more foreign key conditions with one attribute condition.

Example 3. An example of a splitting test which employs simultaneously a foreign key condition (from atom to molecule) and an attribute condition (Charge) to partition the entire set of molecules.

```
molecule(I, L, M)
|− −(yes)[molecule(I,L,M),atom(A,M,C), C≤2.3]
|          |− − ...
|− −(no) [molecule(I,L,M),not(molecule(I,L,M),atom(A,M,C),C≤2.3)]
|          |− − ...
```

The tree construction is stopped when either the number of reference objects in a node is less than a minimum value or the coefficient of determination is greater than a threshold. The coefficient of determination estimates the strength of the relationship between the average response values on partition at hand and the average response values on the entire training set.

4.2 Multi-instance Stepwise Regression

A local hyperplane $y = \mathbf{bx}$ at a leaf node is learned to be close to f on the primary instances constructed for the training reference objects falling in the partition at hand. Primary instances are constructed by fixing one binding among the possible several ones of the (non-determinate) variables involved in the hyperplane and $\mathbf{b}$ is determined by minimizing the least square error (L) on the primary instances. The hyperplane construction starts from representing each training reference object i falling in the partition at hand as a bag B_i of m_i attribute-value vectors (instances). Each instance j of B_i is described by the real valued vector $\mathbf{x_{ij}}$ which includes values of the continuous explanatory variables X_i from the data relations in the foreign key conditions along the tree path from the root to the leaf under analysis. The multiple instances of B_i are only those instances which satisfy the conjunction of test conditions for the reference object i. An example of the construction of the bag of attribute-value vectors which describe a reference object falling in a leaf is reported in Example 4.

Example 4. Let us consider:

1. a database schema which includes the data relations
   Molecule(MolId, LogP and Muta)
   Atom(Atomid, MolId, Charge)
   Bond(BondId, AtomId1, AtomId2, Type)
2. an instance of this database which collects the tuples:
   molecule(m1, 12, 5.1). molecule(m2, 2, 10.1).
   atom(a1, m1, 2.3). atom(a2, m1, 2.5). atom(a4, m1, -0.5). atom(a5, m2, 5.1).
   bond(b1, a1, a2, 5). bond(b2, a1, a2, 2). bond(b3, a1, a3, 1).
   bond(b4, a5, a6, 2). bond(b5, a6, a5, 1).
3. a relational tree that partitions molecules according to atoms and bond is the following:

```
1. molecule(I, L, M)
2. |− −(yes)[molecule(I, L, M),atom(A,M,C) ]
3. |       |− − (yes)[molecule(I,L,M),atom(A,M,C), C≤2.3]
4. |            |−− (yes)[molecule(I,L,M),atom(A,M,C), C≤2.3, bond(B,A,A2,T]
5. |            |− − (no) ...
```

6. | | − − (no)...
7. | − −(no) ...

$m1$ satisfies the conjunction of binary conditions along the path from the node 1 to the node 4, that is, at least one atom with a charge less or equal to 2.3 belongs to $m1$ and this atom is involved in at least one bond. The intensional description provided by the tree at this node can be expressed as the query:

m1(L, C, T) :- molecule(m1,L,Y), atom(M,A,C), C≤2.3, bond(B,A,K,T)

According to the query reported above $m1$ is mapped into the bag attribute-value vectors $\langle 12, 2.3, 5\rangle$ $\langle 12, 2.3, 2\rangle$ $\langle 12, 2.3, 1\rangle$ and the entire bag is assigned to the response value 5.1.

After constructing the multiple attribute-value vectors which represent each reference object falling in the leaf, primary instances and regression coefficients are computed within an EM algorithm that aims at minimizing the least square error of the hyperplane on the training primary instances constructed at the leaf. The hyperplane is built stepwise by sequencing straight line regressions and removing the linear effect of the introduced variables each time a new explanatory variable (regression term) is added to the model. The selection of this *best* term is based on the strength of the resulting least square error on the chosen primary values for X_i. Basics of stepwise construction are provided in Example 5.

Example 5. Suppose we are interested in analyzing a response variable Y in a region R of a feature space described by two continuous explanatory variables X_1 and X_2. In the stepwise construction of a regression model, the initial regression model is approximated by regressing on X_1 for the whole region R, that is $\hat{Y} = \hat{a}_0 + \hat{b}_0 X_1$. As explained in [8], the correct procedure to follow in order to introduce the effect of another variable in the partially constructed regression model is to eliminate the effect of X_1. In practice, we have to compute the regression model for the whole region R, that is, $\hat{X}_2 = \hat{a}_{20} + \hat{b}_{21} X_1$ and to compute the residuals $X_2' = X_2 - \hat{X}_2$ and $Y' = Y - \hat{Y} = Y - (\hat{a}_0 + \hat{b}_0 X_1)$. Finally by regressing Y' on X_2' alone $Y' = \widehat{\beta_{0_3}} + \widehat{\beta_{1_3}} X_2'$ and substituting the equations of X_2' and Y' in the last equation we obtain:

$$Y - (\widehat{\beta_{0_1}} + \widehat{\beta_{1_1}} X_1) = \widehat{\beta_{0_3}} + \widehat{\beta_{1_3}}(X_2 - (\widehat{\beta_{0_2}} + \widehat{\beta_{1_2}} X_1)).$$

that is, $Y = (\widehat{\beta_{0_3}} + \widehat{\beta_{1_1}} - \widehat{\beta_{0_2}\beta_{1_3}}) + (\widehat{\beta_{1_1}} - \widehat{\beta_{1_2}}\widehat{\beta_{1_3}})X_1 + \widehat{\beta_{1_3}} X_2$.

For each bag, the primary value corresponding to the variable to be added to the hyperplane is chosen within the EM implementation described in Algorithm 1. Each time a new variable is added to the hyperplane, primary values of this variable are definitely assigned to the values (I) chosen in the E step. The algorithm starts with an initial random guess (IR) at the hypothesis which is iteratively refined. Each iteration consists in two main steps. In the E step, a binding of X_i is selected from each bag to obtain an hypothesis with least square error (L-error) with respect to the current best guess at the correct hypothesis. In the M step, the current guess of the hypothesis is refined by using linear regression to construct a new regression model from the instances provided at the previous step. The new hyperplane is constructed by determining the coefficient of the

Algorithm 1. EM based selection of regression coefficients and primary values

```
Input: (1) the vector $B$ of $n$ bags $b_i$, where the bag $b_i$ includes the values $x_{i1}, \ldots x_{im}$ for the continuous explanatory variable $X$, (2) the list $L$ of the explanatory variables already included in the hyperplane, (3) the vector $P'$ of the residuals of the primary instances constructed from the explanatory variables already included in the hyperplane, (4) a vector $Y'$ of the residuals of the response values.
Output: (1) the regression coefficient of a straight-line regression between Y' and residual of X', (2) the vector $I$ of $n$ primary values of $X$, one for each bag in $B$
function EM($in : B, L, P', Y'; out : b, I$);
globalL:=MAX;
for r=1, ...,R do
    assign randomly the primary instances to IR and determine residuals IR' by removing the effect of variables in $L$;
    determine the regression coefficient b over IR';
    bestL=MAX; currentL=0; found=true;
    while found do
        IC:=∅ ; currentL:=0;
        for each $b_i$ in B do
            Let $x_{ij}$ be the instance value in $b_i$ that minimizes $L(y'_i, x'_{ij}, b)\}$; /* $x'_{ij}$ is the residual of $x_{ij}$ */
            $IC = IC \cup x_{ij}$; currentL:=currentL+ $L(y'_i, x'_{ij}, b')$;
        end for
        if currentL≥bestL then
            found:=false;
        else
            bestL:=currentL; bR:=b; IR:=IC;
            perform linear regression over IC to obtain b
        end if
    end while
    if bestL<globalL then
        globalL:=bestL; b:=bR; I:=IR
    end if
end for
return ⟨b,I⟩
```

straight-line regression between the residual of Y and the residual of X_i. Both residuals are computed in a stepwise fashion by iteratively removing the effect of the already performed steps of regression. The coefficients of the (sequence of) straight-line regressions (e.g. $\hat{a}_{20}$ and $\hat{b}_{21}$ in Example 5) to determine the residual values involved in an EM step are computed over the primary instances which have already been constructed within the stepwise construction of the hyperplane. EM steps are repeated until the algorithm converges. Due to the fact that the result of any EM run may be influenced by the initial random hypothesis, it is run several times on the same data collection using random restarts. In Algorithm 1, R is the number of random restarts to be used.

After selecting the variable to be added to the hyperplane, the contribution of this term is evaluated according to the F-test and eventually dropped whenever it is not statistically significant. In this last case, the addition of any other candidate cannot be statistically significant, hence the hyperplane construction can be stopped. In this way, MIRT integrates a mechanism of variable subselection as a part of the hyperplane construction, thus solving possible problems of collinearity [8]. If no variable is added to the hyperplane, the prediction at the leaf is simply performed by means of the mean of the response values of the reference objects falling in the leaf. Primary instances constructed within the EM based stepwise construction of the hyperplane are stored at the leaves and are subsequently used to determine the primary instance when an unknown reference object has to be predicted.

An example of the stepwise construction of an hyperplane according to Algorithm 1 is provided in Example 6

Example 6. Let us consider the molecules m1, m2 and m3 described according to the atoms a1, a2, a3, a4, a5, a6 and a7. Each molecule consists of a bag of attribute-value vectors (*Logp*, *Charge*) and a response value (*Muta*).

InstanceId	MolID	Logp	Muta	Atomid	Charge
1	m1	2	7.5	a1	5.1
2	m1	2	7.5	a2	30
3	m1	2	7.5	a3	5
4	m2	5	12.5	a4	11
5	m2	5	12.5	a5	11.5
6	m3	3	9.8	a6	7.1
7	m3	3	9.8	a7	7

We use the stepwise procedure to estimate the regression coefficients of an hyperplane to predict *Muta* that includes *LogP* and *Charge*. For simplicity, we consider R=1 and no F-test is performed when adding a new variable to the hyperplane.

An initial hyperplane is approximated by choosing to regress *Muta* on either *LogP* or *Charge*. At this aim, we compute the regression model for *LogP* by randomly selecting primary values of *LogP* from each bag and iterating in order to minimize the least square error (see Algorithm 1). Since *LogP* assumes a single value on each bag, a single iteration is performed and it computes:

$$Muta = 4.52 + 1.62 LogP \text{ with } I = [2, 5, 3] \text{ and } globalL = 0.25 \tag{1}$$

Similarly, we compute coefficients of a regression model for the multi-instance value of *Charge*. Let us consider the case that a random choice returns [5.1,11,7] as primary values for *Charge*. We use these primary values to determine:

$$Muta = 3.61 + 0.82 Charge \tag{2}$$

Equation 2 minimizes the least square error on the charge values IC=[5,11.5, 7.1] (with currentL=0.196). By using this new set of primary values, we compute:

$$Muta = 3.65 + 0.81 Charge \tag{3}$$

that minimizes the least square error with the charge values IC=[5,11.5, 7.1] (with currentL=0.195). This new set of primary values for *Charge* will lead to stop EM iteration and return:

$$Muta = 3.65 + 0.81Charge \text{ with } I = [5, 11.5, 7.1] \text{ and } globalL = 0.19. \qquad (4)$$

The initial hyperplane is then approximated by regressing *Muta* on *Charge* according to Equation 4. The residual attribute $Muta'$ is computed as follows:

$$Muta' = Muta - (3.65 + 0.81Charge) \; Muta' = [-0.23, -0.12, 0.35] \qquad (5)$$

Finally, we introduce the effect of *LogP*. Let us consider the case the random choice return [2,5,3] as primary values for *LogP*. We firstly compute the residuals $LogP'$ by using the primary values of *Charge* in Equation 4, that is:

$$LogP' = LogP - (-0.52 + 0.5Charge) \qquad (6)$$

and then we compute:

$$Muta' = -10.99LogP' \qquad (7)$$

which is proved to be the best one according to Algorithm 1. In this way we complete the construction of the following hyperplane:

$$Muta = (3.65 + 0.81Charge) - 10.99(LogP - (-0.52 + 0.5Charge)) \qquad (8)$$

by constructing as primary instances of m1, m2 and m3, the attribute-value vectors $\langle m1, 2, 5\rangle$, $\langle m2, 5, 11.5\rangle$ and $\langle m3, 3, 7.1\rangle$, respectively.

4.3 Predicting Unknown Reference Objects

Let τ be the relational model tree induced from relational data stored in D. Let H be the schema of D and ro be a test reference object that is stored in a new instance (T) of the same database schema H. T contains the task-relevant objects which interact with ro according to the foreign key constraints defined in H. The response value of ro is unknown in T.

Starting from the root node of τ, ro is recursively passed down to the left (or right) child according to the fact that the splitting test on left (or right) edge is satisfied. When a leaf node is reached, MIRT constructs instances which describe ro in T according to the task-relevant objects which are related to ro in the partition at hand. The structure (attribute vector) of these instances includes only the explanatory variables $(X_1, X_2, \ldots, X_d)$ which are involved in the hyperplane tagging the leaf. The primary instance of ro is then constructed by fixing, for each variable, one binding over the bag and then by minimizing the distance from the training primary instances stored at the leaf.
Formally speaking, given:

1. the leaf t such that ro reaches t;
2. the hyperplane $y = g(\mathbf{X})$ which tags t such that the attribute vector $\mathbf{X}$ is spanned by d continuous variables $X_1, \ldots, X_d$;

3. the set P of the training primary instances defined on $\mathbf{X}$ and stored in t;
4. the bag B_{ro} of the database instances defined on $\mathbf{X}$ and constructed over T to represent the reference object ro falling in t;
5. the set ψ_i (χ_i), that is, the range of X_i over P (R).

MIRT constructs the primary instance o of ro by assigning each X_i with one value over χ_i in order to minimize the distance from training primary instances stored in P, that is:

$$o = \min_{o \in \chi_1 \times \chi_2 \ldots \times \chi_d} \min_{t \in T} distance(p,t). \tag{9}$$

The distance between p and t is computed on the basis of the Euclidean distance measure, that is:

$$distance(p,t) = \sqrt{\sum_{i=1,\ldots,d} (p[X_i]) - (t[X_i])^2}. \tag{10}$$

Adopting the classical Euclidean distance, as in Equation 10, brings the problem of combining variables whose range may differ in several orders of magnitude. To overcome this problem, each value of X_i is scaled within the range $[0,1]$: the lowest (highest) value is assigned the real value 0 (1). The scaled values of $p[X_i]$ and $t[X_i]$ are obtained as follows:

$$p[X_i]_{scaled} = \frac{p[X_i] - min_j}{max_i - min_i} \quad \text{and} \quad t[X_i]_{scaled} = \frac{t[X_i] - min_j}{max_i - min_i}, \tag{11}$$

where $min_i = \min_{x_i \in \chi_i \cup \psi_i} x_i$ and $max_i = \max_{x_i \in \chi_i \cup \psi_i} x_i$. Once the primary instance o is constructed, it is used to predict the unknown response value by assigning the explanatory variables in the hyperplane at t to the corresponding values in o.

5 Experiments

MIRT is implemented in a Multi-Relational Data Mining system tightly coupled with a relational database (Oracle 10g) and it is empirically evaluated on biological and geographical relational databases. Biological databases represent a benchmark application domain in multi-instance learning.

Experimental Setting. Each dataset is analyzed by means of a 10-fold cross-validation. Ten databases are created so that MIRT can be trained on nine databases and tested on the hold-out database. The system performance is evaluated on the basis of the average mean square error (MSE), that is:

$$MSE = \frac{1}{k} \sum_{i=1}^{10} \sqrt{\frac{1}{\#S_{D_i}} \sum_{j=1}^{\#S_{D_i}} (y_j - \widehat{y}_j(D/D_i))^2} \tag{12}$$

where $D = \{D_1, \ldots, D_k\}$ is a cross-validation partition, D_i is a set of indices of testing databases, k is the number of folds (i.e., 10), $\#S_{D_i}$ is the number of reference objects stored in D_i and $\widehat{y}_j(D/D_i)$ is the value predicted for the j-th testing reference object by the model tree induced on D/D_i.

The thresholds for the stopping criteria are fixed as follows: the minimum number of reference objects falling in an internal node must be greater than the square root of the number of reference objects in the entire database, and the coefficient of determination in an internal node must be below 0.8. R is set to 5, while the maximum number of foreign key constraints to be added with a foreign key test is set to 2.

MIRT is compared with Mr-SMOTI [1] which induces a relational model tree that interleaves splitting tests and regression steps. At regression steps, Mr-SMOTI estimates the regression coefficients of straight-line regressions by assuming multiple-bindings of a non-determinate variable as single instances of least square regression. On one database, i.e. Mutagenesis, we compare MIRT with RE-MAUVE and TILDE-RT. RE-MAUVE is a relational model tree learner which resorts to aggregates to deal with non-determinate variables. TILDE-RT is a relational regression tree learner which associates a constant to each leaf. Finally, we compare MIRT with the propositional model tree learners SMOTI and M5'. In this last case, multi-relational data are transformed into a single table format. Two different transformations are considered. The former (P1) creates a single table by computing join operations for all possible foreign key paths rooted in the target table. This transformation may create multiple tuples for the same reference object. The latter transformation (P2) differs from the previous one because it does not generate multiple tuples for the same reference object. It is obtained by computing aggregates (i.e. the average for continuous values and the mode for discrete values) of-non-determinate variables. In the case of P1 we compute the MSE with respect to the average of the multiple response values output for the same test reference object as final prediction (MSE-G). We also compute the MSE by considering the response value as prediction of each single instance (MSE-S). Differently, in the case of P2, a single response is directly output for each test reference object. For the pairwise comparison of systems, the non-parametric Wilcoxon two-sample paired signed rank test [17] is used.

Data Description. MIRT is tested on four real databases, that is, Mutagenesis, Biodegradability, North West England (NWE) and Munich. Mutagenesis [21] and Biodegradability [10] are molecular databases used as a benchmark for several ILP systems. Mutagenesis is evaluated in three different settings. B0 consists of those data obtained with the molecular modeling package QUANTA. For each compound it obtains the atoms, bonds, bond types, atom types, and partial charges on atoms. B1 consists of definitions in B0 plus indicators ind1, and inda in molecule table. B2 consists in B1 plus variables (attributes) logp, and lumo. Biodegradability is evaluated in four settings. B0 consists of those data derived with SMILES without any global feature on molecule. B1 adds the numerical attributes mWeight and logP. B2 extends B0 by adding the indicator on molecular activity, while B3 includes all global features describing the molecules.

Details are provided in [1]. NWE is a collection of geo-referenced census data provided by the United Kingdom (UK) 1998 census. NWE census data includes values of mortality rate (response variable) and deprivation indexes geo-referenced at the level of 212 wards. Data also include 1045 rails, 2763 roads, 374 urban areas and 1040 waters crossing wards for a total of 5434 tuples. NWE dataset is provided in the European project SPIN!(http://www.ais.fraunhofer.de/KD/SPIN/project.html). Munich (http://www.di.uniba. it/%7Ececi/micFiles/munich db. tar.gz) describes rent-price (response variable) of 2179 flats geo-referenced within the Munich subquarters. The Munich metropolitan area is divided into 3 areal zones, each decomposed into 64 districts, for a total of 446 quarters for a total of 6808 tuples. This data was collected in 1998 to develop the Munich rental guide in 1999.

Results. The average MSE of the multi-relational systems is reported in Table 1. For Mutagensis (B1 and B2), we report MSE of RE-MAUVE and TILDE-RT taken from [22][1][2]. The comparison between MIRT and Mr-SMOTI (as well as Re-MAUVE and TILDE-RT for Mutagenesis) confirms our intuition that the accuracy of a relational model tree is generally improved when regression coefficients of non-determinate variables are estimated according to principles of multi-instance learning. The only database where Mr-SMOTI outperforms MIRT is Mutagenesis B2, in which mutagenecity of molecule strongly depends on the numeric properties of molecules (i.e., lumo and logP) which have single values for each molecule. This result is a confirmation that the stepwise construction of a model tree by interleaving split nodes and regression nodes outperforms the classical construction of a model tree by firstly partitioning data set and then locally deriving the hyperplanes to predict reference objects at each node [1]. Anyway, the advantages of a tree structure with split and regression nodes may be decreased by the presence of outliers values over non-determinate variables. This consideration suggests a future direction of the research described in this work, that is, employing the principles of multi-instance learning in the stepwise induction of relational model trees with split and regression nodes. The results on Mutagenesis show that MIRT also outperforms RE-MAUVE and TILDE-RT.

The results of the comparison between MIRT, SMOTI and M5' are reported in Table 2. The comparison is generally in favor of MIRT. The only statistically significant tests ($p \leq 0.05$) where a propositional learner (SMOTI -P2) outperforms MIRT concern Biodegradability (B2-B3). In general, the comparison of accuracy confirms not only the advantages of the structural approach over the

[1] The MSE of Mr-SMOTI on Mutagenesis significantly differs from the values reported in [22]. Differences may depend from a different tuning of parameters. In this work, we run Mr-SMOTI by allowing the possibility of learning foreign key tests introducing two foreign keys simultaneously (the default is 1), learning a foreign key test and an attribute test simultaneously in the same test (by default this possibility is disabled), filtering splitting tests which select less than 5 molecules on the left and right side of the tree.

[2] The MSE values reported in [22] are without the square radix.

Table 1. MIRT vs Mr-SMOTI, Re-Mauve and TILDE-RT: MSE of the model trees induced on the 10-fold CV of databases. Best MSE are in italics. "-" ("+") means that Mr-SMOTI performs worse (better) than MIRT in a Wilcoxon Test. "–" ("++") denotes the statistically significant values (p≤0.05).

DB		MIRT	Mr-SMOTI	Re-MAUVE no agg	Re-MAUVE agg	TILDE-RT	TILDE-RT agg
Mutgenesis	B0	*1.4*	1.67 –				
	B1	*1.11*	1.28 -	1.40		1.4	
	B2	1.12	*0.95*+	1.20	1.20	1.24	1.36
Biodegradability	B0	*1.32*	1.37 -				
	B1	*1.206*	1.25 -				
	B2	*0.136*	0.49 –				
	B3	*0.142*	0.41 –				
NWE		*0.0024*	0.0026 =				
Munich		*4.66*	4.78 –				

Table 2. MIRT vs SMOTI and M5' (P1 and P2): MSE of the model trees induced on the 10-fold CV of databases. For SMOTI (P1) not all values are available, since the system returns error of memory. Best MSE are in italics. "-" ("+") means that SMOTI/M5' performs worse (better) than MIRT in a Wilcoxon Test. "–" ("++") denotes the statistically significant values (p≤0.05).

DB		MIRT	SMOTI (P1)		M5(P1)		SMOTI (P2)	M5(P2)
		MSE	MSE-G	MSE-S	MSE-G	MSE-S	MSE	MSE
Mutgenesis	B0	*1.4*	2.44 –	3.92 –	1.68 -	2.48 –	2.57 -	1.55 -
	B1	*1.11*	1.62 –	1.65 –	1.16 -	2.10 –	1.38 -	1.13 =
	B2	1.12	2.19 -	2.15 –	1.05 -	1.05 -	1.009 +	*1.007* +
Biodegradability	B0	*1.32*			1.44 –	1.35 -	1.63 –	1.41 -
	B1	*1.20*			1.68 –	1.71 –	2.13 –	1.28 -
	B2	0.13			0.12 +	0.12 +	*0.06* ++	0.18–
	B3	0.14			0.14 =	0.15 -	*0.05* ++	0.18 –
NWE		0.0024	0.0028 –	0.0026 =	0.0028 -	0.0026 -	0.003 –	*0.0023* +
Munich		4.66	5.90 –	6.03 –	5.25 –	5.27 –	5.49 –	*4.62* +

propositional one when mining model trees from multi-relational data, but also the validity of multi-instance approaches in relational regression.

6 Conclusions

MIRT is a novel multi-relational data mining system which induces a relational model tree to predict the response value of a reference object (target object) by taking into account non-determinate task-relevant objects. This work points out the hypothesis that it is almost never the case that all multiple-bindings of a non-determinate variable contributed to the observed response value. Under this hypothesis, a piece-wise hyperplane is constructed by locally sequencing straight-line regressions in a stepwise fashion and estimating coefficients of such regressions on the basis of only the best bindings of the regression variables. Bindings are chosen within an EM implementation that minimizes least square error on primary values. Explanatory variables involved into an hyperplane are a subset of the explanatory variables from the data relations which appear in the splitting test along the tree path from the root to a leaf. Problems of collinearity are naturally solved by stopping the hyperplane construction when the addition of a term is not statistically significant. The comparison between MIRT and the relational model tree system Mr-SMOTI as well as the propositional model tree systems, SMOTI and M5', confirm that identifying the primary instances outperforms existing propositional and structural systems.As a future study, we plan to apply principles of multi-instance learning to construct a relational model tree with split and regression nodes.

Acknowledgments

This work is partial fulfillment of the research objective of ATENEO-2008 project "Knowledge Discovery in Relational Domains". The authors thank anonymous reviewers for their useful suggestions.

References

1. Appice, A.: Learning Relational model Trees. PhD thesis, Department of Computer Science, University of Bari, Bari, Italy (2005)
2. Blockeel, H.: Top-down induction of first order logical decision trees. PhD thesis, Department of Computer Science, Katholieke Universiteit, Leuven, Belgium (1998)
3. Blockeel, H., Page, D., Srinivasan, A.: Multi-instance tree learning. In: Raedt, L.D., Wrobel, S. (eds.) International Conference on Machine Learning, vol. 119, pp. 57–64. ACM, New York (2005)
4. Breiman, L., Friedman, J., Olshen, R., Stone, J.: Classification and regression tree. Wadsworth & Brooks (1984)
5. Davis, J., Costa, V.S., Ray, S., Page, D.: An integrated approach to feature invention and model construction for drug activity prediction. In: Ghahramani, Z. (ed.) International Conference on Machine Learning, vol. 227, pp. 217–224. ACM, New York (2007)
6. Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artif. Intell. 89(1-2), 31–71 (1997)
7. Dooly, D.R., Zhang, Q., Goldman, S.A., Amar, R.A.: Multiple-instance learning of real-valued data. Journal of Machine Learning Research 3, 651–678 (2002)
8. Draper, N.R., Smith, H.: Applied regression analysis. John Wiley & Sons, Chichester (1982)
9. Driessens, K., Džeroski, S.: Combining model-based and instance-based learning for first order regression. In: Raedt, L.D., Wrobel, S. (eds.) International Conference on Machine Learning, ICML 2005, pp. 193–205. ACM, New York (2005)

10. Džeroski, S., Blockeel, H., Kramer, S., Kompare, B., Pfahringer, B., Van Laer, W.: Experiments in predicting biodegradability. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 80–91. Springer, Heidelberg (1999)
11. Džeroski, S., Lavrač, N.: Relational Data Mining. Springer, Heidelberg (2001)
12. Džeroski, S., Todoroski, L., Urbancic, T.: Handling real numbers in inductive logic programming: A step towards better behavioural clones. In: Lavrač, N., Wrobel, S. (eds.) ECML 1995. LNCS, vol. 912, pp. 283–286. Springer, Heidelberg (1995)
13. Hardle, W.: Applied nonparametric Regression. Cambridge University Press, Cambridge (1990)
14. Kramer, S.: Relational Learning vs. Propositionalization: Investigations in Inductive Logic Programming and Propositional Machine Learning. PhD thesis, Vienna University of Technology, Vienna, Austria (1999)
15. Kramer, S., Pfahringer, B., Helma, C.: Stochastic propositionalization of nondeterminate background knowledge. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 80–94. Springer, Heidelberg (1998)
16. Maron, O., Lozano-Perez, T.: A framework for multi-instance learning. Advanced in Neural Information Processing Systems
17. Orkin, M., Drogin, R.: Vital Statistics. McGraw-Hill, New York (1990)
18. Ray, S., Page, D.: Multiple instance regression. In: Brodley, C.E., Danyluk, A.P. (eds.) International Conference on Machine Learning, pp. 425–432. Morgan Kaufmann, San Francisco (2001)
19. Saith, R., Sergent, I.: Embryo selection for transfer in human IVF. Assist Reprod. Rev. 5, 145–154 (1995)
20. Srinivasan, A., Camacho, R.: Numerical reasoning with an ilp system capable of lazy evaluation and customised search. J. Log. Program 40(2-3), 185–213 (1999)
21. Srinivasan, A., Muggleton, S., King, R.D., Sternberg, M.J.E.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: Wrobel, S. (ed.) International Workshop on Inductive Logic Programming, pp. 217–232 (1994)
22. Vens, C., Ramon, J., Blockeel, H.: Remauve: A relational model tree learner. In: Muggleton, S., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 424–438. Springer, Heidelberg (2007)
23. Zhang, Q., Goldman, S.: EM-DD: An improved multiple-instance learning technique. In: Neural Information Processing Systems (2001)

Challenges in Relational Learning for Real-Time Systems Applications

Mark Bartlett[1,2], Iain Bate[2], and Dimitar Kazakov[1]

[1] Artificial Intelligence Group, Department of Computer Science,
University of York, Heslington, York, UK
[2] Real Time Systems Group, Department of Computer Science,
University of York, Heslington, York, UK
{mark.bartlett,iain.bate,dimitar.kazakov}@cs.york.ac.uk

Abstract. The problem of determining the Worse Case Execution Time (WCET) of a piece of code is a fundamental one in the Real Time Systems community. Existing methods either try to gain this information by analysis of the program code or by running extensive timing analyses. This paper presents a new approach to the problem based on using Machine Learning in the form of ILP to infer program properties based on sample executions of the code. Additionally, significant improvements in the range of functions learnable and the time taken for learning can be made by the application of more advanced ILP techniques.

Keywords: Worst Case Execution Time (WCET), Inductive Logic Programming (ILP), Lazy Learning, Symmetry, Efficiency.

1 Introduction

In the area of Real-Time Systems (RTS), the temporal behaviour of systems is of critical importance. In particular, a substantial amount of research effort goes into the issue of guaranteeing that code will be executed within a given time frame. While much scheduling theory exists on this matter, it is commonly assumed that the timing behaviour of the individual software components is known. One of the temporal properties of a process that schedulers require is that of the Worst Case Execution Time (WCET). As the name implies, this is the longest time that the process may require to run given the worst case input possible.

A fundamental problem exists however in determining the WCET of a program. The WCET of a program is in general undecidable, due to the well-known *halting problem* [1]. Therefore in the general case, any non-exhaustive attempt to determine this quantity will only ever be able to return an approximation. We propose that the use of Machine Learning may be a viable alternative to existing methods in the field, allowing for the accurate approximation of WCET in a competitive time.

One particular aspect of determining WCET that is examined here is the issue of deciding on the number of times that a loop is executed. Determining this

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 42–58, 2008.

quantity with high accuracy has the ability to massively increase the accuracy for the WCET approximation as a whole. Again, this problem is in general undecidable, though there are well-defined classes subclasses of loop for which the problem is mathematically decidable. One of these classes, Presburger Loops, will be studied in this paper. For this class, our technique is able to produce an exact solution. Program flow analysis, and in particularly loop bound estimation, has been identified as an important source of overestimation, up to 30%, leading to a significant waste in resources.

There are several aspects of the WCET problem that makes Machine Learning, and ILP in particular, a good candidate for providing a solution. The data to learn from possesses many characteristics that are highly desirable in this realm: it is noiseless, discrete, deterministic and available in virtually limitless quantities [2]. These properties mean that many problems that occur in more complex domains should be avoided completely here.

Nevertheless, there are real challenges in using ILP for this task. These stem primarily from two sources. Firstly, basic ILP isn't particularly well-suited to the learning of numeric data and equation discovery. Secondly, the range of functions to be considered as hypotheses is vast. A significant part of this paper is devoted to showing how these limitations can be overcome in order to massively increase the speed with which hypotheses can be found and to allow the discovery of equations not possible with a naively coded learner.

The rest of this paper proceeds as follows. Firstly, the background to the WCET problem is presented in section 2 along with current approaches to its solution. The suitability of ILP and other related methods are then considered in section 3. Section 4 deals with the construction of a new ILP formulation to solve the a particular aspect of the problem. Firstly, a simple, previously published [2] implementation capable of learning the number of loop executions is presented. The problems that exist with this implementation are then highlighted. From this, a novel implmentation is then presented, through a series of refinements using more advanced ILP techniques, resulting in a learner capable of acquiring a larger class of equations and in a faster time. Results showing the improved ability of the more advanced learner to solve the problem are then given in section 5. The massive improvement in the behaviour of the more advanced learner is also shown. A related result is shown in section 6, in which machine learning was successfully applied to another aspect of WCET analysis, learning a branch predictor. Finally section 7 concludes and considers how ILP can be applied to other aspects of the WCET.

2 Worst-Case Execution Time Analysis

There are three current approaches to estimating WCET. Static analysis, measurement-based analysis and hybrid analysis. Static analysis examines the code and execution environment mathematically to build a model to reason about the program behaviour. In contrast measurement-based analysis comes to an estimate through running execution tests on the code and target platform directly. Hybrid analysis combines the two previous approaches, building models based on the code and combining these with timing measurements found from actual executions. Each of these approaches has different strengths and weaknesses.

2.1 Static Analysis

Static analysis is based on automated reasoning about the execution of a program based on its code and the execution hardware. As the determination of WCET is in general undecidable, static analysis will not in general yield the correct WCET of a piece of code. Instead, the incompleteness of the reasoning process ensures that the estimated WCET is always at least as long as the actual WCET (*safe*), with the overestimate being termed the *pessimism*. This safety is important in hard real-time systems where deadlines must be guaranteed to be met and overruns could literally cost lives. However, pessimism is undesirable as it can lead to underutilised hardware resources or unnecessary investment in faster equipment.

Typically static analysis is done in three phases; flow analysis, low level analysis and calculation.In the first phase, the analyser reasons about the possible paths that may be taken through the code. It does this by building a *control flow graph* (CFG) showing how control passes between basic blocks of code, and then reasoning about this flow. There are several common techniques that may be utilised such as abstract interpretation [3] and symbolic execution [4], the former simulating the effect of code on a value range rather than individual input values and the latter building up constraints that replicate the logical flow of the program. In either case, the CFG will be annotated with derived flow facts, such as the maximum number of loop executions and infeasible paths. Due to the undecidability of the problem, these facts are incomplete.

Following this, low level analysis determines the time that would be taken to execute particular paths on the target platform. This analysis must account for advanced hardware features such as processor pipelines and branch predictors. As modern processors have become more advanced, the potential is for the degree of pessimism to rise [5]. For modern processors, technical details are often difficult to obtain due to commercial confidentially. Therefore, current analysis is generally restricted to parts of the processor for which the low level behaviour is well known and for which analysis is feasible. It appears that this problem will only get worse as processors become even more complex, such as Multi-Processor Systems on a Chip (MPSoC).

Finally, the calculation phase combines these two earlier phases to reveal the WCET estimate of the code. There are three main methods used for this; structure-based [6], path-based [7] and implicit path enumeration (IPET) [8]. Each integrates the flow and timing information, but using different techniques.

2.2 Measurement-Based Analysis

The other existing approach to estimating WCET is measurement based analysis. Here, the code is analysed through repeatedly running it on different inputs in an attempt to manually find and exercise the worse case path.

There are three methods that may be used to do this. Firstly, it is possible to manual generate test data based on human reasoning about the inputs most likely to be time-consuming to process. This can be effective if the tester has good knowledge of the code and problem domain, but this is often not the case. Secondly, a coverage metric can be employed, which defines a set of necessary execution conditions that must be performed by the test suite [9]. For example, in order to achieve branch coverage, it is necessary that both the true and false branch of each conditional are exercised by tests. The coverage method of testing enables systematic examination of the software, but the source code must be available; this is not always possible when library functions are called. Finally, testing can be automated through the use of genetic algorithms [10]. The input used for a test is the chromosome and familiar cross-over and mutation techniques are used to explore the space of all inputs. Fitness can either be scored based directly on execution time, or based on factors within the code such as the number of times a loop executes or whether particular conditional branches were taken.

For any non-exhaustive test data set, however generated, there always remains the possibility that the WCET has not been observed. This means that measurement-based approaches cannot be considered safe. In practice, this is overcome by allocating a margin of error to the worst observed time (for example 10%), though this still does not guarantee the safety of the approach. For any hard real-time system in which deadlines must be met, measurement-based analysis is unsuitable. However, there are many soft real-time applications where an occasional missed deadline can be tolerated. For example, in telecommunication applications, failure to decode a single frame of video in time to display may be acceptable providing such failures are infrequent.

2.3 Hybrid Analysis

Hybrid analysis works like static analysis, but circumvents the problem of unhandleably complex hardware by using actual program executions for the low-level phase. The flow analysis and calculation phases are typically performed in a similar manner to that used in static analysis, with the low-level data coming directly from real executions of the basic blocks.

This approach removes the excessive pessimism associated with low-level analysis on highly complex or poorly understood hardware, but also results in safety no longer being guaranteed. While the testing guarantees the accuracy of timing behaviour for individual blocks, safety may be lost through interactions over longer ranges, such as through pipelines or alterations in the cache contents.

Nevertheless, hybrid analysis is becoming used in domains where occasional deadline misses can be tolerated. The combination of rigorous static code analysis and measurement-based hardware analysis combines some of the strengths of both approaches.

3 Suitability of ILP and Related Methods

3.1 Suitability of ILP

Given the problems in the existing methods for WCET, there is potential for a alternative approach. A technique based on Machine Learning may fit the criteria for this approach. Using Machine Learning, execution traces of programs can be used to infer a mathematical model which is able to reproduce the observed data. This can then be used to provide information to be incorporated into a static analysis. For example, in the case of determining loop bounds, which is examined later, traces of the number of loop executions observed in practice can be used to construct a parametric model which enables the number of loop bounds to be predicted for any given input. This can then be incorporated into a static analysis at the flow analysis stage as a flow fact. Obtaining data through observation rather than through analysis of the code means the results may no longer be safe; this problem may be addressable through a suitable choice of which, and how many, execution traces to learn from.

The approach is somewhat akin to hybrid analysis, though rather than incorporate observed data directly into the analysis, it is used to build a model which can then be included. Additionally, whereas hybrid analysis uses only observed behaviour for the low-level stage, our approach allows models based on observed data to be included at the flow analysis stage as well. Finally, this method allows existing static analysis to be retained and complements it with additional information; in hybrid analysis, reasoning about the low-level behaviour is discarded and replaced in its entirety by execution data.

There are several aspects of the problem which make ILP an ideal Machine Learning technique for the task at hand. The data from which the model will be learned is noiseless; this removes one of the problems that frequently makes ILP difficult to apply to a domain. The variables likely to be encountered are discrete, and often either intervals or categories. This immediately suggests the use of a first-order logical representation for the data and theories such as is seen in ILP. The examples to be learned from will only be positive; one will be unlikely to observe, for example, how many times a loop is *not* executed. Off the shelf ILP tools such as Progol [11] and Aleph [12] have the capability to learn from positive only data built in.

3.2 Dynamic Invariant Detection

Learning the number of loop executions as a function of program variables is highly analogous to the process of dynamic invariant detection. Invariants are relationships between program variables that must always hold true at a certain point in a program. Detecting invariants statically is closely related to the static analysis techniques already described: invariants are found by reasoning mathematically about the program. In contrast, the dynamic technique infers invariants based on observing which properties are always true over a set of program traces.

The most widely used dynamic invariant detector is Daikon [13], which works in four stages. Firstly, the program to be examined is instrumented to record all program variables at each procedure entry and exit point. The instrumented program is then run on a user specified suite of test examples. Next, the resulting traces are processed to establish invariants for particular instrumentation points, and finally these candidate invariants are filtered to remove implied invariants and those likely to be due to chance observations.

Daikon detects a wide range of relationships that can hold between variables (including unary, binary and trinary relationships) and for many different datatypes. However, Daikon in an unmodified form cannot be used to learn program flow information. Daikon learns constraints that apply at procedure entry and exit; for learning of program flow, knowledge is required of how many times the body of a loop is entered and which branch is taken at conditional statements.

While conceivably these problems could be overcome by adding additional variables to the program, such as counters to loops[1], the learning bias is hard-coded into Daikon and difficult to modify. For ILP systems, this can be easily modified through altering the background knowledge. Furthermore, it is unclear how well Daikon can handle complex formulae with multiple variables and several constants which must be abducted.

3.3 LAGRAMGE

Lagramge [14] is a non-linear numerical regression (aka equation discovery) tool closely related to the ILP family, in which the user provides a grammar to describe the range of possible equations that should be considered in the search for the best fit for the data with respect to a given optimality criterion, typically the least squares. Here the grammar provides a description of the way how the independent variables could be linked through the use of certain operators to build an equation that best models the data in a way similar to the use of background predicates in mainstream ILP. The tool offers a choice between ordinary and differential equations, and later versions allow for the simultaneous learning of several equations. There are two aspects that make Lagramge unsuitable for the task at hand: the variable range cannot be restricted to integers, and, even more importantly, the algorithm focuses on minimising a function of the error on the average, whereas here one is interested in outliers and extreme behaviour.

[1] Though doing so would modify the code which may affect the compiled code generated, especially in the presence of compiler optimisations.

4 Loop Bound Learning

4.1 Problem Domain

In order to understand why achieving tight loops bounds for WCET analysis is important, it is worth considering the example of the typical nested loop used in many sort routines.

```
for $i = 0$ to $n$
    for $j = 0$ to $n - i - 1$
        if $a[j] < a[j+1]$ then
            ⋮
        end
    next
next
```

For many existing WCET tools, the maximum number of executions of the outer loop will be found to be n, and the maximum number of executions of the inner loop will be $n-1$. Traditional WCET techniques fail to determine the existence of a relationship between the number of loop executions and a counter, simply assuming that the worst possible value may occur in all interations. This results in an estimate of $n^2 - n$ executions of the loop body. In reality, the inner loop only executes $n - 1$ times when i is 0 and fewer in all other circumstances.

The actual total number of executions of the inner loop body will be $\frac{n^2-n}{2}$, exactly half that obtained by a basic analysis of the loops. Consequentially, we would expect the estimated execution time to be immensely pessimistic and twice the actual execution time. This pessimism arises in the cases where the number of executions of the inner loop is variable and depends on the value of the outer loop counter. A relational formula linking the loop executions to variables in the program can accurately record this situation.

No existing WCET technique is able to automatically detect such dependencies in program flow. As will be shown in later sections, through the use of machine learning for equation discovery, it is possible to establish relationships such as these.

Loop bounds can be data dependent and in the general case can be expressed with arbitrary complexity. These reasons are key to why the problem is classed as undecidable in the general sense [15]. However it is possible to describe restricted classes of loops for which the loop bound is a mathematically decidable problem. Presburger loops are one such class.

Presburger expressions [16] can be written in the form

$$k + \sum_p a_p V_p \qquad k, a_p \in \mathbb{R}$$

where V_p are variables. The number of executions of a loop can be decided if the loop conditionals are Presburger expressions in which the variables are either

```
% tp(A,B) where B = A * (A - 1) / 2
  tp(1,0).   tp(2,1).   tp(3,3).   tp(4,6).
  tp(5,10).  tp(6,15).  tp(7,21).  tp(8,28).
  ...
```

Fig. 1. Sort routine data set

loop counters from an outer containing loop or program variables which do not have their value changed.

```
for $i = (\alpha_0 + \sum_p \alpha_p V_p)$ to $(\alpha'_0 + \sum_p \alpha'_p V_p)$
    for $j = (\beta_0 + \beta_i i + \sum_p \beta_p V_p)$ to $(\beta'_0 + \beta'_i i + \sum_p \beta'_p V_p)$
        for $k = (\gamma_0 + \gamma_i i + \gamma_j j + \sum_p \gamma_p V_p)$ to $(\gamma'_0 + \gamma'_i i + \gamma'_j j + \sum_p \gamma'_p V_p)$
            ⋮
        next
    next
next
```

In the remainder of this paper, we shall restrict the class of loops considered to Presburger loops. There are several reasons for this. Firstly, many nested loops in actual program code correspond to this class, making the results relevant for real world situations. Secondly, as the result is decidable for this class, it is possible to generate data and check the accuracy of learned results without any complications. Finally, for a restricted class such as this, it is possible to make improvements to the efficiency of the learning algorithm based on knowlegde of the target concepts.

4.2 A Naive Loop Bound Learner

The potential of using ILP to learn a WCET bound was first shown by Kazakov and Bate [2] on the case of learning nested loop bounds. It is necessary to give an overview of this work here, as subsequent sections go on to discuss the problems with this implementation and develop an improved learner for the same task.

The use of ILP can be demonstrated with a sort routine which illustrates the case of nested loops where the inner loop bounds are functionally dependent on the outer loop counter. The known equation for this routine was then used to generate pairs of numbers representing the upper bound n and the corresponding number of times of running the inner loop body (see Fig. 1). The variable range was set to $n \in \{1, \ldots, 30\}$.

Two background predicates, `sum/3` and `product/3`, were used which calculated the sum and product of two arguments respectively. Using this formulation and data set, Progol4.4 finds a one-rule model.

```
tp(A,B) :- product(C,A,A),
           sum(D,B,A),
           sum(C,B,D).
```

This translates to a system of three equations:

$$C = A \times A$$
$$D = A + B$$
$$C = B + D$$

Note that the division operator is not part of the background knowledge, nor is Progol allowed to use constants in its hypotheses. Nevertheless, the result is correct, albeit expressed in a somewhat unusual way. Indeed, the above equations can be reduced to:

$$B = \frac{A \times (A - 1)}{2}$$

This is, of course, the correct formula.

Using a similar formulation, it was also shown that simple loops and nested invariant loops can be learned [2]. These early experiments are mostly valuable as a proof of concept, but they already show the potential for empirically deriving accurate upper bound estimates with acceptable amounts of processing. Previously, automatic processing in the WCET domain had not been able to acquire parametric formulae for loop nesting where the inner loop counter depends on the outer.

4.3 Limitations of the Naive Loop Bound Learner

While the learner based on the two background predicates `sum/3` and `product/3` was able to automatically acquire a range of simple expressions for loop bounds, it is unable to acquire the full range of Presburger loops.

In fact, it is not necessary to even look at nested loops to find the first example for which the naive learner cannot find the correct formula. A loop as simple as

```
for $i = 0$ to $1000V_1 + 1000$
    ⋮
next
```

is virtually unlearnable by the naive implementation. In order to find this formula, the learner would need to abduct 2 constants from the data. As there is no a priori reason to suppose that 1000 is a particularly interesting number, the learner must try all possible constant combinations in order to locate this particular example. Either the presented learner would fail to learn this formula at all (due to resource limitations), or would take an extraordinary time to find it. Therefore it is necessary to implement a more effective learner capable of learning the behaviour of Presburger loops in the general case.

4.4 An Improved Loop Bound Learner

While the naive approach is capable of learning a range of functions, including those for variously nested loops, it suffers from limitations. These are of two types. Firstly, the range of functions learnable is actually quite limited in practice. Secondly, the time to learn a new function increases quite considerably as the function becomes more complex. The underlying reason for both of these is that the search space of expressible hypotheses is a very large superset of the potentially occurring functions. This results in some functions being very time-consuming for the learner to reach, and potentially beyond the resource limits of the computer. The remainder of the section develops an improved learner by outlining the exact sources of these limitations and describing ILP techniques that can be used to overcome them. Aleph [12] was used in preference to Progol4.4 (which was used for the naive learner) as it included all the features necessary for the new implementation.

Removing Impossible Hypotheses. Given that the type of functions that should be learnable have been explicitly stated in the previous section, the first modification in building a better learner should be to restrict the expressible hypotheses to as close to this set as possible; there is no point in providing background knowledge which enables a hypothesis search space much larger than the known class of hypotheses.

It can be shown that the total number of executions, E, of a nest of Presburger loops of depth D is equal to a function of the form

$$E = \sum_{\beta_1=0}^{D} \sum_{\beta_2=0}^{D} \cdots \sum_{\beta_n=0}^{D} \left[\alpha_{\beta_1 \beta_2 \ldots \beta_n} \times \prod_{i=1}^{n} V_i^{\beta_i} \right]$$

where

$\alpha_{\beta_1 \ldots \beta_n} \in \mathbb{N}$

All coefficients $\alpha_{\beta_1 \ldots \beta_D} = 0$ if $(\sum_{x=1}^{n} \beta_x) > n$

$V_1 \ldots V_n$ are the variables

In other words, the function is sum of terms, where there is one constant term and the other terms are coefficients multiplied by various combinations of the variables. This can be a little difficult to comprehend in its generalized form, so consider the particular case for 3 levels of nesting and two variables.

$$\begin{aligned} E &= \sum_{\beta_1=0}^{3} \sum_{\beta_2=0}^{3} \left[\alpha_{\beta_1 \beta_2} V_1^{\beta_1} V_2^{\beta_2} \right] \\ &= \alpha_{00} + \alpha_{10} V_1 + \alpha_{20} V_1^2 + \alpha_{30} V_1^3 + \alpha_{01} V_2 + \alpha_{11} V_1 V_2 + \\ &\quad \alpha_{21} V_1^2 V_2 + \alpha_{02} V_2^2 + \alpha_{12} V_1 V_2^2 + \alpha_{03} V_2^3 \end{aligned}$$

In light of this, the background knowledge for the improved learner is altered by removing the `sum/3` and `product/3` predicates and replacing them with more specialised predicates directed towards learning functions of this form. Two new families of predicates are introduced in their place. Firstly, there is a new set of new term generating predicates `make_term/X` which multiply $X - 1$ variables together to give a term. These terms are then taken as input to the second new

```
make_term(Variable1,Variable2,Term) :-
   Term is Variable1 * Variable2.

make_term(Variable1,Variable2,Variable3,Term) :-
   Term is Variable1 * Variable2 * Variable3.

weighted_sum(Constant,Coefficient1,Term1,Output) :-
   Output is Constant + Coefficient1 * Term1.

weighted_sum(Constant,Coefficient1,Coefficient2,Term1,Term2,Output) :-
   Output is Constant + Coefficient1 * Term1 + Coefficient 2 * Term2.
```

Fig. 2. More Specialised Background Predicates

set of predicates `weighted_sum/X` which produces the weighted sum of $\frac{X}{2} - 1$ terms. Examples of some `make_term/X` and `weighted_sum/X` predicates are given in fig. 2. Obviously, the more predicates of each of these types that are added, the greater the class of hypotheses that are representable in the language, but also the larger that the search space of hypotheses will be. For practical reasons, the predicates are limited here to `make_term/1`, `make_term/2`, `make_term/3` and `make_term/4`, and `weighted_sum/4`, `weighted_sum/6` and `weighted_sum/8`. These prove adequate to learn the functions necessary in this paper, and in principle they can be extended trivially to allow more functions, with greater levels of nesting or more variables, to be learned.

Lazy Learning. Having altered the background knowledge to focus the search space on the valid function space, the issue of the unlearnability of some functions by the naive learner is now addressable.

The primary limitation on the functions that are learnable arises from the need to deal with numerical constants in the discovered equation. In the examples previously considered, these constants were either absent or small positive integers. Learning the relationship between a set of input variables and the number of loop executions requires the discovery of an equation expressing the loop executions as a function of the input variables. This presents problems for traditional ILP which is very poor at generating the numbers needed for such formulae. The problem occurs due to the construction of a bottom clause to guide the search.

The task of learning the equation requires the construction of a formula that is true for all the data. When the bottom clause is created, it will contain all possible formulae that are consistent with that datum. However with no limits on the possible formula, this will consist of an infinite number of formulae, the majority of which are inconsistent with any other data. Even in an extremely simplified situation in which there is only a single variable, V, and the formulae for the number of loop executions, E, are limited to the form $E = k + aV$, an infinite number of formulae should be placed in the bottom clause, each with a unique (k, a) pair; again the vast majority would still be found to be inconsistent with all other data points.

One approach to counter this problem is to limit the numbers that can be considered to some given set, but this limits the hypothesis space and may still lead to a large bottom clause and hence an over-large search space. For these reasons, an alternative technique is adopted which retains the size of the hypothesis space, but greatly limits the search space within it.

Lazy learning in ILP was first proposed by Srinivasan and Camacho [17]. Using this technique, clauses featuring constants can be added to the bottom clause as usual, but the constants themselves are only determined later during the subsequent search though the hypothesis space. This enables all the data to be used to determine the constants instead of just the single datum used to generate the bottom clause. Returning to the example given earlier, the bottom clause could have a single $E = k + aV$ clause added to it, with the actual values of k and a being calculated at search time from all the data. Clearly, this reduces the size of the bottom clause and consequently the search space, but crucially not the hypotheses that can be returned. This also decreases the time spent searching for the correct solution.[2]

In essence, lazy learning is used to transform the process of learning from a search of the *function* space to a search of the *functional form* space.

Symmetry Removal. One final improvement to the learner that can be made to reduce the size of the search space is the removal of symmetry. For example, if

```
target(A,B,C) :- make_term(A,B,D), weighted_sum(0,1,D,C).
```

fits the observed data exactly,[3] then so will

```
target(A,B,C) :- make_term(B,A,D), weighted_sum(0,1,D,C).
```

Ideally, this symmetry should be removed to reduce the space that must be searched. Symmetry also creates a problem in the construction of terms of the form A^nB^m; using `make_term/4`, there are 3 different ways to create A^2B based on permuting the order of As and Bs. The more arguments permitted to `make_term` and the larger the number of variables present, the greater this problem becomes.

The approach adopted to remove this symmetry is to prevent two clauses with the same meaning both being added to the bottom clause. This forces the learner to only consider one of the many cases when searching for a solution. Specifically, it is required that variable arguments to a `make_term/x` predicate are sorted in a non-decreasing order[4].

[2] Assuming that a more efficient method exists for finding the constants involved than a brute-force search through their possible values. In the case of fitting the functional forms mentioned in this paper, Gaussian Elimination [18] can used as this more efficient algorithm.

[3] i.e. C = AB.

[4] Similar constraints apply to the term arguments of `weighted_sum/X`. However, for clarity, all the discussion will focus on `make_term/X`.

```
% Symmetry suppressing version for use in creating the bottom clause
make_term(v(X),v(Y),v(Z),t(A)):-
  setting(stage,saturation),
  X =< Y, Y =< Z,
  A is X * Y * Z.

% Symmetry allowing version for use in searching
make_term(v(X),v(Y),v(Z),t(A)):-
  \+ setting(stage,saturation),
  A is X * Y * Z.
```

Fig. 3. Symmetry suppressing and allowing clauses for `make_term/4`

Using this approach, only `make_term(A,B,C)` can be added to the bottom clause if $A > B$ and only `make_term(B,A,C)` if the reverse is true. This immediately removes both sources of symmetry identified above. However, while this property may apply to A and B in the example used to create the bottom clause, it may not hold for other data in the set. Therefore, this requires splitting each background predicate into two: one is used when the bottom clause is constructed and the other used during the search for testing the coverage of a hypothesis. The first clause includes the ordering condition to suppress all but one of the symmetric cases. In contrast, the second allows all the symmetric cases to succeed. These pair of clauses are shown for `make_term/3` in fig. 3. Note that `setting(stage,saturation)` is an internal Aleph predicate that succeeds only when the bottom clause is under construction.

5 Results

Having identified weaknesses in a basic implementation of the loop bound learner and having suggested how they may be addressed, it is necessary to assess the extent to which the modified version overcomes the problems. In order to do this, implementations of both versions were coded for use in Aleph (the simple version being ported from an original implementation in Progol4.4 for fair comparison).

Three problems were chosen which serve to highlight the difference between the learners well. Firstly, a relatively simple example of three loops nested inside each other for which there was no interaction between the loop counters. Secondly, a nested loop structure featuring multiple constants without interaction between loop counters, and finally, an example of the sorting routine nested loops which has been mentioned extensively in this paper.

While benchmark suites do exist for WCET analysis, artificially generated problems are used in preference here. There are multiple reasons for this decision. The benchmark suites typically used are made of very simple functions and are not representative of actual real-time code, which is itself unavailable due to commercial confidentiality. Those benchmarks that do exist feature few functions with interaction effects between counters of nested loops, and those which do are no more complex than sort algorithms. Finally, as no existing techniques are able

ABC	Bottom Clause	Learning Time (s)
Naive Learner	56	2.48
Reformulated Lazy Learner	1139	8.41
+ Symmetry Removal	118	0.70

100ABC+1	Bottom Clause	Learning Time (s)
Naive Learner	88	N A
Reformulated Lazy Learner	1139	8.49
+ Symmetry Removal	118	0.73

A(A-1)/2	Bottom Clause	Learning Time (s)
Naive Learner	99	1.26
Reformulated Lazy Learner	19	0.08
+ Symmetry Removal	11	0.03

Fig. 4. Number of literals in bottom clause and search time for different learners on various problems

to automatically infer these loop bound relationships for interacting loops, there is no data to compare our performance to.

For each loop's structure, two pieces of information were recorded; the number of literals in the bottom clause and the time taken to find the solution. Tests were conducted with the original naive learner, with the reformulated lazy learner, and with the reformulated lazy learner with the symmetry removal turned on. Each learner received the same input data file, and was run in identical conditions. Bottom clause sizes were obtained from one particular datum in each data set; the datum being used for this purpose was the same for all learners.

Results of these experiments are shown in Fig 4. For all experiments, the correct formula was always discovered by the learner.

For the simple nested loop, the results surprising show that the reformulated learner has a larger bottom clause and search time. This counter-intuitive result is actually due to the naive learner excluding some solutions that should be considered potentially true from the solution space. This can be seen in the second example, where $100ABC+1$, which should rightly be in the search space, is not found at all by the naive learner.

For the sorting style loop, it can be seen that reformulation actually reduces the search space. This is because the number of functional forms that could be created for this space, is actually smaller than the number of functions that the naive leaner could return for a target with only a single parameter. The creation of learner operating on functional forms is clearly superior here, even in the absence of symmetry removal.

The effects of symmetry removal are evident throughout. Removing these unproductive clauses has a massive effect both directly on the bottom clause size and as a result on search time. While the reformulation and lazy learning expands the search space to not exclude those candidate hypotheses wrongly excluded by the naive learner, the symmetry reduction shrinks it down again.

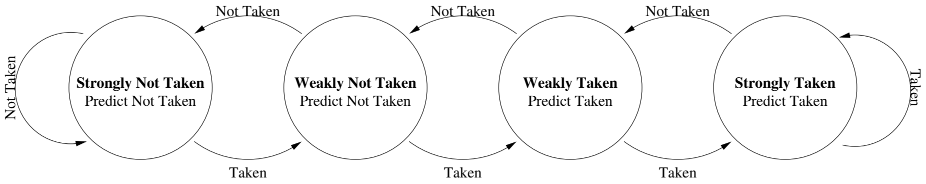


Fig. 5. Two bit prediction scheme

Crucially however, this is done while allowing a solution to always be found if it would be found in the absence of the symmetry removal. The large reduction in clause size and learning time illustrates the power and importance of this technique.

6 Related Work - ILP for Branch Prediction

The applicability of ILP in RTS is not limited to the loop bound learning task, and has also previously been tested on the problem of branch prediction analysis [19].

Modern pipelined microprocessors combine the approach of out-of-order execution with branch prediction and speculative execution to try to alleviate the problem of disrupting the instruction flow into the pipeline due to branches. A simple, but commonly used, dynamic branch prediction technique is an n-bit branch predictor [20] that uses the behaviour of the branch at its previous executions to predict its behaviour on the next occurrence. An n-bit predictor can be visualised as a finite state automaton (FSA) containing 2^n states. Each state predicts whether the branch will be taken or not at the next observation, and has deterministic transitions to other states based on the actual observed behaviour. An example of a two bit predictor is shown in Fig. 5. Common alternative branch prediction schemes include zero bit and one bit prediction.

While older processors have their branch prediction strategy well documented, for modern processors these details are normally commercially confidential. Without this information, WCET analysis must make conservative assumptions and produces unnecessarily pessimistic esimates. Bate and Kazakov [19] applied ILP learning to determine the type of branch predictor used by a processor. Test cases were produced for three configurations of hardware – zero bit, one bit and two bit predictors. The results of these executions were then fed into the learning engine. For each of the test cases the type of branch predictor was correctly learned. Processing was of the order of tens of seconds, an acceptable time for this type of task.

While this work demonstrates definite progress on an aspect of WCET analysis, it suffers from one major drawback. In the published work, the branch predictor is simply chosen from one of several common types. In contrast, at the forefront of chip design, branch predictors are becoming available based on novel prediction schemes. It is details of precisely these chips that are most likely to be commercially confidential. One possible approach is to assume the general task of identifying a finite state automaton with an unknown number of states, choice of initial state and transitions. While the background knowledge may not be difficult to describe, the task at this level appears very hard in the general case. A possible way ahead is to study the known existing variations in the realm of branch predictor design and encode building blocks from which the target automaton is likely to be built.

7 Conclusion

This paper has presented an application of ILP for solving a particular problem in the area of Real-Time Systems. Specifically, the issue of determining the Worst Case Execution Time of a piece of code has been considered. A particular aspect of this problem was tackled; the question of determining the number of executions of a loop body.

Having demonstrated the potential of the technique, a much improved learner was implemented. Using more advanced ILP techniques, it was possible to vastly expand the range of loops for which the number of executions could be learned. Furthermore, the implemented techniques reduced the time needed to learn the loop count substantially.

The results showed that it was possible to accurately achieve bounds for nested loops in which the outer loop counter effected the range of the inner counter. This goes beyond what can be achieved by other existing WCET techniques.

Building on the work presented here, it should be possible to apply ILP to other areas of WCET analysis. In addition to the use of ILP for determining facts about program flow, related work was also shown in which ILP could be used to determine features of the hardware used to execute the code. There are many open issues in this area of WCET analysis, including the simulation of caches and out-of-order pipelines.

References

1. Turing, A.: On Computable Numbers, with an Application to the Entscheidungsproblem (7936). The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life, Plus the Secrets of Enigma (2004)
2. Kazakov, D., Bate, I.: Towards new methods for developing real-time systems: Automatically deriving loop bounds using machine learning. In: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (2006)
3. Thesing, S., Souyris, J., Heckmann, R., Randimbivololona, F., Langenbach, M., Wilhelm, R., Ferdinand, C.: An abstract interpretation-based timing validation of hard real-time avionics software. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 625–632 (2003)

4. Coen-Porisini, A., Denaro, G., Ghezzi, C., Pezzè, M.: Using symbolic execution for verifying Safety-Critical systems. In: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 142–151 (2001)
5. Engblom, J.: Analysis of the execution time unpredictability caused by dynamic branch prediction. In: Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 152–159 (2003)
6. Colin, A., Puaut, I.: Worst case execution time analysis for a processor with branch prediction. The Journal of Real-Time Systems 18(2-3), 249–274 (2000)
7. Healy, C., Arnold, R., Müller, F., Whalley, D., Harmon, M.: Bounding pipeline and instruction cache performance. IEEE Transactions on Computers 48(1) (1999)
8. Li, Y.T.S., Malik, S.: Performance analysis of embedded software using implicit path enumeration. In: Proceedings of the 32nd Design Automation Conference, pp. 456–461 (1995)
9. McMinn, P.: Search-based software test data generation: a survey. Software Testing, Verification & Reliability 14(2), 105–156 (2004)
10. Wegener, J., Sthamer, H., Jones, B., Eyres, D.: Testing real-time systems using genetic algorithms. Software Quality Journal 6(2), 127–135 (1997)
11. Muggleton, S.: Learning from Positive Data. In: Muggleton, S. (ed.) ILP 1996. LNCS, vol. 1314, pp. 358–376. Springer, Heidelberg (1997)
12. Srinivasan, A.: The Aleph Manual. Computing Laboratory. Oxford University Press, Oxford (2000)
13. Ernst, M.: Dynamically Discovering Likely Program Invariants. PhD thesis, University of Washington (2000)
14. Todorovski, L., Dzeroski, S.: Declarative bias in equation discovery. In: Proceedings of the Fourteenth International Conference on Machine Learning, pp. 376–384 (1997)
15. Chapman, R.: Static Timing Analysis and Program Proof. PhD thesis, Department of Computer Science, University of York (1995)
16. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. s. n (1931)
17. Srinivasan, A., Camacho, R.: Numerical reasoning with an ILP system capable of lazy evaluation and customised search. The Journal of Logic Programming 40(2-3), 185–213 (1999)
18. Atkinson, K.: An introduction to numerical analysis. John Wiley, Chichester (1989)
19. Bate, I., Kazakov, D.: New directions in worst-case execution time analysis. In: IEEE Congress on Evolutionary Computation (IEEE CEC 2008) within 2008 IEEE World Congress on Computational Intelligence (WCCI 2008) (2008)
20. Smith, J.: A study of branch prediction strategies. In: Proceedings of the 8th International Symposium on Computer Architecture, pp. 135–148 (1981)

Discriminative Structure Learning of Markov Logic Networks

Marenglen Biba, Stefano Ferilli, and Floriana Esposito

Department of Computer Science, University of Bari
Via E.Orabona, 4 - 70125, Bari, Italy
{biba,ferilli,esposito}@di.uniba.it

Abstract. Markov Logic Networks (MLNs) combine Markov networks and first-order logic by attaching weights to first-order formulas and viewing these as templates for features of Markov networks. Learning the structure of MLNs is performed by state-of-the-art methods by maximizing the likelihood of a relational database. This can lead to suboptimal results given prediction tasks. On the other hand better results in prediction problems have been achieved by discriminative learning of MLNs weights given a certain structure. In this paper we propose an algorithm for learning the structure of MLNs discriminatively by maximimizing the conditional likelihood of the query predicates instead of the joint likelihood of all predicates. The algorithm chooses the structures by maximizing conditional likelihood and sets the parameters by maximum likelihood. Experiments in two real-world domains show that the proposed algorithm improves over the state-of-the-art discriminative weight learning algorithm for MLNs in terms of conditional likelihood. We also compare the proposed algorithm with the state-of-the-art generative structure learning algorithm for MLNs and confirm the results in [22] showing that for small datasets the generative algorithm is competitive, while for larger datasets the discriminative algorithm outperfoms the generative one.

1 Introduction

Many real-world application domains are characterized by both uncertainty and complex relational structure. Statistical learning focuses on the former, and relational learning on the latter. Probabilistic Inductive Logic Programming (PILP) [7] or Statistical Relational Learning [10] aim at combining the power of both. PILP and SRL can be viewed as combining ILP principles (such as refinement operators) with statistical learning. One of the representation formalisms in this area is Markov Logic which subsumes both finite first-order logic and probabilistic graphical models as special cases [30]. Upon this formalism, Markov Logic Networks (MLNs) can be built serving as templates for constructing Markov Networks (MNs). In Markov Logic a weight is attached to each clause and learning an MLN consists in structure learning (learning the clauses) and weight learning (setting the weight of each clause).

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 59–76, 2008.

In [30] structure learning was performed through CLAUDIEN [6] followed by a weight learning phase in which maximum pseudo-likelihood [1] weights were learned for each clause. In [14] structure is learned in a single phase using weighted pseudo-likelihood as the evaluation measure in a beam search. The algorithm performs systematic greedy search being therefore very suscetible to local optima. The state-of-the-art algorithm for generative structure learning is that in [24] which follows a bottom-up approach trying to consider fewer candidates for evaluation. This algorithm uses a propositional Markov network learning method to construct template networks that guide the construction of candidate clauses. In this way, it generates fewer clauses for evaluation.

Generative approches optimize the joint distribution of all the variables. This can lead to suboptimal results for predictive tasks because of the mismatch between the objective function used (likelihood or a function thereof) and the goal of classication (maximizing accuracy or conditional likelihood). In contrast discriminative approaches maximize the conditional likelihood of a set of outputs given a set of inputs [16] and this often produces better results for prediction problems. In [31] the voted perceptron based algorithm for discriminative weight learning of MLNs was shown to greatly outperform maximum-likelihood and pseudo-likelihood approches for two real-world prediction problems. Recently, the algorithm in [21], outperforming the voted perceptron became the state-of-the-art method for discriminative weight learning of MLNs. However, both discriminative approches to MLNs learn weights for a fixed structure, given by a domain expert or learned through another structure learning method (usually generative). Better results could be achieved if the structure could be learned in a discriminative fashion. Unfortunately, the computational cost of optimizing structure and parameters for conditional likelihood is prohibitive. In this paper we show that the simple approximation of choosing structures by maximizing conditional likelihood while setting parameters by maximum likelihood can produce better results in terms of predictive accuracy. Structures are scored through a very fast inference algorithm MC-SAT [27] whose lazy version Lazy-MC-SAT [28] greatly reduces memory requirements, while parameters are learned through a quasi-Newton optimization method like L-BFGS that has been found to be much faster [34] than iterative scaling initially used for Markov Networks' weight learning [5]. We show through experiments in two real-world domains that the proposed algorithm improves over the state-of-the-art algorithm of [21] in terms of conditional likelihood of the query predicates.

Discriminative approaches may not always provide the highest classification accuracy. An empirical and theoretical comparison of discriminative and generative classifiers (logistic regression and Naïve Bayes (NB)) is given in [22]. It is shown that for small sample sizes the generative NB classier can outperform a discriminatively trained model. This is consistent with the fact that, for the same representation, discriminative training has lower bias and higher variance than generative training, and the variance term dominates at small sample sizes [8,9]. For the dataset sizes typically found in practice, however, the results in [12,22,11] all support the choice of discriminative training. An experimental comparison of discriminative and generative parameter training on both discriminatively and generatively structured Bayesian Network classifiers has been performed in [25]. In this paper we perform an experimental comparison between generative and discriminative structure learning algorithms for MLNs and confirm the results in [22] in the case of MLNs by showing that on a small dataset the generative algorithm is competitive, while on a larger dataset the discriminative algorithm outperforms the generative one in terms of conditional likelihood.

The paper is organized as follows: in Section 2 we introduce MNs and MLNs, in Section 3 we describe existing generative structure learning and discriminative weight learning approaches for MLNs, Section 4 introduces the iterated local search metaheuristic, Section 5 describes a discriminative algorithm for MLNs structure learning, Section 6 presents the experiments, Section 7 describes related work and we conclude in Section 8;

2 Markov Networks and Markov Logic Networks

A Markov network (also known as Markov random field) is a model for the joint distribution of a set of variables $\mathrm{X} = (\mathrm{X}_1,\mathrm{X}_2,\ldots,\mathrm{X}_n) \in \chi$ [5]. It is composed of an undirected graph G and a set of potential functions. The graph has a node for each variable, and the model has a potential function ϕ_k for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by:

$$P(X = x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the partition function, is given by:

$$Z = \sum_{x \in \chi}\prod_k \phi_k(x_{\{k\}})$$

Markov networks are often conveniently represented as log-linear models, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to:

$$P(X = x) = \frac{1}{Z}\exp(\sum_j w_j f_j(x))$$

A feature may be any real-valued function of the state. We will focus on binary features, $f_j \in \{0,1\}$. In the most direct translation from the potential-function form, there is one feature corresponding to each possible state x_k of each clique, with its weight being $\log(\phi(x_{\{k\}})$. This representation is exponential in the size of the cliques. However a much smaller number of features (e.g., logical functions of the state of the clique) can be specified, allowing for a more compact representation than the potential-function form, particularly when large cliques are present. MLNs take advantage of this.

A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in Markov logic is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

A Markov logic network [30] L is a set of pairs $(F_i; w_i)$, where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_p\}$ it defines a Markov network $M_{L;C}$ as follows:

1. $M_{L;C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.

2. $M_{L;C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L. Thus there is an edge between two nodes of $M_{L;C}$ iff the corresponding ground predicates appear together in at least one grounding of one formula in L. An MLN can be viewed as a template for constructing Markov networks. The probability distribution over possible worlds x specified by the ground Markov network $M_{L;C}$ is given by

$$P(X = x) = \frac{1}{Z} \exp(\sum_{i=1}^{F} w_i n_i(x))$$

where F is the number of formulas in the MLN and $n_i(x)$ is the number of true groundings of F_i in x. As formula weights increase, an MLN increasingly resembles a purely logical KB, becoming equivalent to one in the limit of all infinite weights.

In this paper we focus on MLNs whose formulas are function-free clauses and assume domain closure (it has been proven that no expressiveness is lost), ensuring that the Markov networks generated are finite. In this case, the groundings of a formula are formed simply by replacing its variables with constants in all possible ways.

3 Structure and Parameter Learning of MLNs

3.1 Generative Structure Learning of MLNs

One of the approaches for learning Markov Network weights is iterative scaling [5]. However, maximizing the likelihood (or posterior) using a quasi-Newton optimization method like L-BFGS has recently been found to be much faster

[34]. Regarding structure learning, the authors in [5] induce conjunctive features by starting with a set of atomic features (the original variables), conjoining each current feature with each atomic feature, adding to the network the conjunction that most increases likelihood, and repeating. The work in [23] extends this to the case of conditional random fields, which are Markov networks trained to maximize the conditional likelihood of a set of outputs given a set of inputs.

The first attempt to learn MLNs was that in [30], where the authors used CLAUDIEN [6] to learn the clauses of MLNs and then learned the weights by maximizing pseudo-likelihood. In [14] another method was proposed that combines ideas from ILP and feature induction of Markov networks. This algorithm, that performs a beam or shortest first search in the space of clauses guided by a weighted pseudo-log-likelihood (WPLL) measure [1], outperformed that of [30]. Recently, in [24] a bottom-up approach was proposed in order to reduce the search space. This algorithm uses a propositional Markov network learning method to construct template networks that guide the construction of candidate clauses. In this way, it generates fewer candidates for evaluation. For every candidate structure, in both [14,24] the parameters that optimize the WPLL are set through L-BFGS that approximates the second-derivative of the WPLL by keeping a running finite-sized window of previous first-derivatives.

3.2 Discriminative Parameter Learning of MLNs

Learning MLNs in a discriminative fashion has produced for predictive tasks much better results than generative approaches as the results in [31] show. In this work the voted-perceptron algorithm was generalized to arbitrary MLNs by replacing the Viterbi algorithm with a weighted satisfiability solver. The new algorithm is essentially gradient descent with an MPE approximation to the expected sufficient statistics (true clause counts) and these can vary widely between clauses, causing the learning problem to be highly ill-conditioned, and making gradient descent very slow. In [21] a preconditioned scaled conjugate gradient approach is shown to outperform the algorithm in [31] in terms of learning time and prediction accuracy. This algorithm is based on the scaled conjugate gradient method and very good results are obtained with a simple approach: per-weight learning weights, with the weight's learning rate being the global one divided by the corresponding clause's empirical number of true groundings.

However, for both these algorithms the structure is supposed to be given by an expert or learned previously and they focus only on the parameter learning task. This can lead to suboptimal results if the clauses given by an expert do not capture the essential dependencies in the domain in order to improve classification accuracy. On the other side, since to the best of our knowledge, no attempt has been made to learn the structure of MLNs discriminatively, the clauses learned by generative structure learning algorithms tend to optimize the joint distribution of all the variables and applying discriminative weight learning after the structure has been learned generatively may lead to suboptimal results since the initial goal of the learned structure was not to discriminate query predicates.

4 Iterated Local Search

Many widely known and high-performance local search algorithms make use of randomized choice in generating or selecting candidate solutions for a given combinatorial problem instance. These algorithms are called stochastic local search (SLS) algorithms [13] and represent one of the most successful and widely used approaches for solving hard combinatorial problem. Many "simple" SLS methods come from other search methods by just randomizing the selection of the candidates during search, such as Randomized Iterative Improvement (RII), Uniformed Random Walk, etc. Many other SLS methods combine "simple" SLS methods to exploit the abilities of each of these during search. These are known as Hybrid SLS methods [13]. ILS is one of these metaheuristics because it can be easily combined with other SLS methods.

One of the simplest and most intuitive ideas for addressing the fundamental issue of escaping local optima is to use two types of SLS steps: one for reaching local optima as efficiently as possible, and the other for effectively escaping local optima. ILS methods [13,20] exploit this key idea, and essentially use two types of search steps alternatingly to perform a walk in the space of local optima w.r.t the given evaluation function. The algorithm works as follows: The search process starts from a randomly selected element of the search space. From this initial candidate solution, a locally optimal solution is obtained by applying a subsidiary local search procedure. Then each iteration step of the algorithm consists of three major steps: first a perturbation method is applied to the current candidate solution *s*; this yields a modified candidate solution *s'* from which in the next step a subsidiary local search is performed until a local optimum *s"* is obtained. In the last third step, an acceptance criterion is used to decide from which of the two local optima *s* or *s'* the search process is continued. The algorithm can terminate after some steps have not produced improvement or simply after a certain number of steps. The choice of the components of the ILS has a great impact on the performance of the algorithm.

In general, it is not straightforward to decide whether to use a systematic or SLS algorithm in a certain task. Systematic and SLS algorithms can be considered complementary to each other. SLS algorithms are advantageous in many situations, particularly if reasonably good solutions are required within a short time, if parallel processing is used and if knowledge about the problem domain is rather limited. In other cases, when time constraints are less important and some knowledge about the problem domain can be exploited, systematic search may be a better choice. Structure learning of MLNs is a hard optimization problem due to the large space to be explored, thus SLS methods are suitable for finding solutions of high quality in short time. Moreover, one of the key advantages of SLS methods is that they can greatly speed up learning through parallel processing, where speedups proportional to the number of CPUs can be achieved [13].

5 Discriminative Structure Learning of MLNs

In this section we describe our proposal for tailoring ILS metaheuristic to the problem of learning the structure of MLNs and describe how weights are set and how structures are scored. The approach we follow is similar to [12] where Bayesian Networks were learned by setting weights through maximum likelihood and choosing structures by maximizing conditional likelihood.

5.1 Search Strategy

Algorithm 1 (Discriminative Structure Learning - DSL) iteratively adds the best clause to the current MLN until two consecutive steps have not produced improvement (however other stopping criteria could be applied). It can start from an empty network or from an existing KB. Like in [30,14] we add all unit clauses (single predicates) to the MLN. The initial weights are learned in *LearnWeights* through L-BFGS and the initial structure is scored in *ComputeCLL* through MC-SAT. The search for the best clause is performed in *SearchBestClause* described by Algorithm 2. The algorithm performs an iterated local search to find the best clause to add to the MLN. It starts by randomly choosing a unit clause CL_C in the search space. Then it performs a greedy local search to efficiently reach a local optimum CL_S. At this point, a perturbation method is applied leading to the neighbor CL'_C of CL_S and then a greedy local search is applied to CL'_C to reach another local optimum CL'_S . The *accept* function decides whether the search must continue from the previous local optimum CL_C or from the last found local optimum CL'_S (*accept* can perform random walk or iterative improvement in the space of local optima).

Careful choice of the various components of Algorithm 2 is important to achieve high performance. The clause perturbation operator (flipping the sign of literals, removing literals or adding literals) has the goal to jump in a different region of the search space where search should start with the next iteration. There can be strong or weak perturbations which means that if the jump in the search space is near to the current local optimum the subsidiary local search procedure $LocalSearch_{II}$ (Algorithm 3) may fall again in the same local optimum and enter regions with the same value of the objective function called *plateau*, but if the jump is too far, $LocalSearch_{II}$ may take too many steps to reach another good solution. In our algorithm we use only strong perturbations, i.e., we always re-start from unit clauses (in future work we intend to dynamically adapt the nature of the perturbation). Regarding the procedure $LocalSearch_{II}$, we decided to use an iterative improvement approach (the walk probability is set to zero and the best clause is always chosen in $step_{II}$) in order to balance intensification (greedily increase solution quality by exploiting the evaluation function) and diversification (randomness induced by strong perturbation to avoid search

Algorithm 1. Discriminative Structure Learning

```
Input: P:set of predicates, MLN:Markov Logic Network, RDB:Relational Database
CLS = All clauses in MLN $\cup$ P;
LearnWeights(MLN,RDB);
BestScore = ComputeCLL(MLN,RDB);
repeat
  BestClause = SearchBestClause(P,MLN,BestScore,CLS,RDB);
  if BestClause $\neq$ null then
        Add BestClause to MLN;
        BestScore = ComputeCLL(MLN,RDB);
  end if
until BestClause = null for two consecutive steps
return MLN
```

stagnation). In future work we intend to further weaken intensification by using a higher walk probability. Finally, the *accept* function always accepts the best solution found so far.

5.2 Setting Parameters through Maximum Likelihood

For every candidate structure, the parameters that optimize the WPLL are set through L-BFGS. As pointed out in [14] a potentially serious problem that arises when evaluating candidate clauses using WPLL is that the optimal (maximum WPLL) weights need to be computed for each candidate. Since this involves numerical optimization, and needs to be done millions of times, it could easily make the algorithm too slow. In [23,5] the problem is addressed by assuming that the weights of previous features do not change when testing a new one. Surprisingly, the authors in [14] found this to be unnecessary if the very simple approach of initializing L-BFGS with the current weights (and zero weight for a new clause) is used. Although in principle all weights could change as the result of introducing or modifying a clause, in practice this is very rare. Second-order, quadratic-convergence methods like L-BFGS are known to be very fast if started near the optimum [34]. This is what happened in [14]: L-BFGS typically converges in just a few iterations, sometimes one. We use the same approach for setting the parameters that optimize the WPLL.

5.3 Efficient Structure Scoring

In order to score MLN structures, we need to perform inference over the network. A very fast algorithm for inference in MLNs is MC-SAT [27]. Since probabilistic inference methods like MCMC or belief propagation tend to give poor results when deterministic or near-deterministic dependencies are present, and logical ones like satisfiability testing are inapplicable to probabilistic dependencies, MC-SAT combines ideas from both MCMC and satisfiability to handle probabilistic, deterministic and near-deterministic dependencies that are typical of statistical

Algorithm 2. SearchBestClause

```
Input: P: set of predicates, MLN: Markov Logic Network, BestScore: current best score, CLS: List of clauses, RDB: Relational Database)
$CL_C$ = Random Pick a clause in CLS $\cup$ P;
$CL_S$ = $\mathit{LocalSearch}_{II}(CL_S)$;
BestClause = $CL_S$;
repeat
  $CL'_C$ = $\mathit{Perturb}(CL_S)$;
  $CL'_S$ = $\mathit{LocalSearch}_{II}(CL'_C$,MLN,BestScore);
  if ComputeCLL(BestClause,MLN,RDB) $\leq$ ComputeCLL($CL'_S$,MLN,RDB) then
    BestClause = $CL'_S$;
    Add BestClause to MLN;
    BestScore = ComputeCLL($CL'_S$,MLN,RDB)
  end if
  $CL_S$ = accept($CL_S$,$CL'_S$);
until two consecutive steps have not produced improvement
Return BestClause
```

relational learning. MC-SAT was shown to greatly outperform Gibbs sampling and simulated tempering in two real-world datasets regarding entity resolution and collective classification.

Even though MC-SAT is a very fast inference algorithm, scoring candidate structures at each step can be potentially very expensive since inference has to be performed for each candidate clause added to the current structure. One problem that arises is that fully instantiating a finite first-order theory requires memory in the order of the number of constants raised to the length of the clauses, which signicantly limits the size of domains where the problem can still be tractable. To avoid this problem, we used a lazy version of MC-SAT, Lazy-MC-SAT [28] which reduces memory and time by orders of magnitude compared to MC-SAT. Before Lazy-MC-SAT was introduced, the LazySat algorithm [33] was shown to greatly reduce memory requirements by exploiting the sparseness of relational domains (i.e., only a small fraction of ground atoms are true, and most clauses are trivially satisfied). The authors in [28] generalize the ideas in [33] by proposing a general method for applying lazy inference to a broad class of algorithms such as other SAT solvers or MCMC methods. Another problem is that even though Lazy-MC-SAT makes memory requirements tractable, it can take too much time to construct the Markov random field in the first step of MC-SAT for every candidate structure.

To make the execution of Lazy-MC-SAT tractable for every candidate structure, we use the following simple heuristics: 1) We score through Lazy-MC-SAT only those candidates that produce an improvement in WPLL. Once the parameters are set through L-BFGS, it is straightforward to compute the gain in WPLL for each candidate. This reduces the number of candidates to be scored through Lazy-MC-SAT for a gain in CLL. 2) We pose a memory limit for Lazy-MC-SAT on the clause activation phase and this greatly speeds up the whole inference

Algorithm 3. LocalSearch$_{II}$

```
Input: (CL$_C$: current clause)
wp: walk probability, the probability of performing an improvement step or a random step
repeat
    NBHD = Neighborhood of CL$_C$ constructed using the clause construction operators;
    CL$_S$ = $Step_{RII}$(CL$_C$,NBHD,wp);
    CL$_C$ = CL$_S$;
until two consecutive steps do not produce improvement
Return CL$_S$;

$Step_{RII}$(CL$_C$,NBHD,wp)
U = random(]0,1]); random number using a Uniform Probability Distribution
if ($U \leq wp$) then
    CL$_S$ = step$_{URW}$(CL$_C$,NBHD)
    Uninformed Random Walk: randomly choose a neighbor from NBHD
else
    CL$_S$ = step$_{II}$(CL$_C$,NBHD)
    Iterative Improvement: choose the best among the improving neighbours in NBHD.
    If there is no improving neighbor choose the minimally worsening one
end if
Return CL$_S$
```

task. Although in principle this limit can reduce the accuracy of inference, we found that in most cases the memory limit is never reached making the overall inference task very fast. 3) We pose a time limit in the clause activation phase in order to avoid those rare cases where the step takes a very long time to be completed. For most candidate structures such a time limit is never reached and for those rare cases where time limit is reached, inference is performed using the activated clauses within the limit.

We found that these simple approximations greatly speed up the scoring of each structure at each step. Filtering the potential candidates through the gain in WPLL can in principle exclude good candidates due to the mismatch between the optimization of WPLL and that of CLL. However, we empirically found that most candidates not improving WPLL, did not improve CLL. Further investigation on this issue may help to select better or more candidates to be scored through Lazy-MC-SAT.

6 Experiments

Through experimental evaluation we want to answer the following questions:

(Q1) Is the DSL algorithm competitive with state-of-the-art discriminative training algorithms of MLNs?

(Q2) Is the DSL algorithm competitive with the state-of-the-art generative algorithm for structure learning of MLNs?

(Q3) Is the DSL algorithm competitive with pure probabilistic approaches such as Naïve Bayes and Bayesian Networks?

(Q4) Is the DSL algorithm competitive with state-of-the-art ILP systems for the task of structure learning of MLNs?

(Q5) Does the DSL algorithm always perform better than BUSL for classification tasks? If not, are there any regimes in which each algorithm performs better?

Regarding question **(Q1)** we have to compare DSL with Preconditioned Scaled Conjugate Gradient (PSCG), the state-of-the-art discriminative training algorithm for MLNs proposed in [21]. It must be noted that this algorithm takes in input a fixed structure, and with the clausal knowledge base we use in our experiments for Cora (each dataset comes with a hand-coded knowledge base), PSCG has achieved the best published results. We also exclude the approach of adapting the rule set and then learning weights with PSCG, since it would be computationally very expensive.

To answer question **(Q2)** we have to perform experimental comparison with the Bottom-Up Structure Learning (BUSL) algorithm [24] which is the state-of-the-art algorithm for this task. Since in principle, the MLNs structure can be learned using any ILP technique it would be interesting to know how the DSL compares to ILP approaches. In [14], the proposed algorithm based on beam search (BS) was shown to outperform FOIL and the state-of-the-art ILP system Aleph for the task of learning MLNs structure. Moreover, BS outperformed both Naïve Bayes and Bayesian Networks in terms of CLL and AUC. Since in [24] was shown that BUSL outperforms the BS algorithm of [14], our baseline for questions **(Q2)**, **(Q3)** and **(Q4)** is again BUSL. Regarding question **(Q5)**, we compare DSL and BUSL on two datasets with the goal of discovering regimes in which each one can perform better. We will use two datasets, one of which can be considered of small size and the other one of much larger size.

6.1 Datasets

We carried out experiments on two publicly-available databases: the UW-CSE database (available at http://alchemy.cs.washington.edu/data/uw-cse) used by [14,30,24] and the Cora dataset originally labeled by Andrew McCallum. Both represent standard relational datasets and are used for two important relational tasks: Cora for entity resolution and UW-CSE for social network analysis. For Cora we used a cleaned version from [32], with five splits for cross-validation.

The published UW-CSE dataset consists of 15 predicates divided into 10 types. Types include: publication, person, course, etc. Predicates include: Student (person), Professor(person), AdvisedBy(person1, person2),TaughtBy(course, person, quarter), Publication (paper, person) etc. The dataset contains a total of 2673 tuples (true ground atoms, with the remainder assumed false). We used the hand-coded knowledge base provided with it, which includes 94 formulas stating regularities like: each student has at most one advisor; if a student is an author of a paper, so is her advisor; etc. Notice that these statements are not always true, but are typically true. The task is to predict who is whose advisor from information about coauthorships, classes taught, etc. More precisely, the query atoms are all groundings of AdvisedBy(person1, person2), and the evidence atoms are all groundings of all other predicates except Student(person) and Professor(person), corresponding to the Partial Information scenario in [30].

The Cora dataset consists of 1295 citations of 132 different computer science papers, drawn from the Cora Computer Science Research Paper Engine. The task is to predict which citations refer to the same paper, given the words in their author, title, and venue fields. The labeled data also specify which pairs of author, title, and venue fields refer to the same entities. We performed experiments for each field in order to evaluate the ability of the model to deduplicate fields as well as citations. Since the number of possible equivalences is very large, like the authors did in [21] we used the canopies found in [32] to make this problem tractable. The dataset contains a total of 70367 tuples (true and false ground atoms, with the remainder assumed false).

6.2 Systems and Methodology

We implemented the DSL algorithm in the Alchemy package [15]. We used the implementation of L-BFGS and Lazy-MC-SAT in Alchemy to learn maximum WPLL weights and compute CLL during clause search.

Regarding parameter learning, we compared our algorithm performance with the state-of-the-art algorithm PSCG of [21] for discriminative weight learning of MLNs. This algorithm takes as input an MLN and the evidence (groundings of non-query predicates) and discriminatively trains the MLN to optimize the CLL of the query predicates given evidence. DSL and PSCG both optimize the CLL of the query predicates and a comparison between these algorithms would be useful to understand if learning automatically the clauses from scratch can improve over hand-coded MLN structures in terms of classification accuracy of the query predicates given evidence. We performed all the experiments on a 2.13 GHz Intel Core2 Duo CPU. For the UW-CSE dataset we trained PSCG on the hand-coded knowledge base provided with the dataset. We used the implementation of PSCG in the Alchemy package and ran this algorithm with the default parameters for 10 hours. For the Cora dataset, for PSCG we report the results obtained in [21]. For both datasets, for the DSL algorithm we used the following parameters: The mean and variance of the Gaussian prior were set to 0 and 100, respectively; maximum variables per clause = 4; maximum predicates per clause = 4; penalization of weighted pseudo-likelihood = 0.01 for UW-CSE and 0.001 for Cora. For L-BFGS we used the following parameters: maximum iterations = 10,000 (tight) and 10 (loose); convergence threshold = 10^{-5} (tight) and 10^{-4} (loose). For Lazy-MC-SAT during learning we used the following parameters: memory limit = 200MB, maximum number of steps for Gibbs sampling = 100, simulated annealing temperature = 0.5.

Regarding BUSL, for both datasets, we used the following parameters: The mean and variance of the Gaussian prior were set to 0 and 100, respectively; maximum variables per clause: 5 for UW-CSE and 6 for Cora; maximum predicates per clause = 6; penalization of WPLL: 0.01 for UW-CSE and 0.001 for Cora. minWeight 0.5 for UW-CSE and 0.01 for Cora; For L-BFGS we used the following parameters: maximum iterations = 10,000 (tight) and 10 (loose); convergence threshold = 10^{-5} (tight) and 10^{-4} (loose).

In the UW-CSE domain, we used the same leave-one-area-out methodology as in [30]. In the Cora domain, we performed 5-fold cross-validation. For each train/test split, one of the training folds is used as tuning set for computing the CLL. For each system on each test set, we measured the CLL and the area under the precision-recall curve (AUC) for the query predicates. The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives (i.e., ground atoms that are false and predicted to be false). The CLL of a query predicate is the average over all its groundings of the ground atoms log-probability given evidence. The precision-recall curve for a predicate is computed by varying the CLL threshold above which a ground atom is predicted to be true; i.e. the predicates whose probability of being true is greater than the threshold are positive and the rest are negative. For the computation of AUC we used the package of [3].

It must be noted also that since the goal of the experimental study was to verify that DSL is competitive to other state-of-the-art techniques, and not to boost performance, we did not try to optimize any parameter.

6.3 Results

After learning the structure discriminatively, we performed inference on the test fold for both datasets by using MC-SAT with number of steps = 10000 and simulated annealing temperature = 0.5. For each experiment, on the test fold all the groundings of the query predicates were commented: advisedBy for the UW-CSE dataset (professor and student are also commented) and sameBib, sameTitle, sameAuthor and sameVenue for Cora. MC-SAT produces probability outputs for every grounding of the query predicate on the test fold. We used these values to compute the average CLL over all the groundings.

The results for all algorithms on the UW-CSE dataset are reported in Table 1 where CLL is averaged over all the groundings of the predicate advisedBy in the test fold. Regarding DSL and PSCG, in this domain DSL performs better in terms of CLL in all folds of the dataset and in two folds in terms of AUC. Overall, DSL performs better in terms of CLL and worse in terms of AUC. For the Cora dataset the results are reported in Table 2 where for each query predicate we report the average of CLL of its groundings over the test fold (for each predicate, training is performed on four folds and testing on the remaining one in a 5-fold cross-validation). For Cora, DSL performs better in terms of CLL but worse in terms of AUC for all the query predicates. We observed empirically on each fold that the performances in terms of CLL and AUC were always balanced, a slightly better performance in CLL always resulted in a slightly worse performance in terms of AUC and vice versa. Since CLL determines the quality of the probability predictions output by the algorithm, DSL outperforms PSCG in terms of the

Table 1. Results for DSL, PSCG and BUSL for the query predicate advisedBy in the UW-CSE domain

	DSL		PSCG		BUSL	
area	CLL	AUC	CLL	AUC	CLL	AUC
language	-0.036±0.012	0.032	-0.049±0.016	0.011	-0.024±0.008	0.115
graphics	-0.016±0.002	0.009	-0.023±0.005	0.005	-0.014±0.002	0.007
systems	-0.021±0.004	0.023	-0.026±0.005	0.069	-0.295±0.000	0.007
theory	-0.019±0.005	0.031	-0.028±0.007	0.101	-0.013±0.003	0.032
ai	-0.021±0.002	0.014	-0.032±0.005	0.034	-0.019±0.003	0.013
Overall	-0.023±0.005	0.022	-0.032±0.008	0.044	-0.073±0.003	0.035

Table 2. Results for DSL, PSCG and BUSL for all query predicates in the Cora domain

	DSL		PSCG		BUSL	
predicate	CLL	AUC	CLL	AUC	CLL	AUC
sameBib	-0.116±0.001	0.495	-0.291±0.003	0.990	-0.566±0.001	0.138
sameTitle	-0.076±0.005	0.430	-0.231±0.014	0.953	-0.100±0.004	0.419
sameAuthor	-0.126±0.007	0.590	-0.182±0.013	0.999	-0.834±0.009	0.323
sameVenue	-0.100±0.003	0.247	-0.444±0.012	0.823	-0.232±0.005	0.218
Overall	-0.105±0.004	0.441	-0.287±0.000	0.941	-0.433±0.005	0.275

ability to predict correctly the query predicates given evidence. However, since AUC is useful to predict the few positives in the data, PSCG produces better results for only positive examples. Hence, these results answer question **(Q1)**. The worse performance of DSL in terms of AUC can be due to the approach of DSL of optimizing the conditional likelihood during structure learning that may lead to overfitting. This issue deserves further investigation. However, it must be noted that PSCG has achieved the best published results on Cora in terms of AUC [21]. Moreover, since we did not try to optimize parameters, it is interesting to know if the results would change with a higher number of variables and literals per clause. For DSL, we plan to perform more extensive experiments with larger search parameters in order to boost performance.

Regarding DSL against BUSL, the results show that DSL performs better than BUSL in terms of CLL on both datasets. It must be noted, however, that for UW-CSE, BUSL performed generally better than DSL, but produced very low results in one fold. In terms of AUC, BUSL performs slightly better on the UW-CSE dataset while in the Cora dataset DSL outperforms BUSL. Therefore, questions **(Q2)**, **(Q3)** and **(Q4)** can be answered affirmatively. DSL is competitive with BUSL even though for BUSL, in the UW-CSE domain, we used optimized parameters taken from [24] in terms of number of variables and literals per clause, while for DSL we did not optimize any parameter.

Regarding question **(Q5)**, the goal was whether previous results of [22] carry on to MLNs, that on small datasets generative approaches can perform better than discriminative ones. The UW-CSE dataset with a total of 2673 tuples can be considered of much smaller size compared to Cora that has 70367 tuples. The results of Table 1 show that on the UW-CSE dataset, the generative algorithm BUSL performs better in terms of AUC and is competitive in terms of CLL since it underperforms DSL only because of the low results in the *systems* fold of the dataset. Thus we can answer question **(Q5)** confirming the results in [22] that on small datasets generative approaches can perform better than discriminative ones, while for larger datasets discriminative approches outperform generative ones.

Finally, we give examples of clauses from MLN structures learned for both datasets (we omit the relative weights). For the UW-CSE dataset examples of learned clauses are:

$$advisedBy(a1, a2) \vee inPhase(a1, a3) \vee inPhase(a2, a4)$$

$$\neg student(a1) \vee position(a2, a3) \vee advisedBy(a1, a2)$$

These clauses model the relation *advisedBy* between students and professors. In the first clause, *a1* and *a2* are variables that denote persons (students or professors) and the predicate *inPhase* states for each of these persons the phase of their university career. In the second clause, the predicate *position* relates the person denoted by *a2* (only professors have a position) to his university position.

For Cora, examples of learned clauses are the following:

$sameAuthor(a1,a2) \vee \neg hasWordAuthor(a1,a3) \vee \neg hasWordAuthor(a2,a3) \vee a1 = a2$

$$\neg title(a1, a2) \vee \neg title(a3, a2) \vee \neg sameBib(a3, a1)$$

In the first clause, *a1* and *a2* denote author fields while the predicate *hasWordAuthor* relates author fields to words contained in these fields. In the second rule the predicate *title* relates titles to their respective citations and the predicate *sameBib* is true if both its arguments denote the same citation.

7 Related Work

Many works in the SRL or PILP area have addressed classification tasks. Our discriminative method falls among those approaches that tightly integrate ILP and statistical learning in a single step for structure learning. The earlier works in this direction are those in [4,26] that employ statistical models such as maximum entropy modeling in [4] and logistic regression in [26]. These approaches can be computationally very expensive. A simpler approach that integrates FOIL [29] and Naïve Bayes is nFOIL proposed in [17]. This approach interleaves the steps of generating rules and scoring them through CLL. In another work [2] these steps are coupled by scoring the clauses through the improvement in classification accuracy. This algorithm incrementally builds a Bayes net during rule learning and each candidate rule is introduced in the network and scored by whether it improves the performance of the classifier. In a recent approach [19], the kFOIL system integrates ILP and support vector learning. kFOIL constructs the feature space by leveraging FOIL search for a set of relevant clauses. The search is driven by the performance obtained by a support vector machine based on the resulting kernel. The authors showed that kFOIL improves over nFOIL. Recently, in TFOIL [18], Tree Augmented Naïve Bayes, a generalization of Naïve Bayes was integrated with FOIL and it was shown that TFOIL outperforms nFOIL.

The most closely related approach to the DSL algorithm is nFOIL (and TFOIL as an extension) which is the first system in literature to tightly integrate feature construction and Naïve Bayes. Such a dynamic propositionalization was shown to be superior compared to static propositionalization approaches that use Naïve Bayes only to post-process the rule set. The approach is different from ours in that nFOIL selects features and parameters that jointly optimize a probabilistic score on the training set, while DSL maximizes the likelihood on the training data but selects the clauses based on the tuning set. This approach is similar to SAYU [2] that uses the tuning set to compute the score in terms of classification accuracy or AUC, with the difference that DSL uses CLL as score instead of AUC. Another difference with nFOIL and SAYU is that DSL, to perform inference for the computation of CLL, uses MC-SAT that is able to handle probabilistic, deterministic and near-deterministic dependencies that are typical of statistical relational learning. Moreover, the lazy version Lazy-MC-SAT reduces memory and time by orders of magnitude as the results in [28] show. This makes it possible to apply the proposed algorithm to very large domains.

8 Conclusions and Future Work

Markov Logic Networks are a powerful representation that combines first-order logic and probability by attaching weights to first-order formulas and viewing these as templates for features of Markov networks. In this paper we have introduced an algorithm that learns discriminatively first-order clauses and their weights. The algorithm scores the candidate structures by maximizing conditional likelihood while setting the parameters by maximum pseudo-likelihood. Empirical evaluation with real-world data in two domains show the promise of our approach improving over the state-of-the-art discriminative weight learning algorithm for MLNs in terms of conditional log-likelihood of the query predicates given evidence. We have also compared the proposed algorithm with the state-of-the-art generative structure learning algorithm and shown that on small datasets the generative approach is competitive, while on larger datasets the discriminative approach outperforms the generative one.

Directions for future work include speeding up the counting of the number of true groundings of a first-order clause which is the most expensive step for parameters' setting; learning the structure discriminatively and then applying a weight learning algorithm such as PSCG; developing methods to avoid overfitting; studying the relationship between the performance in terms of CLL and AUC and try to improve the accuracy in predicting the positives; scoring structures through AUC instead of CLL; adapting dynamically the nature of perturbation in the iterated local search procedure; performing more experiments with optimized parameters regarding the number of variables and literals per clause; finding heuristics to select, among clauses that do not improve WPLL, potential candidates that can improve CLL.

Acknowledgements. We would like to thank Pedro Domingos for helpful discussions, Daniel Lowd for help on PSCG, Marc Sumner for help on Alchemy and Lilyana Mihalkova per help on BUSL. We also thank the anonymous reviewers for their comments.

References

1. Besag, J.: Statistical analysis of non-lattice data. Statistician 24, 179–195 (1975)
2. Davis, J., Burnside, E., de Castro Dutra, I., Page, D., Santos Costa, V.: An integrated approach to learning Bayesian networks of rules. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 84–95. Springer, Heidelberg (2005)
3. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: Proc. of 23rd Intl. Conf. on Machine Learning, pp. 233–240 (2006)
4. Dehaspe, L.: Maximum entropy modeling with clausal constraints. In: Proc. of ILP 1997, pp. 109–124 (1997)
5. Della Pietra, S., Pietra, V.D., Laferty, J.: Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 380–392 (1997)
6. De Raedt, L., Dehaspe, L.: Clausal discovery. Machine Learning 26, 99–146 (1997)
7. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S.: Probabilistic Inductive Logic Programming - Theory and Applications. Springer, Heidelberg (2008)
8. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning 29, 103–130 (1997)
9. Friedman, J.H.: On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Data Mining and Knowledge Discovery 1, 55–77 (1997)
10. Getoor, L., Taskar, B.: Introduction to statistical relational learning. MIT Press, Cambridge (2007)
11. Greiner, R., Su, X., Shen, S., Zhou, W.: Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. Machine Learning 59, 297–322 (2005)
12. Grossman, D., Domingos, P.: Learning bayesian network classiers by maximizing conditional likelihood. In: Proc. 21st Int'l Conf. on Machine Learning, pp. 361–368. ACM Press, Banf (2004)
13. Hoos, H.H., Stutzle, T.: Stochastic local search: Foundations and applications. Morgan Kaufmann, San Francisco (2005)
14. Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: Proc, 22nd Int'l Conf. on Machine Learning, pp. 441–448 (2005)
15. Kok, S., Singla, P., Richardson, M., Domingos, P.: The alchemy system for statistical relational ai (Technical Report). Department of Computer Science and Engineering, University of Washington, Seattle, WA (2005), http://alchemy.cs.washington.edu/
16. Laferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. 18th Int'l Conf. on Machine Learning, pp. 282–289 (2001)
17. Landwehr, N., Kersting, K., De Raedt, L.: nFOIL: Integrating Naive Bayes and FOIL. In: Proc. 20th Nat'l Conf. on Artificial Intelligence, pp. 795–800. AAAI Press, Menlo Park (2005)
18. Landwehr, N., Kersting, K., De Raedt, L.: Integrating Naive Bayes and FOIL. Journal of Machine Learning Research, 481–507 (2007)

19. Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P.: kFOIL: Learning Simple Relational Kernels. In: Proc. 21st Nat'l Conf. on Artificial Intelligence. AAAI Press, Menlo Park (2006)
20. Loureno, H.R., Martin, O., Stutzle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 321–353. Kluwer Academic Publishers, Dordrecht (2002)
21. Lowd, D., Domingos, P.: Efficient weight learning for markov logic networks. In: Proc. of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 200–211 (2007)
22. Ng, A.Y., Jordan, M.I.: On discriminative vs. generative: A comparison of logistic regression and naive Bayes. In: Advances in Neural Information Processing Systems, vol. 14, pp. 841–848. MIT Press, Cambridge (2002)
23. McCallum, A.: Efficiently inducing features of conditional random fields. In: Proc. 19th Conf. on Uncertainty in Artificial Intelligence, pp. 403–410 (2003)
24. Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: Proc. 24th Int'l Conf. on Machine Learning, pp. 625–632 (2007)
25. Pernkopf, F., Bilmes, J.: Discriminative versus generative parameter and structure learning of Bayesian network classifiers. In: Proc, 22nd Int'l Conf. on Machine Learning, pp. 657–664 (2005)
26. Popescul, A., Ungar, L., Lawrence, S., Pennock, D.: Statistical Relational Learning for Document Mining. In: Proc. 3rd Int'l Conf. on Data Mining, pp. 275–282 (2003)
27. Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: Proc. 21st Nat'l Conf. on Artificial Intelligence, pp. 458–463. AAAI Press, Menlo Park (2006)
28. Poon, H., Domingos, P., Sumner, M.: A General Method for Reducing the Complexity of Relational Inference and its Application to MCMC. In: Proc. 23rd Nat'l Conf. on Artificial Intelligence. AAAI Press, Chicago (to appear, 2008)
29. Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)
30. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–236 (2006)
31. Singla, P., Domingos, P.: Discriminative training of markov logic networks. In: Proc. 20th Nat'l Conf. on Artificial Intelligence, pp. 868–873. AAAI Press, Menlo Park (2005)
32. Singla, P., Domingos, P.: Entity resolution with markov logic. In: Proc. 6th Int'l Conf. on Data Mining, pp. 572–582. IEEE Computer Society Press, Los Alamitos (2006)
33. Singla, P., Domingos, P.: Memory-efficient inference in relational domains. In: Proc. 21st Nat'l Conf. on Artificial Intelligence, pp. 488–493. AAAI Press, Menlo Park (2006)
34. Sha, F., Pereira, F.: Shallow parsing with conditional random fields. In: Proc. HLT-NAACL, pp. 134–141 (2003)

An Experiment in Robot Discovery with ILP

Gregor Leban, Jure Žabkar, and Ivan Bratko

Faculty of Computer and Information Science, University of Ljubljana,
Tržaška cesta 25, SI-1001 Ljubljana, Slovenia
{gregor.leban,jure.zabkar,ivan.bratko}@fri.uni-lj.si

Abstract. We describe an experiment in the application of ILP to autonomous discovery in a robotic domain. An autonomous robot is performing experiments in its world, collecting data and formulating predictive theories about this world. In particular, we are interested in the robot's "gaining insights" through predicate invention. In the first experimental scenario in a pushing blocks domain, the robot discovers the notion of objects' movability. The second scenario is about discovering the notion of obstacle. We describe experiments with a simulated robot, as well as an experiment with a real robot when robot's observations contain noise.

1 Introduction

In this paper we describe an application of an ILP system Hyper [3] to a simple robotic domain. The autonomous robot observes its environment which in our experiments consisted of two movable and two non-movable boxes (Fig. 1). It can perform experiments in this environment and collect data about its performed actions and the resulting observations. Our goal is to provide the robot a learning system that enables it to automatically discover new useful notions by exploring the domain. This is also known as *gaining insights* about the domain.

Although there has not been a unique generally accepted definition of the notion of insight, one definition that corresponds well to the present case study is as follows. An insight is a new piece of knowledge that enables a simplification of the robot's current theory. A variant of this definition is: Insight is the discovery of a new concept (e.g. in logic: the discovery of a new predicate) that enables the formulation of a new theory using the current theory as background knowledge and the observations as learning examples. This second definition precisely corresponds to the insight in the presented experiments.

We present two scenarios in which the robot learns the notions of *movability* and *obstacle*, respectively. In the *movability* scenario, where the robot's task is to push different objects, the robot discovers the notion of object's movability which helps it to explain the observations. In the *obstacle* scenario, the robot is moving around in the environment and discovers the notion of an obstacle, which helps to explain why in some cases it is not able to reach its desired goal position.

Our approach tackles two interesting topics in ILP, namely the predicate invention [12,13,10,2] and automatic generation of negative learning examples.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 77–90, 2008.

Predicate invention is the induction of useful subconcepts that makes the final theory shorter or learnable at all. The invented predicate is therefore usually an interesting concept by itself – it is whether so common that appears often in the final concept or it is so important that enables the target concept to be learned at all. Predicate invention is used for gaining insights by discovering new concepts. Automatic generation of negative learning examples is needed due to our use of ILP system Hyper which is not enable to learn from positive examples only [3].

For the purpose of our experiments, we extended Hyper by adding the capability of interpreting negated conditions in hypotheses as negation as failure. Also, a new sound heuristic was added for efficiency reasons: never refine a clause that is never used in the proof of any negative example. Any refinement of such a clause is provably redundant.

The paper is organized as follows. First we describe the related work. We then present how predicate invention can be done using ILP system Hyper and describe different experiments we performed when learning two notions, namely movability and obstacle. Next, we present a method for automatic generation of negative examples using the positive examples. We finish with concluding remarks on the factors, which in our opinion had the greatest impact on the time needed to learn the notions.

2 Related Work

Automated discovery systems such as BACON [11], LIVE [17] and DIDO [16] learn models by making experiments in their environment. Of these, only LIVE is an autonomous system while other require human interaction. Klingspor, Morik and Rieger have used ILP to tackle a problem similar to ours [7,9,14,15]. They report several difficulties using ILP. First, they have to convert numerical values to qualitative terms in order to use ILP at all. Another problem they face is a large size of training data. Their algorithm, GRDT, uses grammars to restrict the space of possible hypothesis in order to eliminate this problem. The greatest challenge for them seems to be the construction of negative examples. Trying to generate negative examples by the closed-world assumption they obtained extremely large data set which their algorithm could not handle. They solved the problem by implicitly applying the closed-world assumption through their algorithm. As described in [9], they work only with simulated data and don't report handling of noise. They learn models separately in several levels of the hierarchy of concepts. Rieger [15] is learning probabilistic automata in a similar domain but without the obstacles. In the past few years, there has been a significant interest in relational representations for modeling and learning in robotic domains. Relational reinforcement learning [6] and relational Markov decision processes [8] are basic approaches that are, in robotic domains, mainly applied in learning planning policies [20,4,5].

Boström [2] discusses two topics that are relevant to our problem, namely the predicate invention and learning from positive examples only. His system, MERLIN 2.0, uses a technique for inducing Hidden Markov Models from positive examples only [19]. Instead of minimizing the size of the automaton it maximizes the posterior probability. In our approach, HYPER actually uses positive as well as negative examples in the learning process and is in this respect conceptually much different. On the other hand, it constructs negative examples from positive ones automatically, so the input to the algorithm are positive examples only.

3 Predicate Invention Using Hyper

Learning definitions of relations in ILP can be, in machine learning terminology, regarded as supervised learning. It is supervised in the sense that the learning algorithm (ILP system) is given labeled examples (in ILP these correspond to the positive and negative examples) and the result of learning is a model (a hypothesis in the form of a predicate) that discriminates between positive and negative examples. In order to be able to learn a definition of the target predicate, we therefore have to be given some true and false examples of this predicate. For example, in order to learn the notion of movability, we would have to provide examples of movable and examples of unmovable objects.

The supervised way of learning is not suitable for the purpose of "gaining insights" since we have to specify in advance which notion we want to learn by providing positive and negative examples of the notion. Instead, we would like the robot to automatically discover new useful notions by inventing new predicates when they are needed. A notion can be considered as useful if it helps the robot to explain its past observations and enables it to predict the results of new actions.

In order to be able to learn new notions using Hyper, we represented the learning problem in the following way. We defined the target predicate (the predicate that we wish to learn) to be a predicate which specifies a command that was given to the robot and the resulting state after executing this command. For the movability notion, for example, the target predicate move(Obj, Start, Dist, End) contains the command to the robot to move the object Obj from its starting position Start by a specific distance Dist, and the object's position End after executing the command. The positive examples of such target predicate can easily be created by storing the commands and their results while the robot is exploring its environment. Automatic generation of negative examples using the positive examples is discussed in Section 5.

Based on the definition of the target predicate, Hyper will search for a hypothesis, possibly consisting of multiple clauses, that will explain the collected data. In order to enable Hyper to discover new notions in the form of new predicates, we add a new, auxiliary (yet unknown) predicate p to Hyper's starting hypothesis. This predicate will serve as a placeholder for the new notion. As an example, consider the definition of the starting hypothesis for the movable notion:

```
% A placeholder for the new predicate to be invented
start_clause([p(Obj)] / [Obj:object]).
start_clause([move(Obj, Start, Dist, End)]
        / [Obj:object, Start:position, Dist:position, End:position]).
start_clause([move(Obj, Start, Dist, End)]
        / [Obj:object, Start:position, Dist:position, End:position]).
```

There are two empty move clauses in the starting hypothesis since one is needed to describe examples where the robot pushes a movable object and another for describing examples where it pushes an unmovable object. We also have an empty clause for the predicate p. When generating possible hypotheses, Hyper can try to add to all three clauses the predicates from the background knowledge and the auxiliary predicate p. With these modifications, Hyper will be able to automatically discover a new notion, represented by p, which will be, if it is useful for the task, used in the target predicate move.

There is also a downside to the discovery of new notions using a placeholder predicate. As for all starting clauses in Hyper, we have to specify p's arguments and their types in advance. Although principles for choosing the arguments for the invented predicates exist [18], we manually specified the arguments and their types in our experiments. For the movability scenario, for example, we specified that the predicate p accepts one argument of type object.

4 Experiments

In this section we describe the experiments we performed when learning the notions of movable and obstacle. We also provide the description of the relevant background predicates and the form in which the data traces were collected. For each experiment we report the time needed to find the correct hypothesis and the number of the refined hypotheses during the search. The prolog implementation that we used was SWI-Prolog.

In all experiments we used the following setting. The robot was placed in a room large enough so that it never bumped into the walls during the experimentation. In the room there were four objects – objects a and d were movable by the robot, while the objects b and c were not. The starting position of the robot and the objects is depicted in Figure 1.

To learn the notions we first used data collected using an XPERSim robot simulator [1] and then repeated the learning using noisy data from a real robot. All results presented here were obtained using the real robot data. The data that was collected while executing the commands (robot's sensory information and the positions of the objects) was stored in the form of predicates at, contact, and moving, which will be described in the next section. The amount of noise that was present in the data was about 10% of the variable span.

4.1 Learning the Notion of Object Movability

One of the simplest notions that the robot might find useful in its interaction with the world is the notion of object movability. In order for the robot to be

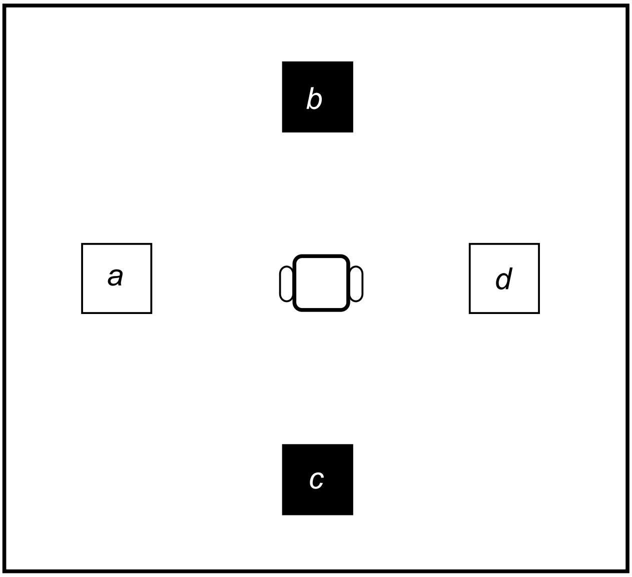


Fig. 1. The robot in its environment consisting of two movable (objects *a* and *d*) and two non-movable boxes (objects *b* and *c*)

able to discover such a notion, we set up the following experiment. We placed a robot inside room with two movable and two unmovable objects. The robot was then to execute several commands of the form "move the object Obj for the distance Dist" (where Dist is a vector with x and y values). We expected that the robot will in some cases be able to successfully move the object for the desired distance, and in some cases the position of the object will not change at all. Since it is the object's movability, which determines whether its position will change upon pushing or not, this would be a valuable concept to the robot that would allow it to predict the results of its actions.

In order to learn the concept of movability we saved several robot's commands and their results in the form of a target predicate move(Obj, Start, Dist, End), which states that the robot was supposed to move the object Obj from its Start position by distance Dist and the command finished with the object being at position End. The position and distance variables are vectors with real numbers. We tried to discover this notion by providing the data traces in two ways - using only allocentric (global) coordinates, and using allocentric coordinates together with the robot's sensory measurements. Using the robot's sensory measurements, we also performed an experiment with two robots, where one robot is stronger and is able to move one object, which appears to be unmovable to the weaker robot. We will now describe all three experiments in detail.

Notion of movability using global coordinates. In this experiment we used an over-head camera which was able to track the position of each of the objects over time. The state of the world at different time points was therefore completely described using facts in form of a predicate at(Ob, Time, Pos) which states that the object Obj was at time Time at position Pos. The data trace that we used contained about 450 at facts. The other background predicates that we allowed Hyper to use were:

- approxEqual(Pos1, Pos2) – the predicate succeeds if position Pos1 is approximately the same as position Pos2. The predicate is needed to take into account the noise which is present in the data.
- add(Pos1, Dist, Pos2) – the predicate can be used in two ways. When Pos2 is uninstantiated (a variable), the predicate computes the sum of the position Pos1 (which is a list with x and y coordinates) and distance Dist (also in a form of a list with two values) and returns the result as the variable Pos2. When Pos2 is instantiated then the predicate succeeds if Pos1 + Dist approximately equals Pos2.
- different(Pos1, Pos2) – the predicate succeeds if the position Pos1 is not approximately equal to position Pos2. This predicate is basically the negation of the predicate approxEqual.

As already mentioned, the position and distance arguments, that are present in the target and background predicates are vectors represented as lists containing two values. It is important to note that when specifying types of predicate arguments, these arguments have to be of a special type (in our case we named them position) and not as a list of two values of type number. For example, the declaration of the background predicate at had to be:

```
backliteral( at(Obj, T, P), [ Obj:object], [ T:time, P:position]).
```

Declaring the predicate at (and similarly other predicates) like:

```
backliteral( at(Obj, T, [X,Y]), [ Obj:object], [T:time, X:number, Y:number]).
```

would force Hyper to introduce a larger number of new variables. As a result, when searching for a correct hypothesis, Hyper would have to try to unify a larger number of arguments which would cause a significant (and unnecessary) increase of computational complexity.

Using the described target and background predicates Hyper induced the following theory of moving in this world:

```
p(Obj) :–
      at(Obj, T1, Pos1),
      at(Obj, T2, Pos2),
      different(Pos1, Pos2).
move(Obj, Start, Dist, End) :–
      approxEqual(Start, End),
      not p(Obj).
move(Obj, Start, Dist, End) :–
      add(Start, Dist, End),
      p(Obj).
```

We can see here that Hyper used the p predicate as a placeholder for inventing the predicate movable. It defined object as movable if there exist two time points at which the position of the object is different. All move commands that were collected can then be explained in two ways:

1. the starting and ending positions of the object are approximately the same and the object is unmovable, or
2. the object is movable and the starting position plus distance equals to the end position of the object.

To find the solution Hyper needed 58 minutes on a Pentium PC 2.66 GHz computer and in the process refined 25,241 hypotheses.

Notion of movability using global coordinates and sensory information. In this experiment we used global coordinates only for storing the information on object's coordinates in the positive examples. The data that we collected while the robot was executing commands came from its sensors and we stored it in the form of the following predicates:

- contact(Obj, Time, Val) – value Val can be 0 or 1 and describes whether the robot was in contact with an object Obj at time Time.
- moving(Object, Time, Val) – value Val can be 0 or 1 and describes whether the object Object was being moved at time Time.
- moving(Time, Val) – value Val can be 0 or 1 and describes whether the robot was moving forward at time Time.

The additional background predicates were also the add and approxEqual predicates that were already described in the previous experiment. The solution which Hyper found in a few seconds was:

```
p(Obj) :–
        moving(Obj, T, 1).
move(Obj, Start, Dist, End) :–
        approxEqual(Start, End),
        not p(Obj).
move(Obj, Start, Dist, End) :–
        add(Start, Dist, End),
        p(Obj).
```

We can see that Hyper found a short definition of movable without using the contact predicate. The reason why the predicate moving/3 suffices is that in this experiment the only time an object would be moving is when the robot would be pushing it (whenever the last argument of predicate moving is 1, the last argument of contact is also 1) which makes the contact predicate redundant. Since we also wanted to find a more specific definition of movability, which would state that the object is movable if it can be moved by the robot, we created another experiment with two robots.

Notion of movability using two robots. In this experiment we used two robots, one slightly stronger than the other. There were still four objects (*a*, *b*, *c*, and *d*). The robot *r*1 was able to move the object *d*, while the stronger robot *r*2 was also able to move the object *a*. Because of two robots, the move commands that were issued in this experiment contained as the first argument the name of the robot that was ordered to execute the command. Both robots were issued several move commands. The moving and contact predicates also had to be modified and now contained the robot's name as the first argument. The solution that Hyper found in a few seconds was:

```
p(R, Obj) :–
      moving(R, T, 1),
      contact(R, Obj, T, 1).
move(R, Obj, Start, Dist, End) :–
      approxEqual(Start, End),
      not p(R, Obj).
move(R, Obj, Start, Dist, End) :–
      add(Start, Dist, End),
      p(R, Obj).
```

From this solution we see that the moving predicate is no longer sufficient to determine the movability of the object. By this definition, the object is movable by a robot if, while in contact with the object, the robot can still move forward.

4.2 Learning the Notion of an Obstacle

The second notion that we were very interested to learn was the notion of an obstacle. An obstacle is an object, which impedes the robot's progress on its desired path and prevents it from reaching the goal position. Learning this notion is very important for the robot since it enables it to predict in advance whether a specific goal can be reached successfully or not and thus allowing it to create an alternative plan.

In order to learn this concept we performed the following experiment. We modified the commands to the robot from moving an object into simply moving the robot from current position to some other position. The target predicate move now had the form move(Start, Dist, End), which means that the robot was ordered to move from its Start position by distance Dist and was at the end at position End. When there is no obstacle on the way, the End position is simply the sum of Start and Dist vectors. On the other hand, when the robot bumps on the way into an unmovable object, it is unable to move further and its End position is the place of impact with the object. The position and distance variables are again vectors with real numbers.

In this experiment we again used two movable and two unmovable objects. The data that we collected was in the form of at facts, describing the position of all four objects at different time points. Beside the approxEqual and add predicates that were already described and used in the movability scenario, we allowed our ILP system to use the following background predicates:

- interpolate(Pos, Dist, Points) – returns a list containing a sample of points that lie on the straight line from Pos to Pos+Dist.
- occupied(Points, Obj, Pos) – the predicate accepts as input the list of points Points and succeeds if any of these points is occupied by an object. If it is, then it returns the information about the object and its location. Since each object is represented as a two-dimensional point depicting the center of the object, we also consider a point to be occupied if it lies in the near vicinity of the object's center.
- movable(Obj) – this predicate was learned in the previous experiments and is now used as a background predicate. It succeeds if the object Obj is movable, and fails otherwise.

The fact that we allow the movable predicate to be used as a background predicate when learning the obstacle notion demonstrates the idea of evolving theories in the XPERO project – the theories that are learned are used in the following experiments where they can help in explaining new observations and creating new theories. In this process of learning it might even become apparent that the old theories are not completely correct and need to be updated or refined.

Using the modified target predicate move and the described background predicates, Hyper was not able to find a solution in one week of computer time. We noticed that the evaluation of individual hypotheses was very slow and often took several seconds. One of the reasons for the slowness proved to be the combination of interpolate and occupied predicates which take a long time to evaluate. To speed up the search we introduced the following two modifications:

1. We changed the type of the Dist variable in move, add, and interpolate predicates from type position to type distance. Although the Dist variable is of the same form (a list with two numeric values) as, for example, the Start and End variables, its meaning is nevertheless different. Introducing this change significantly reduces the size of the search space, since there is now a much smaller number of possible variable unifications.
2. We implemented a predicate, called reduceData, which was called after the data trace in the form of at facts was loaded. The purpose of this predicate was to remove at facts at those time points where the position of an object did not change when compared to the position at the previous time point. The reason why these facts can and should be removed is that they do not add any new information while they do significantly increase the time needed to evaluate a hypothesis. After calling this predicate, out of original 600 only 20 at facts remained.

Using these modifications, Hyper found the following solution:

```
p(Start, Dist, Obj) :–
        interpolate(Start, Dist, Positions),
        occupied(Positions, Obj, Pos),
        not movable(Obj).
move(Start, Dist, End) :–
        add(Start, Dist, End),
        not p(Start, Dist, Obj).
move(Start, Dist, End) :–
        p(Start, Dist, Obj),
        at(Obj, Time, Pos),
        approxEqual(Pos, End).
```

To find this hypothesis, Hyper refined 197,378 hypotheses and needed 207 minutes. Since the combination of predicates interpolate and occupied proved to be very slow we wanted to try an alternative which would work faster. We therefore replaced those background predicates with a predicate liesBetween(Start, Dist, Obj). This predicate succeeds if there is an object that lies on a straight line between Start and Start+Dist and in this case returns the object's name in the Object variable. Using this predicate Hyper was able to find the following solution:

```
p(Start, Dist, Obj) :–
        liesBetween(Start, Dist, Obj),
        not movable(Obj).
move(Start, Dist, End) :–
        add(Start, Dist, End),
        not p(Start, Dist, Obj).
move(Start, Dist, End) :–
        p(Start, Dist, Obj),
        at(Obj, Time, Pos),
        approxEqual(Pos, End).
```

To find this definition Hyper needed 28 minutes and in the process refined 71,224 hypotheses.

5 Generating Negative Examples

In order for Hyper to learn the target predicate, we have to specify positive and negative examples. In our experiments, the positive examples were simply provided by the robot, since they consist of the command given to the robot and of the result after its execution. The negative examples, on the other hand, cannot be collected by the robot since they consist of cases, which cannot occur in the real world.

One way of creating negative examples is by using a form of the closed-world assumption. According to the closed-world assumption, everything that is currently not known to be true is considered to be false. A particular form of the closed-world assumption is that the target predicate specifies a functional relation. That is, the last argument of the predicate is a function of the other arguments. Using this assumption we can create a negative example by simply taking a positive example and changing the result of the command to a different (possibly random) value.

The described approach, although it can be used to automatically generate any number of negative examples, is inadequate. The definition of the target predicate that we wish to discover is one that is specific enough that it will enable the robot to predict the results of its actions. However, if we generate the negative examples as described, we are most likely to get an overly general hypothesis that will, when used, predict multiple possible outcomes for a given command. Consider, for example, the notion of movability. If we generate one negative example for each positive example by randomly changing the last argument of the example, Hyper finds the following theory:

```
p(Obj).
move(Obj, Start, Dist, End) :–
      add(Start, Dist, End).
move(Obj, Start, Dist, End) :–
      approxEqual(Start, End).
```

As we can see, the theory is very general but it is nevertheless consistent with the given data. The reason why this overly general hypothesis is sufficient is that we have not provided some “critical” negative examples. An example of such a negative example would be one where the robot was ordered to move a movable object, but the End position is equal to the Start position (this should clearly happen only when trying to move an unmovable object). Similarly, a critical negative example for trying to move an unmovable object would be one where the End position is the sum of the Start and Dist variables. Of course, if it would be feasible to create a huge number of negative examples for each positive example then we would very likely also create the critical negative examples. However, to keep hypothesis evaluation reasonably fast, we should have as few negative examples as possible.

How can we then “guess” what are the values of the critical negative examples? One might try to generate the value of the End variable based on the other two variables in the command (Start and Dist). We can generate negative examples where the End variable equals the first or the second variable, or a value computed using any arithmetic operator on them, as long as the value is different from the End value of the positive example. In this way, we can successfully generate critical examples for the movability notion. This approach, unfortunately, does not work in general. It does not work even for the obstacle notion, where a critical example is one where the End position is similar to the position of an unmovable object on the robot’s path.

The approach that we used for generating critical negative examples in our experiments and which we believe can generally be used when learning new notions, is as following. First, we start by creating a small number of negative examples by taking positive examples and changing the result of the command to some random value. Afterwards we run Hyper, which will find the simple, overly general theory shown above. Using this theory, the robot will now be able to compute the expected results of its actions. We can test this theory on the already collected positive examples. One thing that we will notice when predicting the results of robot's actions is that for each positive example, the theory predicts two possible End positions for the object: one End position equals to Start and the other to Start+Dist. Since the positive example also contains the information about the actual End position, we can construct a new negative example using the robot's command and the incorrect prediction. We can run Hyper again using this extended set of negative examples and find a new hypothesis, which will also be consistent with the added negative examples. To find the correct theory, we will therefore iteratively modify the current theory until it will give a single prediction for each robot's command. In this way, we can use intermediate hypotheses as generators of critical negative examples and therefore add in each iteration only those negative examples that will force the theory to be refined.

Using the presented approach we successfully generated negative examples in all our experiments. For both notions, the movability as well as the obstacle notion, only two iterations were needed to find the final theory. Although we presented the procedure as if it needs human intervention, it can be run completely automatically.

6 Conclusion

In the paper we presented our experiments with predicate invention using ILP system Hyper in a simple robotic domain. We showed examples where Hyper was able to successfully invent several auxiliary predicates that represent the notion of the object's movability and the notion of an obstacle. The data that was used for learning was collected using a real robot and therefore contained a significant amount of noise.

Based on the performed experiments, we can draw some conclusions on the factors that have the biggest impact on our ability to learn a desired notion:

1. **The number of background predicates.** Adding unnecessary background predicates increases the number of possible hypotheses that ILP system has to test and therefore makes the search for the right theory slower.
2. **The number and type of variables in the predicates.** This factor has a huge impact on the speed of learning. If the predicates that now contain the variables of type position would be redefined so that each value of the list would be its own variable, the theory would not be found in a reasonable time. It is also very important to make the types of variables different whenever possible. This is evident from our obstacle scenario, where we had to change the type of the Dist variable from position to distance.
3. **Maximum clause length.** When Hyper searches for a theory it generates hypotheses where the length of each clause is limited to some predefined

maximum length. Increasing maximum clause length increases the time complexity, since it allows Hyper to use more literals and therefore increases the search space. Hypotheses with longer clauses also take longer to evaluate since variables in each literal can often be assigned several different values.

4. **Slow predicates.** It is likely that Hyper will have to test a large number of hypotheses before it will find the right one. It is, therefore, important that the validity of each hypothesis can be evaluated as quickly as possible. For that to happen, we have to provide as efficient implementation of background predicates as possible.
5. **The length of the data traces.** In the experiments above we demonstrated how in some cases increasing the length of the traces significantly increases the time needed to find the solution. We can use different techniques to reduce the length of the traces. One possibility is to use data sampling, where we use only every n-th case from the trace. A solution that we used when discovering the obstacle notion is to remove cases with the same values. Such cases are redundant and can usually be removed without losing any relevant information.

Acknowledgment

The work described in this article has been funded by the European Commission's Sixth Framework Programme under contract no. 029427 as part of the Specific Targeted Research Project XPERO ("Robotic Learning by Experimentation").

References

1. Awaad, I.S., Leon, B.E.: Xpersim: Simulation of the robotic experimenter. Technical report, University of Applied Sciences Bonn-Rhein-Sieg (2006)
2. Boström, H.: Predicate invention and learning from positive examples only. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 226–237. Springer, Heidelberg (1998)
3. Bratko, I.: Prolog Programming for Artificial Intelligence. Addison-Wesley, Reading (2001)
4. Cocora, A., Kersting, K., Plagemann, C., Burgard, W., De Raedt, L.: Learning relational navigation policies. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2006)
5. Deshpande, A., Milch, B., Zettlemoyer, L.S., Kaelbling, L.P.: Learning probabilistic relational dynamics for multiple tasks. In: Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence (UAI) (2007)
6. Dzeroski, S., De Raedt, L., Blockeel, H.: Relational reinforcement learning. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 11–22. Springer, Heidelberg (1998)
7. Kaiser, M., Klingspor, V., Morik, K., Rieger, A., Acame, M., del, J., Millan, R.: Learning techniques for mobile systems (1995)
8. Kersting, K., Van Otterlo, M., De Raedt, L.: Bellman goes relational. In: Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004), Banff, Alberta, Canada (2004)

9. Klingspor, V., Morik, K., Rieger, A.D.: Learning concepts from sensor data of a mobile robot. Machine Learning 23(2-3), 305–332 (1996)
10. Kramer, S.: Predicate invention: A comprehensive view. Technical Report OFAI-TR-95-32 (1995)
11. Langley, P., Simon, H., Bradshaw, G., Zytkow, J.: Scientific discovery: Computational explorations of the creative processes (1987)
12. Muggleton, S.: A strategy for constructing new predicates in first order logic. In: Sleeman, D. (ed.) Proceedings of the 3rd European Working Session on Learning, pp. 123–130. Pitman (1988)
13. Muggleton, S.: Predicate invention and utility. Journal of Experimental and Theoretical Artificial Intelligence 6(1), 121–130 (1994)
14. Rieger, A.: Data preparation for inductive learning in robotics (1995)
15. Rieger, A.: Learning to guide a robot via perceptrons. In: Ghallab, M., Milani, A. (eds.) New Directions in AI Planning, pp. 383–394. IOS Press, Amsterdam (1996)
16. Scott, P.D., Markovitch, S.: Learning novel domains through curiosity and conjecture. In: IJCAI, pp. 669–674 (1989)
17. Shen, W.-M.: Discovery as autonomous learning from the environment. Machine Learning 12, 143–165 (1993)
18. Stahl, I., Weber, I.: The arguments of newly invented predicates in ILP. In: Wrobel, S. (ed.) Proceedings of the 4th International Workshop on Inductive Logic Programming. Gesellschaft für Mathematik und Datenverarbeitung MBH, vol. 237, pp. 233–246 (1994)
19. Stolcke, A., Omohundro, S.M.: Best-first model merging for hidden Markov model induction. Technical Report TR-94-003, 1947 Center Street, Berkeley, CA (1994)
20. Zettlemoyer, L.S., Pasula, H.M., Kaelbling, L.P.: Learning planning rules in noisy stochastic worlds. In: Proc. 20th National Conference on Artificial Intelligence (2005)

Using the Bottom Clause and Mode Declarations on FOL Theory Revision from Examples

Ana Luísa Duboc, Aline Paes, and Gerson Zaverucha

Department of Systems Engineering and Computer Science - COPPE
Federal University of Rio de Janeiro (UFRJ), Brazil
{aduboc,ampaes,gerson}@cos.ufrj.br

Abstract. Theory revision systems are designed to improve the accuracy of an initial theory, producing more accurate and comprehensible theories than purely inductive methods. Such systems search for points where examples are misclassified and modify them using revision operators. This includes trying to add antecedents in clauses usually generated in a top-down approach, considering all the literals of the knowledge base. This leads to a huge search space which dominates the cost of the revision process. ILP Mode Directed Inverse Entailment systems restrict the search for antecedents to the literals of the bottom clause. In this work the bottom clause and modes declarations are introduced to improve the efficiency of theory revision antecedent addition. Experimental results compared to FORTE revision system show that the runtime of the revision process is on average three orders of magnitude faster, and generate more comprehensible theories without decreasing the accuracy. Moreover, the proposed theory revision approach significantly improves predictive accuracy over theories generated by Aleph system.

1 Introduction

Inductive Logic Programming algorithms learn First-order Logic theories from a set of negative and positive examples and a fixed background knowledge(BK) [Muggleton, 1992]. The BK is composed of a set of clauses assumed as correct and therefore they can not be modified. Usually, the final hypothesis is found through a covering approach, where at each iteration a single clause is learned and the positive examples covered by such clause are removed from the list of examples. Although the covering algorithm is fast, it can generate many clauses unnecessarily long and only locally optimal [Bratko, 1999]. Besides that, the covering approach generally learns single predicates. On the other hand, FOL revision systems not only can also assume a part of the BK as correct and such that will not be modified but they are also capable of modifying a part of the initial knowledge which is approximately correct [Wrobel, 1996]. The initial knowledge which can be modified is called as *initial theory.* In order to improve the initial theory, the points which are failing in correctly classifying some example are identified and modifications are proposed to these points. In this way, FOL revision systems do not use a covering approach, it considers the whole

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 91–106, 2008.

theory instead of individual clauses when proposing these modifications. Thus, more comprehensible final hypothesis are generated, with better global clauses, and it is also possible to learn multiple predicates simultaneously.

Usually, theory revision systems such as FORTE [Richards and Mooney, 1995] include addition of antecedents in a clause as one of the possible modifications in a theory. In that system, the search for antecedents follows FOIL [Quinlan, 1990] top-down approach which eventually generates a huge search space of literals with some of them do not covering even one positive example. On the other hand, ILP algorithms such as Progol [Muggleton, 1995] and Aleph [Srinivasan, 2001], restrict the search for literals to those ones belonging to the bottom clause. The bottom clause is the set of relevant literals to a positive example, collected from a mode directed search in the BK. This hybrid bottom-up and top-down approach frequently generates many fewer literals, and they are also guaranteed to cover at least one positive example (that one used to generate the bottom clause). Thus, in this work, we propose to use the literals of the bottom clause as search space for antecedents to add in a clause. Additionally, we propose the use of mode declarations to validate if the antecedents of the bottom clause can effectively be added to the current clause.

The rest of this paper is organized as follows. Section 2 reviews some background knowledge concerning theory revision. Then, the proposed approach of this work is presented in section 3, followed by the experimental results in section 4. Finally, the work is concluded in section 5.

2 First-Order Logic Theories Revision

Figure 1 presents a schema for theory revision. The theory revision system receives a *theory* Γ and a set of examples $E^+ \cup E^-$, divided in a set of positive and negative examples, respectively. The theory Γ includes two components: an invariant component, named *background knowledge (BK)*, and one component that can be modified (H), named *initial theory*.

Ideally, the revision process should generate a final theory that it will prove all the positive examples in E^+ and none of the negative examples, E^-, that is, the final theory will be consistent with the dataset. Notice that learning in Inductive Logic Programming (ILP) can be seen as a particular case of theory revision where H is initially empty.

Revision Points. Many clauses can be involved in the proof of a negative example or in no proof of a positive example, as many clauses can be correct and do not need to be modified. Therefore, it is necessary to find the theories points that need to be corrected, called revision points. There are two types of revision points:

- *Generalization* - the literal in a clause responsible for the failure of proving positive examples (failure point) and other antecedents (contributing points) that may have contributed to this failure;
- *Specialization* - clauses used in successful proofs of negative examples.

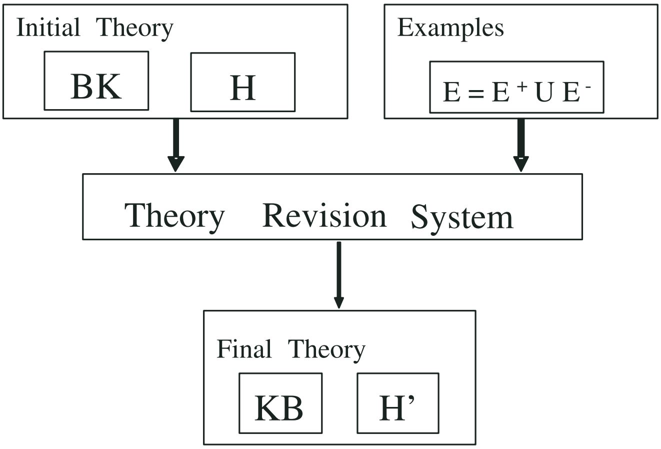


Fig. 1. Revision Theory Schema

The specification of the revision point determines the type of revision operator that will be applied to make the theory consistent with the dataset. One may consider two types of operators. One may add previously missing answers through *generalization* or by removing incorrect answers through *specialization* [Wrobel, 1996, Richards and Mooney, 1995].

Revision Operators. Theory revision relies on operators that propose modifications at each *revision point*. Any operator used in first-order machine learning can be used in a theory revision system. Below, we briefly describe some operators used in this work.

The operators for specialization are:

- *Delete-rule* - this commonly used operator removes a clause that was used in the prove of a negative example.
- *Add-antecedent* - this operator adds antecedents to an incorrect clause.

The generalization operators are:

- *Delete-antecedent* - this operator removes failed antecedents from clauses that could be used to prove positive examples.
- *Rule-Addition* - this operator generates new clauses from existing ones using deletion of antecedents followed by addition of antecedents. It is also possible to create an entirely new clause.

There are other approaches of generalization and specialization operators. For more details on revision operators we refer the reader to [Richards and Mooney, 1995]and [Wrobel, 1996].

FORTE. In this work we follow the FORTE revision system (First Order Revision of Theories from examples) [Richards and Mooney, 1995]. FORTE performs a hill-climbing search through a space of both specialization and generalization operators in an attempt to find a minimal revision to a theory that makes it consistent with the set of training examples. FORTE is a batch revision system in the sense that the examples are all processed at once. The top-level algorithm is exhibited as Algorithm 1.1. The key ideas are:

1. Identify all the revision points in the current theory.
2. Generate a set of proposed revisions for each revision point starting from that one with the highest potential and working down the list. *Potential* is defined as the number of misclassified examples that could be turned into correctly classified from a revision in that point.
3. Score each revision through the actual increase in theory accuracy it achieves.
4. Retain the revision which most increases the score.

FORTE stops when the potential of next revision point is less than the score of the best revision to date. If the best revision really improves the theory it is implemented. Conceptually, each operator develops its revision using the entire training set. However, in practice, this is usually unnecessary and thus FORTE considers only the examples whose provability can be affected after proposing some revision.

Algorithm 1.1. FORTE Algorithm (Richards and Mooney, 1995)

```
repeat
    1. generate revision points;
    2. sort revision points by potential (high to low);
    3. for each revision point
    4.     generate revisions;
    5.     update best revision found;
    6. until potential of next revision point is less than the score of the best revision
    to date
    7. if best revision improves the theory
    8.     implement best revision
until no revision improves the theory;
```

Antecedents addition. Now we further discuss the addition antecedents operation of FORTE since our purpose is making it more efficient. There are two algorithms for adding antecedents in a clause:

1. Hill climbing - This algorithm is based on FOIL and add one antecedent at time. It works as follows. First, all possible antecedents are created and scored using FOIL score. Then, the antecedent with the best score is selected. If its score is better than the current clause score, it is added to the clause. This process continues until either there are no more antecedents to be added in the clause or if no antecedent can improve the current score. This approach is susceptible to local maxima.

2. Relational Pathfinding - In this approach a sequence of antecedents is added to a clause at once in attempt to get out of local maxima, as, sometimes, none of the antecedents put individually in the clause improves its performance.

   The relational pathfinding algorithm is based on the assumption that generally in relational domains there is a path with a fixed set of relations connecting a set of terms and such path satisfies the target concept. A relational domain can be represented as a graph where the nodes are the terms and the edges are the relations among them. Thus, we can define a *relational path* as the set of edges (relations) which connect nodes (terms) of the graph. To better visualize such approach, consider, for instance, the graph in Figure 2, which represents a part of the family domain. In this graph, horizontal lines denote marriage relationships, and the remaining lines denote parental relationships:

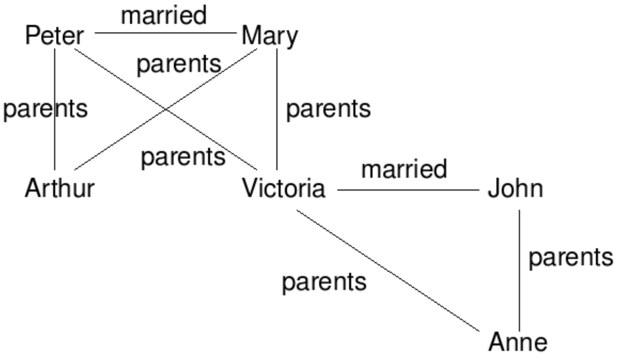


Fig. 2. An example of a relational graph representing part of the family domain

Now, suppose we want to learn the target concept *grandfather*, given an empty initial rule and a positive example *grandfather(peter,anne)*. The relational path between the terms *peter* and *anne* is composed of the relation *parents* connecting *peter* to *victoria*, and also of the relation *parents* connecting *victoria* to *anne*. From that relations, is formed the path *parents(peter,victoria),parents(victoria,anne)*, which can be used to define the target concept: $grandfather(A, B) : -parents(A, C), parents(C, B)$.

Therefore, the *relational pathfinding* algorithm aims to find such relational paths given a relational domain, since important concepts are represented by a small set of fixed paths between terms defining a positive example. From the point of view of theory revision, this algorithm can be used whenever a clause needs to be specialized and it does not have relational paths connecting its variables. In this case, a positive example proved by the clause is chosen to instantiate it, and then, from the ground clause, relational paths to the terms without a relationship in the clause are searched.

If the new relations found have introduced new terms that appear only once, FORTE tries to complete the clause by adding relations that hold between these singletons and other terms in the clause; these new relations are not allowed to eliminate any of the currently provable positive instances. If FORTE is unable to use all of the new singletons, the relational path is rejected. In this work, differently from FORTE, we do not reject a path if it has a singleton from a predicate of arity > 3, provided that this predicate has completed the path among the terms of the original clause.

When specializing a clause using any of these algorithms, if the antecedents added to the clause make some positive example become not proved, the just revised clause is added to the set of modifications proposed to the theory and a new search for antecedents starts from the original clause, in an attempt to recover the provability of the positive examples. This process continues until all the positive examples originally covered by the initial clause continue to be proved.

Antecedents Generation. When creating literals to add in a clause, all the predicates defined on the knowledge base are considered. A literal is created from a predicate by replacing its arguments for variables. These variables may be the ones that already exist on the clause being modified, or can be new variables in a limit of $n - 1$ new ones, where n is the arity of the predicate. When creating these literals the algorithms above consider only the constraints below, following FOIL:

1. At least one variable of the new literal must be in the clause being revised;
2. The arguments of the literals must obey the types defined in the knowledge base.

Such constraints does not explore properly the connections among variables, since they are not defined as input or output variables (there are no *modes declarations*). Besides, several different variations of the same literal are generated from the existing variables and the new ones. Clearly, the larger is the number of new variables on the clause, the more possible antecedents are created. Actually, the space complexity grows exponentially on the number of new variables since the complexity of enumerating all possible combinations of variables is exponential according to the arity of the predicate. Thus, following such approach, the antecedents addition operation is completely inefficient, since all the literals created must be scored in order to choose the best one. As it is known, usual scoring functions in ILP are very expensive since they involve attempts of proving all examples.

The process of generating literals is slightly different for the two antecedents generation algorithms. The algorithms for these process can be seen in 1.2 and 1.3, for hill climbing and relational pathfinding algorithms, respectively.

As we said before, the relational pathfinding algorithm starts from a clause grounded from a positive example covered by the clause. The terms in the ground clause will be the nodes in the graph, connected by the relations defined in clause's body clause. The algorithm constructs the graph iteratively, starting from these initial nodes and expanding them until finding the relational paths. The *end values* are the terms (nodes) created when a node is expanded.

Algorithm 1.2. Hill Climbing Antecedent generation

```
$C$ = Clause to be specialized
for each literal in the knowledge base do
    return the variables and their types from $C$
    return the arguments and their types from the particular literal
    for each combination of variables from $C$ do
        Verify if the combination is valid as possible arguments of the literal
        if the combination is valid then
            Replace the arguments of the literal for the combination of variables found
            return the literal as a new antecedent
        else
            Go to the next combination of variables available
    $N$ = number of arguments of the particular literal
    $i = 1$
    while $i \leq N$ - 1 do
        Create a new variable $L$
        $V$ = Variables of $C + L$
        for each combination in $V$ that includes $L$ and some variable of $C$ do
            Verify if the combination is valid as possible arguments of the literal
            if the combination is valid then
                Replace the arguments of the literal for the combination of variables found
                return the literal as a new antecedent
            else
                Go to the next combination of variables available
        $i = i + 1$
    Go to the next literal in the knowledge base
```

3 Using the Bottom Clause to Search for Antecedents When Revising a FOL Theory

Since only the constraints above are considered when generating literals and the search space for antecedents is composed of the whole knowledge base, the complexity of the addition antecedents operation in a clause becomes very high and dominates the cost of the revision process. Aiming to reduce such cost, in this work we propose the following modifications to FORTE system:

1. to use the variabilized bottom clause as search space of literals, which reduces the search space and also impose the following constraints:
   - to limit the maximum number of different instantiations of a literal (the recall number);
   - to limit the number of new variables in a clause;
   - to guarantee that at least one positive example is covered (the one which generates the bottom clause).
2. to declare modes to the arguments of a literal. Thus, now the arguments are defined as input, output and constant.

Algorithm 1.3. Relational Pathfinding Antecedent generation

```
for each literal in the knowledge base do
  return the end values and their types in the node been expanded
  return the arguments and their types from the particular literal
  for each combination of end values in the node do
    Verify if the combination is valid as possible arguments of the literal
    if the combination is valid then
      Replace the arguments of the literal for the combination of end values found
      return the literal as a new antecedent
    else
      Go to the next combination of end values
  $N$ = number of arguments of the particular literal
  $i = 1$
  while $i \leq N$ - 1 do
    Create a new variable $L$
    $V$ = end values of the node + $L$
    for each combination in $V$ that includes $L$ and some end value of the the node
    do
      Verify if the combination is valid as possible arguments of the literal
      if the combination is valid then
        Replace the arguments of the literal for the combination of end values
        found
        Search in the background knowledge for a fact that unifies with the literal
        created
        if a fact is found then
          return the literal as a new antecedent
        else
          Go to the next combination of variables available
      else
        Go to the next combination of variables available
    $i = i + 1$
  Go to the next literal in the knowledge base
```

When using modes declarations and the bottom clause in FOL theory revision, the search space is reduced and consequently the runtime of the revision process is speeded up. Additionally, more comprehensible theories are generated since we are focusing on the literals which are really relevant.

Using the bottom clause as search space for antecedents. In order to generate the bottom clause in FORTE system, we use the saturation phase, defined by Progol/Aleph. The Bottom Clause is created immediately before the search for antecedents begins, from a positive example covered by the clause being modified(base clause). Note that as we are revising an existing theory, the body of the base clause probably is not empty. We choose a covered positive example because when specializing a clause the goal is to make the negative examples covered by the clause become unprovable while the originally provable positive examples are still covered. Thus, naturally, the bottom clause must be composed of the relevant literals for at least one positive example covered by the clause. Also, in this way, the base clause is a subset of the bottom clause. The created bottom clause becomes the search space for antecedents, which improves the efficiency of the addition antecedents operation since it has many fewer literals than the previously space of the whole knowledge base. Additionally, we have the guarantee that at least one positive example continues covered by the generated literals, which was not guaranteed before by the top-down search for antecedents. Also, as the bottom clause is already variabilized it is not necessary to generate neither new literals nor new variables. Previously, such operation had an exponential cost according the arity of predicates. It is important to emphasize that the constraints of FOIL continues to be met here, as the arguments of the literals in the bottom clause must obey their types and there is a linking variable between the literal being added in the clause and the literals of the current clause.

The new antecedent generation algorithm using bottom clause for hill climbing and relational pathfinding can be seen in 1.4 and 1.5, respectively.

Algorithm 1.4. Hill climbing antecedent generation using bottom clause

```
$C$ = Clause to be specialized
for each literal in the bottom clause do
   return the variables of $C$
   return the variables of the particular literal
   if the literal has a variable in common with $C$ then
      return the literal as a new antecedent
   else
      Go to the next literal in the bottom clause
```

Using the modes declaration to validate antecedents. The modes declaration in Mode Directed Inverse Entailment (MDIE) search describes the relations between the arguments and their types and also defines if a predicate can be used in the head of the clause (*modeh*) or in the body of the clause (*modeb*). They can also constrain the number of different instantiations of a predicate in a clause, through the *recall number*. To do so, the arguments of a literal can be of three modes:

- Input (+) - an input variable of type T in a body literal B_i appears as an output variable of type T in a body literal that appears before B_i, or appears as an input variable of type T in the head of the clause.
- Output($-$) - an output variable of type T in the head of the clause must appears as an output variable of type T in any literal of the body of the clause.
- Constant(#) - an argument denoted by $\#T$ must be ground with terms of type T

It is necessary to define the modes of the arguments in literals when generating the bottom clause, since it is created from a MDIE approach. Also, the

Algorithm 1.5. Relational pathfinding antecedent generation using bottom clause

```
for each literal in bottom clause do
    return the end values with the respective variables they represent in the node
    been expanded
    return the variables of the particular literal
    if the literal has a variable in common with the variables bound to the end values
    then
        Instantiate the variables in the literal that represent some end value
        Search in the background knowledge a fact that unify with the literal
        if a fact is found then
            return the literal as a new antecedent
        else
            Go to the next literal in the bottom clause
    else
        Go to the next literal in the bottom clause
```

antecedents being added in the clause must be validated according to the modes declarations and the current clause in order to decide if the antecedent can effectively be added in the clause. These operations become necessary because the bottom clause considers the modes for all its literals together and when one of them is chosen to be added in the clause it may not satisfy some mode declaration. Suppose, for example, the clause below, whose body is empty

$$head(A, B).$$

and the mode definition to this predicate is

$$modeh(1, head(+a, +a)).$$

which indicates that this predicate can be used in the head of a clause, it is allowed only one instantiation of it ($recall = 1$) and their arguments are both of input, whose types are a.

Now, suppose we are adding antecedents in that clause and the bottom clause is as following

$$head(A, B) : -body_1(A, C), body_2(B, C), body_3(C, A)$$

and the modes declarations for the predicates of the body are

$$modeb(*, body_1(+a, -a))$$
$$modeb(*, body_2(+a, -a))$$
$$modeb(*, body_3(+a, -a))$$

which indicates that the clause can have infinite different instantiations of the predicates $body_i$ ($recall = *$) and the arguments of these predicate have both type a, where the first one is an input term and the second one is an output term.

If we did not consider the modes declarations the literal $body_3(C, A)$ could be added in the current clause. However, the variable C can not appear in the place of an input variable, according to modes definitions since it has not appeared before in the clause. Thus, it becomes necessary to verify if the antecedent follows the mode definition before it is scored to be put in the clause.

The modes declaration are used in different ways, depending on the addition antecedents' algorithm being used. In the case of *hill climbing algorithm*, only one antecedent is added at once and therefore, before scoring the antecedent, the algorithm verifies if it obeys the modes declaration taking into account the current clause. Thus, fewer literals are evaluated which decreases the cost of the addition antecedents process.

In the *relational pathfinding* algorithm, on the other hand, the antecedents cannot be validated just after they are picked from the bottom clause, since more than one antecedent will be added at once and therefore the mode declarations can be valid in the whole path but not in just one of the antecedents. Thus, the whole path is validated according to mode declarations. If the relational path does not obey the modes declarations it is discarded, which makes many fewer clauses be evaluated and consequently decreases the runtime of the addition of antecedents.

4 Experimental Results

We performed some experiments in order to verify the benefits of revising FOL theories using the bottom clause and modes declarations. Specifically, we would like to answer the following questions:

1. What is the runtime reduction using the bottom clause and modes during the revision process?
2. Are the accuracies obtained by the modified revision process at least maintained?
3. Is it possible to generate more comprehensible theories than the original theory revision system?
4. Are the predictive accuracies obtained by the modified system better than the accuracies returned by a traditional ILP system?

Four ILP benchmarks were considered to perform the experiments, all of them concerning the Alzheimer domain [King et al., 1995], which compares 37 analogues of Tacrine, a drug against Alzheimer's disease, according to four properties as described below:

1. inhibit **amine** re-uptake
2. low **toxic**ity,
3. high acetyl **choline**sterase inhibition and
4. good reversal of **scopolamine**-induced memory deficiency.

The datasets are composed of 686, 886, 1326 and 642 examples, respectively, equally divided in positive and negative examples.

4.1 Experimental Methodology

To overcoming the overfiting problem during training, similar to [Baião et al., 2003],we applied K-fold stratified cross validation approach to split the input data into disjoint training and test sets and, within that, a t-fold stratified cross-validation approach to split training data into disjoint training and tuning sets, keeping the rate of positive and negative examples in each fold [Kohavi, 1995]. We considered k=10 and t=5. The significance test used was *corrected two-tailed paired t-test* [Nadeau and Bengio, 2003], with $p < 0.05$.

The initial theories provided to the revision algorithm were obtained from Aleph system. A different theory was generated for each fold and each one of these theories was revised considering its respective fold (the same folds are used to generate and revise the theories). In this work, we are assuming that both modes and types definitions are correct and therefore can not be modified.

4.2 Results

In order to answer the first three questions above, we compared the runtime, the predictive accuracies and the size of the theories returned by original FORTE and FORTE considering modes and bottom clause. Both hill climbing and relational pathfinding algorithms were considered, running independently. The results are presented in Tables 1 and 2, where $\star$ indicates that modified FORTE with bottom clause and modes is significantly better than original FORTE. Each value in the tables is the average of 10-folds cross validation.

As it can be seen from the tables, FORTE using bottom clause and modes speeds up significantly the runtime of the revision process while still returns more comprehensible theories and maintains the predictive accuracy, compared to original FORTE. The ”?” in tables represents the cases where the algorithm is running for more than 100 hours without finishing even one fold. The biggest speedup was of 49× in the Toxic dataset, using the relational pathfinding algorithm. On average, it was obtained a speedup of three orders of magnitude.

We have done experiments considering 50% of the examples to generate the theories and all the set of examples to revise them. However, the results obtained from these experiments were similar to these ones presented here (generating and revising theories from all the set of examples).

Table 1. Runtime in seconds, predictive accuracy and size of the revised theories, using *hill climbing* algorithm for adding antecedents in a clause

	Original FORTE			FORTE BC MODES		
Datasets	Runtime	Accuracy	Size	Runtime	Accuracy	Size
Amine	616.90	72.47	96.3	42.32 $\star$	70.68	33.5 $\star$
Toxic	754.54	70.86	69.60	29.01 $\star$	72.31	31.60 $\star$
Choline	376.04	65.89	66.90	282.7	64.48	54.80
Scopo	3499.4	63.40	110	102.9 $\star$	63.53	44.43 $\star$

Table 2. Runtime in seconds, predictive accuracy and size of the revised theories, using Relational Pathfinding algorithm for adding antecedents in a clause

	Original FORTE			FORTE BC MODES		
Datasets	Runtime	Accuracy	Size	Runtime	Accuracy	Size
Amine	7347.8	73.24	181.1	271.5 $\star$	74.24	55.8 $\star$
Toxic	11766	78.56	129.84	241.0 $\star$	77.75	65.70 $\star$
Choline	> 360000	?	?	751.2 $\star$	64.48	54.8
Scopo	20061.34	65.47	253.8	444.4 $\star$	64.29	88.6 $\star$

In order to answer the last question, we exhibit in Figures 3, 4, 5 and 6, the learning curves comparing the modified version of FORTE considering the bottom clause and modes to Aleph, using both algorithms for adding antecedents. The points in the curves represent predictive accuracies obtained when learning (Aleph) and revising (Modified FORTE) considering different percentages of the examples in the datasets. Each point in the learning curves is the average of

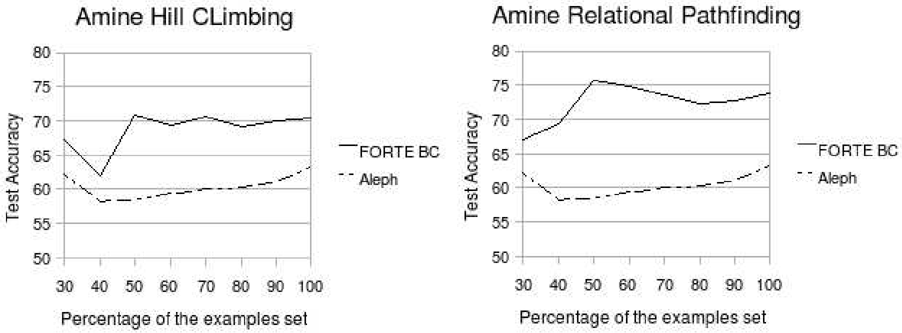


Fig. 3. Learning curves of Amine dataset

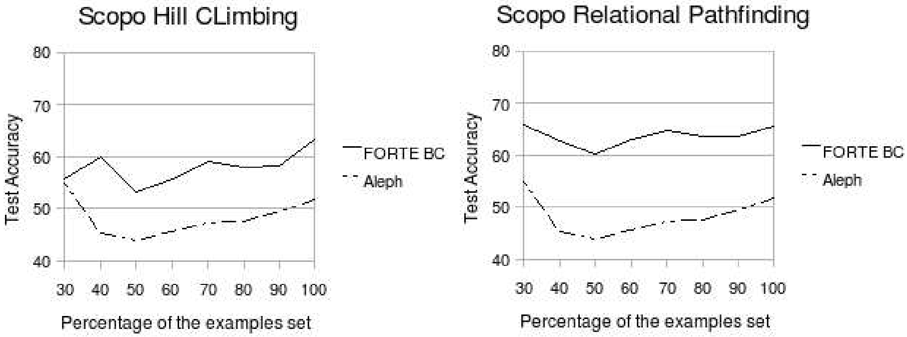


Fig. 4. Learning curves of Scopolamine dataset

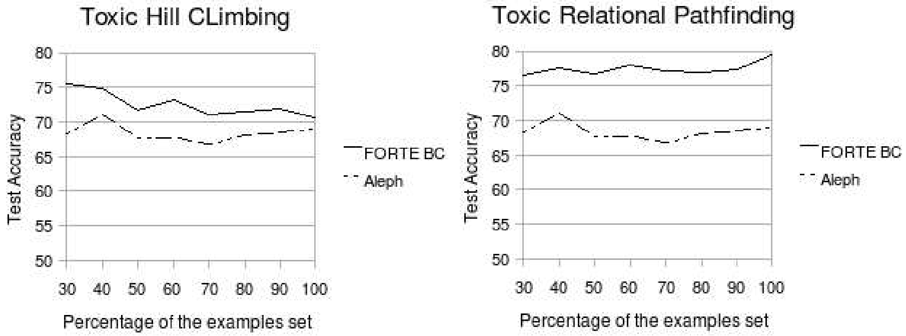


Fig. 5. Learning curves of Toxic dataset

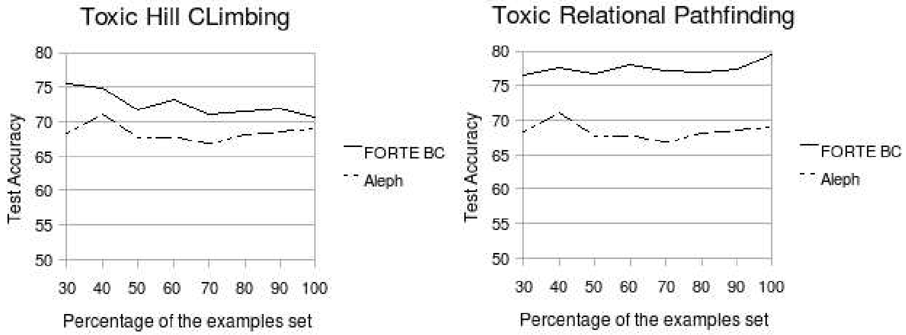


Fig. 6. Learning curves of Choline dataset

10-folds cross validation. The same set of examples was used to learning and revising the learned theories.

As we can observe from the figures, we positively answer the final question, since the accuracies obtained by the modified FORTE are always significantly better than accuracies obtained by Aleph, considering both algorithms for adding antecedents.

5 Conclusions and Future Work

The antecedent addition operation of FORTE theory revision system follows the top-down approach of FOIL. This leads to a huge search space which dominates the cost of the revision process; moreover, it does not properly explore the connections among variables. In this work the efficiency of theory revision antecedent addition was improved by the introduction of the bottom clause to define the search space of antecedents for both algorithms Hill Climbing and Relational Pathfinding, and the introduction of modes declarations, to define which literals of the bottom clause can effectively be added to the clause been revised.

Experimental results show that a significant increase on efficiency was reached when comparing the modified FORTE with modes and bottom clause to the original FORTE: on average, it was obtained a speedup of three orders of magnitude. Additionally, the accuracies obtained by the modified revision process ware at least maintained compared to the original system. Also, more comprehensible theories were generated when using the modified revision system.

The results also show that the predictive accuracies obtained by the modified system are always better than the accuracies returned by the traditional ILP system Aleph, considering both algorithms for adding antecedents.

When considering huge datasets, usually many facts of the background knowledge are irrelevant to a particular problem. Meanwhile, the bottom clause created by PROGOL/Aleph systems includes every piece of the background knowledge in its body, which might lead to huge bottom clauses. Therefore, a work in progress incorporates the ideas of BETH system [Tang et al., 2003] in the revision process of FORTE. BETH reduces the search space, by considering only the relevant literals of the bottom clause. In this way, the bottom clause becomes ”virtual” since it is not constructed beforehand, as in done in Aleph, but it has been discovered during the search for a good clause.

In [Ong et al., 2005], the relational pathfinding algorithm was applied to the learning phase of Aleph system. As a future work, we intend to compare this approach to the relational pathfinding with bottom clause and modes used in FORTE system, as proposed in this paper.

In [Paes et al., 2008], stochastic local search techniques were applied to the FORTE system and was obtained improvement in running time (on average, it was obtained a speedup of one order of magnitude) as well as in accuracy. As in the original system, the antecedent addition operation used all the knowledge base. Therefore, to further improve the efficiency a future work will merge both works and use the bottom clause as the search space when generating antecedents in the stochastic revision system.

As stated by [Mooney, 1992], knowledge assimilation requires the ability to incrementally revise a domain theory as new data is provided. However, for the best of our knowledge, only [Esposito et al., 2000] developed a first-order theory incremental revision system. Therefore, we intend to aggregate the ideas of incremental batch learners to our revision system.

Acknowledgments

The first and second authors are financially supported by Capes and the third author by CNPq. We would like to thank Bradley Richards, Raymond Mooney, Ashwin Srinivasan and Stephen Muggleton for making FORTE, Aleph and Progol systems available. We would also like to thank Eric Silva, Rafael Pereira and Guilherme Niedu for helping on the implementation of modes in FORTE system.

References

Baião et al., 2003. Baião, F., Mattoso, M., Shavlik, J., Zaverucha, G.: Applying theory revision to the design of distributed databases. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 57–74. Springer, Heidelberg (2003)

Bratko, 1999. Bratko, I.: Refining complete hypotheses in ILP. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 44–55. Springer, Heidelberg (1999)

Esposito et al., 2000. Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Multistrategy theory revision: Induction and abduction in inthelex. Machine Learning Journal 38(1/2), 133–156 (2000)

King et al., 1995. King, R.D., Sternberg, M.J.E., Srinivasan, A.: Relating chemical activity to structure: An examination of ILP successes. New Generation Computing 13(3-4), 411–433 (1995)

Kohavi, 1995. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI), pp. 1137–1145 (1995)

Mooney, 1992. Mooney, R.J.: Batch versus incremental theory refinement. In: Proceedings of the 1992 AAAI Spring Symposium on Knowledge Assimilation, Standford (1992)

Muggleton, 1992. Muggleton, S.: Inductive logic programming. Academic Press, New York (1992)

Muggleton, 1995. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)

Nadeau and Bengio, 2003. Nadeau, C., Bengio, Y.: Inference for the generalization error. Machine Learning 52(3), 239–281 (2003)

Ong et al., 2005. Ong, I.M., Dutra, I.C., Page, D., Costa, V.C.: Mode directed path finding. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 673–681. Springer, Heidelberg (2005)

Paes et al., 2008. Paes, A., Zaverucha, G., Costa, V.S.: Revising first-order logic theories from examples through stochastic local search. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 200–210. Springer, Heidelberg (2008)

Quinlan, 1990. Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)

Richards and Mooney, 1995. Richards, B.L., Mooney, R.J.: Automated refinement of first-order Horn-clause domain theories. Machine Learning 19(2), 95–131 (1995)

Srinivasan, 2001. Srinivasan, A.: The Aleph Manual (2001)

Tang et al., 2003. Tang, L.R., Mooney, R.L., Melville, P.: Scaling up ilp to large examples: Results on link discovery for counter-terrorism. In: Proceedings of the KDD-2003 Workshop om Multi-Relational Data Mining, Washington, DC, pp. 107–121 (2003)

Wrobel, 1996. Wrobel, S.: First-order theory refinement. In: Raedt, L.D. (ed.) Advances in Inductive Logic Programming, pp. 14–33. IOS Press, Amsterdam (1996)

DL-FOIL
Concept Learning in Description Logics

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

LACAM – Dipartimento di Informatica – Università degli studi di Bari
Via Orabona, 4 – 70125 – Bari, Italy
{fanizzi,claudia.damato,esposito}@di.uniba.it

Abstract. In this paper we focus on learning concept descriptions expressed in Description Logics. After stating the learning problem in this context, a FOIL-like algorithm is presented that can be applied to general DL languages, discussing related theoretical aspects of learning with the inherent incompleteness underlying the semantics of this representation. Subsequently we present an experimental evaluation of the implementation of this algorithm performed on some real ontologies in order to empirically assess its performance.

1 Introduction

Description Logics (DLs) is the family of representation languages underlying the standard ontology languages designed for knowledge bases in the Semantic Web [3]. These logics constitute a specific fragment of First Order Logic (FOL) that differs from the standard clausal forms employed in Inductive Logic Programming and related multi-relational settings, namely they have a different syntax and especially very different semantics [4, 12]. These considerations justify the growing interest in investigating concept learning in such new formalisms.

Early work on learning in DLs essentially focused on demonstrating the PAC-learnability for various languages derived from Classic. In particular, Cohen and Hirsh investigate the CoreClassic DL proving that it is not PAC-learnable [6] as well as demonstrating the PAC-learnability of a peculiar class among its sub-languages such as C-CLASSIC [7], through the LcsLearn algorithm, together with an empirical evaluation of its implementation.

These approaches tend to cast supervised concept learning as performed through a structural generalizing operator working on equivalent graph representations of the concept descriptions. It is also worth mentioning unsupervised learning methodologies for DL concept descriptions, whose prototypical example is Kluster [16], a polynomial-time algorithm for the induction of BACK terminologies, which exploits the tractability of the standard inferences in this DL language [1].

More recently also learning in hybrid languages, mixing clausal and description logics, have been investigated. Kietz [15] studied the learnability of DL programs. Other related approaches propose learning methods for hybrid languages,

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 107–121, 2008.

such as Carin-$\mathcal{ALN}$ [21] and $\mathcal{AL}$-log [19], that allow simple DLs to be combined with Datalog.

In this work, we focus on learning concepts in expressive DLs endowed with most of the constructors, seeking for a tradeoff between expressiveness, efficiency and completeness of the resulting learning system. Indeed, more expressiveness requires more computational resources for the most common inferences; hence, algorithms dealing with larger DL languages must face the complexity of reasoning. In our vision inductive inference should be employed in order to help the knowledge engineer construct new concept definitions that can eventually be further refined, either manually or by means of other semi-automatic tools. This would save the engineer from finding trivial regularities in the examples so that he/she could concentrate the efforts in refining the induced definition.

We implemented a specific new version of the Foil algorithm [20], resulting in the DL-Foil system, that is adapted to learning the DL representations supporting the OWL-DL language. The main components of this new systems are represented by a set of refinement operators borrowed from other similar systems [13, 18] proposed in the literature and by a different gain function which must take into account the open-world assumption, namely, many instances may be available which cannot be ascribed to the target concept nor to its negation. This requires a different setting, similar to learning with unknown class attributes [11], requiring a special treatment of the unlabeled individuals.

A preliminary experiment presented in this paper applies the DL-Foil system to real ontologies which represent a real testbed w.r.t. the datasets employed for testing YinYang [13] and DL-Learner [18] which limit their scope to $\mathcal{ALC}$ ontologies. This also demonstrates the usage of the method as a means for performing approximations of concepts across ontologies described with different languages [5, 1].

The outcomes in terms of precision and recall were satisfactory, despite of the employment of incomplete refinement operators. However these outcomes are not as meaningful as they might be in a standard (closed-world) setting. Namely, since many test instances might not be proved to be examples or counter-examples, we resort to different performance metrics, measuring the alignment of the classification decided by the concept descriptions induced by DL-Foil with the classification derived deductively by a DL reasoner. This allows measuring the amount of unlabeled instances that may be ascribed to the newly induced concepts (or to their negations), which may constitute a real added value brought by the inductive method. Actually these abductive conclusions should be evaluated by the expert who helped during the construction of the ontology. However, this is not always possible.

The paper is organized as follows. After the next section introducing the representation, in Sect. 3 the refinement operators and the algorithm exploiting them are discussed and then, Sect. 4, the adaptation of the Foil algorithm. In Sect. 5 the experiments proving the effectiveness of the approach are reported. Finally, possible developments are reported in Sect. 6.

2 Description Logics: Syntax and Semantics

In this section we shortly recall syntax and semantics of the DL representation. For brevity, we cannot report syntax and semantics of the various constructors, which can be easily be found in the reference manual [1]. In turn, the DL concept descriptions are straightforwardly mapped onto XML serializations of the standard ontology languages [9].

Roughly, the formalisms are concept-centric: they distinguish *concepts* from *relations* that are used to describe restrictions on concepts. In a DL language, primitive *concepts* $N_C = \{C, D, \ldots\}$ are interpreted as subsets of a domain of objects (resources) and primitive *roles* $N_R = \{R, S, \ldots\}$ are interpreted as binary relations on such a domain (properties). Individuals represent the objects through names from $N_I = \{a, b, \ldots\}$.

Complex concept descriptions are built using atomic concepts and primitive roles by means of specific constructors. The meaning of the descriptions is defined by an *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is the *domain* of the interpretation and the functor $\cdot^\mathcal{I}$ stands for the *interpretation function*, mapping the intension of concepts and roles to their extension (respectively, a subset of the domain and a relation defined on such domain).

The *top* concept $\top$ is interpreted as the whole domain $\Delta^\mathcal{I}$, while the *bottom* concept $\bot$ corresponds to $\emptyset$. Complex descriptions can be built in $\mathcal{ALC}$ using the following constructors[1]. The language supports *full negation*: given any concept description C, denoted $\neg C$, it amounts to $\Delta^\mathcal{I} \setminus C^\mathcal{I}$. The *conjunction* of concepts, denoted with $C_1 \sqcap C_2$, yields an extension $C_1^\mathcal{I} \cap C_2^\mathcal{I}$ and, dually, concept *disjunction*, denoted with $C_1 \sqcup C_2$, yields $C_1^\mathcal{I} \cup C_2^\mathcal{I}$. Finally, there are two restrictions on roles: the *existential restriction*, denoted with $\exists R.C$, and interpreted as the set $\{x \in \Delta^\mathcal{I} \mid \exists y \in \Delta^\mathcal{I} : (x, y) \in R^\mathcal{I} \wedge y \in C^\mathcal{I}\}$ and the *value restriction*, denoted with $\forall R.C$, whose extension is $\{x \in \Delta^\mathcal{I} \mid \forall y \in \Delta^\mathcal{I} : (x, y) \in R^\mathcal{I} \rightarrow y \in C^\mathcal{I}\}$.

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains two components: a T-box $\mathcal{T}$ and an A-box $\mathcal{A}$. $\mathcal{T}$ is a set of terminological axioms $C \sqsubseteq D$, yet we will consider only definitions $A \equiv D$, where $A \in N_C$ is a concept name (atomic) and D is a concept description given in terms of the language constructors, meaning $A^\mathcal{I} = D^\mathcal{I}$. The ABox $\mathcal{A}$ contains extensional assertions (ground facts) on concepts and roles, e.g. $C(a)$ and $R(a, b)$, meaning, respectively, that $a^\mathcal{I} \in C^\mathcal{I}$ and $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$. Note that the *unique names assumption* is not necessarily made[2].

Further constructors extend the expressiveness of the $\mathcal{ALC}$ language. We are interested in the languages that constitute the counterpart of OWL-DL, namely $\mathcal{SHOIQ}(D)$ that, roughly, extends $\mathcal{ALC}$ with transitive roles, role hierarchies,

[1] In fact, the $\mathcal{ALC}$ corresponds to the fragment of first-order logic obtained by restricting the syntax to formulae containing two variables. $\mathcal{ALC}$ has a modal logic counterpart, namely the multi-modal version of the logic K [1].

[2] Different individual names may be mapped onto the same domain object, in principle.

individual classes, inverse roles and qualified number restrictions. Besides, concrete domains[3] ($\mathbf{D}$) can be dealt with.

The set-theoretic notion of *subsumption* between concepts (or roles) can be given in terms of the interpretations:

Definition 2.1 (subsumption). *Given two concept descriptions C and D in $\mathcal{T}$, C* subsumes *D, denoted by $C \sqsupseteq D$, iff for every interpretation $\mathcal{I}$ of $\mathcal{T}$ it holds that $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. Hence, $C \equiv D$ amounts to $C \sqsupseteq D$ and $D \sqsupseteq C$.*

Example 2.1. A concept definition in the proposed language may be:
$\mathtt{Father} \equiv \sqcap \mathtt{Male} \sqcap \exists \mathtt{hasChild}.\top$
which translates the sentence: *"a father is a male that has someone as his child"* ($\top$ denotes the most general concept).

Now, if we define two new concepts:
$\mathtt{FatherWithoutSons} \equiv \mathtt{Male} \sqcap \exists \mathtt{hasChild}.\top \sqcap \forall \mathtt{hasChild}.(\neg \mathtt{Male})$
and
$\mathtt{Parent} \equiv (\mathtt{Male} \sqcup \mathtt{Female}) \sqcap \exists \mathtt{hasChild}.\top$
then it is easy to see that $\mathtt{Father} \sqsupseteq \mathtt{FatherWithoutSons}$ and $\mathtt{Parent} \sqsupseteq \mathtt{Father}$, yet $\mathtt{Father} \not\sqsupseteq \mathtt{Parent}$ and $\mathtt{FatherWithoutSons} \not\sqsupseteq \mathtt{Father}$.

A-box assertions are ground facts like:
$\mathtt{Father}(\mathtt{edward})$, $\mathtt{Male}(\mathtt{charles})$, $\mathtt{hasChild}.\mathtt{Male}(\mathtt{edward},\mathtt{charles})$,
$\geq 1.\mathtt{hasChild}(\mathtt{edward})$, $\exists \mathtt{hasChild}.\top(\mathtt{charles})$ and so on.

The most important inference service from the inductive point of view is *instance checking* [1], that amounts to ascertain class-membership assertions: $\mathcal{K} \models C(a)$, where $\mathcal{K}$ is the knowledge base a is an individual name and C is a concept definition given in terms of the concepts accounted for in $\mathcal{K}$.

An important difference with other FOL fragments is the *open-world assumption* (OWA) which makes it more difficult to answer class-membership queries. Thus it may happen that an object that cannot be proved to belong to a certain concept is not necessarily a counterexample for that concept. That would only be interpreted[4] as a case of insufficient (incomplete) knowledge for that assertion.

Example 2.2 (cont'd). Given the concepts
$\mathtt{MotherWithoutDaughters} \equiv \mathtt{Mother} \sqcap \forall \mathtt{hasChild}.\neg \mathtt{Female}$
and
$\mathtt{Super\text{-}motherMother} \geq 3.\mathtt{hasChild}$
and the ABox:

$$\begin{aligned}\mathcal{A} = \{\, & \mathtt{Female}(\mathtt{elisabeth}), \mathtt{Female}(\mathtt{diana}),\\ & \mathtt{Male}(\mathtt{charles}), \mathtt{Male}(\mathtt{edward}), \mathtt{Male}(\mathtt{andrew}),\\ & \mathtt{MotherWithoutDaughters}(\mathtt{diana}),\\ & \mathtt{hasChild}(\mathtt{elisabeth},\mathtt{charles}), \mathtt{hasChild}(\mathtt{elisabeth},\mathtt{edward}),\\ & \mathtt{hasChild}(\mathtt{elisabeth},\mathtt{andrew}), \mathtt{hasChild}(\mathtt{diana},\mathtt{william}),\\ & \mathtt{hasChild}(\mathtt{charles},\mathtt{william}) \,\}\end{aligned}$$

[3] Concrete domains include data types such as numerical types, but also more elaborate domains, such as tuples of the relational calculus, spatial regions, or time intervals.

[4] A model could be constructed for both the membership and non-membership case [1].

One may infer
$\mathcal{K} \models \texttt{Super-mother}(\texttt{elisabeth})$
but not
$\mathcal{K} \models \texttt{MotherWithoutDaughters}(\texttt{elisabeth})$
because it may well be that a daughter is not known yet.

This is perfectly compatible with the typical scenario related to the Semantic Web, where new resources may continuously be made available across the Web, hence a complete knowledge cannot be assumed at any time.

The other inference service provided by DL reasoner is concept *retrieval*: given a certain concept, retrieve all the individuals that can be proved to belong to it.

3 Learning as Search in DLs

After recalling the basics of DLs, we are ready to formally define the learning problem in this setting.

Definition 3.1 (learning problem). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base.*
Given

- *a (new) target concept name C*
- *a set of positive and negative examples $\mathsf{Ind}^+_C(\mathcal{A}) \cup \mathsf{Ind}^-_C(\mathcal{A}) \subseteq \mathsf{Ind}(\mathcal{A})$ where $\mathsf{Ind}(\mathcal{A})$ is the set of individuals occurring in $\mathcal{A}$,*
  $\mathsf{Ind}^+_C(\mathcal{A}) = \{a \in \mathsf{Ind}(\mathcal{A}) \mid \mathcal{K} \models C(a)\}$, $\mathsf{Ind}^-_C(\mathcal{A}) = \{a \in \mathsf{Ind}(\mathcal{A}) \mid \mathcal{K} \models \neg C(a)\}$

***Suppose**, in case a definition for C is already available (refinement problem), that it holds:*

$$\exists a \in \mathsf{Ind}^+_C(\mathcal{A}) \quad \mathcal{K} \not\models C(a) \qquad \text{or} \qquad \exists b \in \mathsf{Ind}^-_C(\mathcal{A})) \quad \mathcal{K} \not\models \neg C(b)$$

***Buld** a concept definition $C \equiv D$ such that*

$$\mathcal{K} \models C(a) \quad \forall a \in \mathsf{Ind}^+_C(\mathcal{A}) \qquad \text{and} \qquad \mathcal{K} \not\models C(b) \quad \forall b \in \mathsf{Ind}^-_C(\mathcal{A})$$

The definition given above can be interpreted as a generic supervised concept learning task. $\mathsf{Ind}^+_C(\mathcal{A})$ and $\mathsf{Ind}^-_C(\mathcal{A})$ represent respectively the sets of positive and negative examples, whereas $C \equiv D$ is the hypothesis to be induced. The intermediate clause was added for generalizing the definition covering also refinement problems, in which a definition for C is already available but it may be defective w.r.t. to some positive or negative examples.

As known from related works [10, 13, 17], the subsumption relationship (see Def. 2.1) induces a partial order on the space of all the possible concept descriptions. Hence the inductive problem stated above can be cast as a search of the right concept definition (hypothesis) in the induced search space.

In such a setting, one can define suitable operators in order to traverse the search space. As usual in inductive search, we will define two operators in order to obtain, given a starting (incorrect) hypothesis in the search space, one (or some) of its generalizations/specializations.

Of course, given a set of concept definitions belonging to the space of the descriptions allowed by the reference language, which is partially ordered by subsumption, there is an infinite number of generalizations and specializations. Usually one tries to devise operators that can move efficiently throughout the space in pursuit of one of the target hypotheses.

Now we can define both the downward (specializing) operator ρ and the upward (generalizing) operator δ [13]:

Definition 3.2 (downward operator ρ). *$\rho = (\rho_\sqcup, \rho_\sqcap)$, where:*
$[\rho_\sqcup]$ given a description in normal form $D = D_1 \sqcup \cdots \sqcup D_n$:

- *$D' \in \rho_\sqcup(D)$ if $D' = \bigsqcup_{1 \leq i,j \leq n} D_i$ for some $j \neq i, 1 \leq j \leq n$*
- *$D' \in \rho_\sqcup(D)$ if $D' = D'_i \sqcup \bigsqcup^{j \neq i}_{1 \leq i,j \leq n} D_k$ for some $D'_i \in \rho_\sqcap(D_i)$*

$[\rho_\sqcap]$ given a conjunctive description $C = C_1 \sqcap \cdots \sqcap C_m$ and a set of concept descriptions $\mathcal{A}^- = \{E_k \mid 1 \leq k \leq p\}$:

- *$C' \in \rho_\sqcap(C)$ if $C' = C \sqcap C_{j+1}$*
  for some $C_{j+1} \not\sqsupseteq C$ and $C_{j+1} \not\sqsupseteq E_h$, for $h \in \{1, \ldots, p\}$
- *$C' \in \rho_\sqcap(C)$ if $C' = (C \sqcup \neg C_j) \sqcap C'_j$ for some $j \in \{1, \ldots, m\}$, where:*
  - *$C'_j = \exists R.D'_j$, $C_j = \exists R.D_j$ and $D'_j \in \rho_\sqcup(D_j)$ or*
  - *$C'_j = \forall R.D'_j$, $C_j = \forall R.D_j$ and $D'_j \in \rho_\sqcup(D_j)$*

$\rho_\sqcup$ simply drops one top-level disjunct or replaces it with a downward refinement obtained with $\rho_\sqcap$. $\rho_\sqcap$ adds new conjuncts or replaces one with a refinement obtained by specializing (through $\rho_\sqcup$) the concepts in the scope of a universal or existential restriction.

Definition 3.3 (upward operator δ). *$\delta = (\delta_\sqcup, \delta_\sqcap)$, where:*
$[\delta_\sqcup]$ given a description in normal form $D = D_1 \sqcup \cdots \sqcup D_n$ and a set of concept descriptions $\mathcal{A}^+ = \{E_h \mid 1 \leq h \leq p\}$:

- *$D' \in \delta_\sqcup(D)$ if $D' = D \sqcup D_{n+1}$ for some D_{n+1} such that $D_{n+1} \not\sqsubseteq D_i$, $i \in \{1, \ldots, n\}$ and $D_{j+1} \sqsupseteq E_h$ for some $h \in \{1, \ldots, p\}$*
- *$D' \in \delta_\sqcup(D)$ if $D' = D'_i \sqcup \bigsqcup^{k \neq i}_{1 \leq i,k \leq n} D_i$ for some $D'_i \in \delta_\sqcap(D_i)$*

$[\delta_\sqcap]$ given a conjunctive description $C = C_1 \sqcap \cdots \sqcap C_m$:

- *$C' \in \delta_\sqcap(C)$ if $C' = \bigsqcap^{j \neq i}_{1 \leq i,j \leq m} C_i$*
- *$C' \in \delta_\sqcap(C)$ if $C' = \bigsqcap^{j \neq i}_{1 \leq i,j \leq m} C_i \quad \sqcap C'_j$, where:*
  - *$C'_j = \exists R.D'_j$, $C_j = \exists R.D_j$ and $D'_j \in \delta_\sqcup(D_j)$ or*
  - *$C'_j = \forall R.D'_j$, $C_j = \forall R.D_j$ and $D'_j \in \delta_\sqcup(D_j)$*

$\delta_\sqcup$ and $\delta_\sqcap$ simply perform dual operation w.r.t. $\rho_\sqcup$ and $\rho_\sqcap$, respectively. See [13] for examples.

Other operators [13] may exploit also the knowledge conveyed by the positive and negative examples in order to prune the possible results yielded by a single generalization/specialization step and to better direct the search for suitable solutions to the problem. Instead of using the examples in a mere *generate-and-test* strategy based on these operators, they can be exploited more directly, in order to influence the choices made during the refinement process.

These operators cannot be complete for most expressive DLs [17]. However, we are not looking for too precise operators that likely lead to overfit the data. E.g. the LCS function [7] is a generalizing operator that may be used to compute upper refinement of a concept w.r.t. to an uncovered positive instance (represented by its most specific concept, see [1, 13]), yet this results in a simple union of the two descriptions which deprives the result of any added generalization w.r.t. future examples.

The next step is embedding these simple operators in a suitable learning algorithm.

4 The Learning Algorithm

Various search strategies have been experimented as well as other evaluation measures. Those that we will present in the following are those which gave the best results.

The main aim of this work was conceiving a learning algorithm that could overcome two limitation of the current DL learning systems, namely avoiding the computation of the most specific concepts and the language dependence. Indeed, following the early work, YinYang [13] requires lifting the instances to the concept level through a suitable approximate operator and then start learning from such extremely specific concept descriptions. This setting has the disadvantages of approximation and language-dependence.

DL-Learner [18] partly mitigates these disadvantages for it does not need to compute such approximations since it is essentially based on a genetic programming procedure based on refinement operators whose fitness is computed on the grounds of the covered instances.

Also, in our new algorithm conceived in order to solve the learning problem, the (downward) refinement operators previously defined play a central role. A sketch of the main routine that makes up the algorithm is reported in Fig. 1.

Like in the original Foil algorithm [20], the generalization routine computes (partial) generalizations as long as they do not cover any negative example. If this occurs, the specialization routine is invoked for solving these sub-problems. This routine applies the idea of specializing using the (incomplete) refinement operator defined in the previous section. The specialization continues until no negative example is covered (or a very limited amount[5] of them). The partial generalizations built on each outer loop are finally grouped together in a disjunction which is an allowed constructor for DLs more expressive than (or equal to) $\mathcal{ALC}$. Also the outer while-loop can be exited before covering all the positive examples for avoiding overfitting generalizations.

[5] The actual exit-condition for the inner loop being: $|Negatives| - |CoveredNegatives| < \varepsilon$, for some small constant ε.

```
function DL-Foil(Positives, Negatives, Unlabeled): Generalization
input    Positives, Negatives, Unlabeled: positive, negative and unlabeled individuals
output   Generalization: concept definition

begin
Generalization $\leftarrow \bot$
PositivesToCover $\leftarrow$ Positives
while PositivesToCover $\neq \emptyset$ do
    begin
    PartialDef $\leftarrow \top$
    CoveredNegatives $\leftarrow$ Negatives
    while CoveredNegatives $\neq \emptyset$ do
        begin
        PartialDef $\leftarrow$ specialize(PartialDef, PositivesToCover, CoveredNegatives, Unlabeled)
        CoveredNegatives $\leftarrow \{n \in Negatives \mid \mathcal{K} \models \neg PartialDef(n)\}$
        end
    CoveredPositives $\leftarrow \{p \in PositivesToCover \mid \mathcal{K} \models PartialDef(p)\}$
    Generalization $\leftarrow$ Generalization $\sqcup$ PartialDef
    PositivesToCover $\leftarrow$ PositivesToCover $\setminus$ CoveredPositives
    end
return Generalization
end
```

Fig. 1. The main generalizing routine in DL-Foil

The specialization function specialize (reported in Fig. 2) is called from within the inner loop of the generalization procedure in order to specialize an overly general partial generalization. The function searches for proper refinements that provide at least a minimal gain (see below) fixed with a threshold (*MINGAIN*).

In Foil-I [14], the gain function has to take into account incomplete examples. Similarly to a semi-supervised learning setting, the gain function that is evaluated for choosing the best refinement is computed as follows:

$$p_1 \cdot \left[\log \frac{p_1 + u_1 w_1}{p_1 + n_1 + u_1} - \log \frac{p_0 + u_0 w_0}{p_0 + n_0 + u_0} \right]$$

where p_1, n_1 and u_1 represent, resp., the number of positive, negative and unlabeled examples covered by the specialization and p_0, n_0 and u_0 stand for the number of positive, negative and unlabeled examples covered by the former definition, and the weights w_0, w_1 are determined by the prior probability of the positive examples, resp., in the current and former concept definition. In order to avoid null numerators, a further correction of the probabilities is performed by resorting to the m-estimate procedure.

Despite of its simplicity the complexity of the algorithm is largely determined by the calls to reasoning services, namely subsumption and instance-checking. If we consider the $\mathcal{ALC}$ logic the complexity of these inferences is P-space. However, the algorithm can be thought as building an (upper) $\mathcal{ALC}$-approximation of target concepts, given a knowledge base that can contain definitions expressed in more complex languages, which in turn require more complex reasoning algorithms (see details on $\mathcal{SHOIN}(\mathbf{D})$ [1]). The number of nodes visited during

```
function SPECIALIZE($PartialDef$, $Positives$, $Negatives$, $Unlabeled$): $Refinement$
input    $PartialDef$: concept definition
         $Positives$, $Negatives$, $Unlabeled$: positive, negative and unlabeled individuals
output   $Refinement$: concept definition
const    $MAXNUM$: maximum real number
         $MINGAIN$: minimal acceptable gain
         $NUMSPECS$: number of specializations to be generated

begin
$bestGain \leftarrow -MAXNUM$
while $bestGain < MINGAIN$ do
    for $i \leftarrow 1$ to $NUMSPECS$ do
        begin
        $specialization \leftarrow$ GETRANDOMREFINEMENT($\rho$, $PartialDef$)
        $CoveredNegatives \leftarrow \{n \in Negatives \mid \mathcal{K} \models \neg PartialDef(n)\}$
        $CoveredPositives \leftarrow \{p \in Positives \mid \mathcal{K} \models PartialDef(p)\}$
        $thisGain \leftarrow$ GAIN($CoveredPositives, CoveredNegatives, Unlabeled, Positives, Negatives$)
        if $thisGain > bestGain$ then
            begin
            $bestConcept \leftarrow refConcept$
            $bestGain \leftarrow thisGain$
            end
        end
return $Refinement$
end
```

Fig. 2. The specializing routine in DL-FOIL

the search grows with the expressiveness of the language because the algorithm searches a sub-space of the actual search space induced by the adopted language.

5 Preliminary Experiments

5.1 Experimental Setting

In order to perform a preliminary experimentation on real ontologies DL-FOIL, it was applied to a number of concept retrieval problems solved by using inductive classification of the individuals w.r.t. a number of query concepts.

To this purpose, we selected a number of ontologies from different domains represented in OWL, namely: NEWTESTAMENTNAMES (NTN) from the Protégé library[6] accounting for characters and places mentioned in the book, the BioPax glycolysis ontology[7] (BioPax) describing the glycolysis pathway from the EcoCyc database, translated into BioPax format. It is intended to show what a pathway from an existing database might look like after being translated into BioPAX format. The FINANCIAL ontology[8], built for eBanking applications, deals with accounts, holders, loans and related events. Tab. 1 summarizes important details

[6] http://protege.stanford.edu/plugins/owl/owl-library

[7] http://www.biopax.org/Downloads/Level1v1.4/biopax-example-ecocyc-glycolysis.owl

[8] http://www.cs.put.poznan.pl/alawrynowicz/financial.owl

Table 1. Facts concerning the ontologies employed in the experiments

Ontology	DL language	#concepts	#object prop.	#data prop.	#individuals
BioPAX	$\mathcal{ALCHF}(D)$	28	19	30	323
NTN	$\mathcal{SHIF}(D)$	47	27	8	676
Financial	$\mathcal{ALCIF}$	60	17	0	1000

concerning these ontologies. The sizes of the ontologies are to be measured in terms of thousands of triples.

For each ontology, 30 target queries were randomly generated by composition of 2 through 8 primitive or defined concepts from each knowledge base by means of the concept constructors: intersection, union, universal or existential restrictions. Given the overall set of individuals mentioned in the ABox, this was split in training and test sets according to the ten-fold cross-validation procedure. A standard reasoner[9] was employed to decide their class-membership (and non-membership) w.r.t. the query concepts. The performance was evaluated comparing the definitions of the query concepts induced by the system to those that were randomly generated, determining the class-membership of the test examples.

Note that the constant *NUMSPECS* that determines the maximum number of specializations evaluated per turn was set to 15 (larger numbers yield better results but may lower the efficiency).

5.2 Results

Standard IR measures. Initially the standard IR measures precision, recall, F_1-measure were employed to evaluate the system performance. Specifically we considered a two-way classification where relevance coincides with class-membership and the rest of instances are considered irrelevant (true negatives if nothing could be concluded for their membership).

The outcomes are reported in Fig. 2. For each knowledge base, we report the average values obtained over the 30 queries as well as their standard deviation and minimum-maximum ranges of values.

It is possible to note that precision and recall are generally good but not very high which is also reflected by the F-measure. This happens because these parameters are taken on the grounds of the positive instances which are likely to be in a limited amount w.r.t. the overall number of individuals, especially when random concepts are considered. This is also the cause for the variance to be quite high for all experiments. For the sake of repeatability, we did not employ artificially populated ontologies. We rather employed ontologies as they can be found in the Web. Of course more largely populated ontologies would help assess more stable results. Actually for Financial we selected a reduced number of

[9] Pellet v. 1.5.1 was employed for instance-checking. The reasoner is publicly available at: http://pellet.owldl.com.

Table 2. Experimental results in terms of standard IR measures: averages ± standard deviations and [min,max] intervals

ontology	precision	recall	F_1-measure
BioPax	66.0 ± 24.1 [28.3;99.4]	76.5 ± 21.7 [36.5;100.0]	69.6 ± 21.0 [31.9;99.7]
NTN	59.0 ± 36.8 [18.8;100.0]	64.9 ± 25.7 [27.9;100.0]	59.1 ± 30.0 [22.5;100.0]
Financial	62.1 ± 40.7 [19.1;99.1]	64.8 ± 37.0 [24.3;99.0]	63.3 ± 39.1 [66.7;21.3;99.0]

instances. Selecting larger numbers of individuals, we could obtain better and more stable results.

The reason for precision being less than recall is probably due to the OWA. Indeed, in many cases it was observed that the inductive classification deemed some individuals as relevant for the query issued while the DL reasoner was not able to assess this relevance and this was computed as a mistake while it may likely turn out to be a correct inference when judged by a human agent.

Because of the problems issued by the OWA, different indices would be needed in this case that may make explicit both the rate of inductively classified individuals and the nature of the mistakes.

Alternative measures. Due to the OWA, cases were observed when, it could not be (deductively) ascertained whether a resource was relevant or not for a given query. Then a three-way classification is preferable. Hence, we introduced the following indices for a further evaluation [8]. Essentially they measure the correspondence between the classification provided by the reasoner for the instances w.r.t. the test concept and the definition induced by our system.

- *match rate*: number of cases of individuals that got exactly the same classification with both definitions;
- *omission error rate*: amount of individuals for which class-membership w.r.t. the given query could not determined using the induced definition, while they actually belong (do not belong) to the query concept;
- *commission error rate*: amount of individuals found not to belong to the query concept according to the induced definition, while they actually belong to it and vice-versa.
- *induction rate*: amount of individuals found to belong or not to belong to the query concept according to the induced definition, while either case is not logically derivable from the knowledge base with the original definition

Tab. 3 reports the outcomes in terms of these new indices. Preliminarily, we found that the search procedure was accurate enough: it made few critical mistakes especially when the considered concepts are known to have many examples (and counterexamples) in the ontology. However, it is important to note that, in each experiment, the commission error was limited but not absent, as

Table 3. Results with alternative indices: averages ± standard deviations and [min,max] intervals

ontology	match rate	commission error rate	omission error rate	induction rate
BioPax	**76.9** ± 15.7 [56.0;99.4]	**19.7** ± 15.9 [0.0;44.0]	**7.0** ± 20.0 [0.0;64.0]	**7.5** ± 23.7 [0.0;75.0]
NTN	**78.0** ± 19.2 [43.6;95.4]	**16.1** ± 4.0 [0.0;10.8]	**6.4** ± 8.1 [1.2;21.3]	**14.0** ± 10.1 [2.5;27.1]
Financial	**75.5** ± 20.8 [50.2;98.1]	**16.1** ± 12.8 [0.6;25.6]	**4.5** ± 5.1 [1.0;12.6]	**3.7** ± 7.9 [0.3;18.1]

in the experiments with other classification methods [8]. The cases of queries for which this measure was high are due to the limited amount of examples available (too narrow concepts). Even few mistakes provoked high error rates. This is also due to the absence of axioms stating explicitly the disjointness of some concepts.

Also the omission error rates are quite low. They are comparable with the amount of inductive conclusions that could be drawn with the induced definitions. Again these figures may vary as a consequence of the presence / absence of knowledge about the disjunction of (sibling) concepts in the subsumption hierarchies. In an ontology population perspective, the cases of induction are interesting because they suggest new assertions which cannot be logically derived by using a deductive reasoner yet they might be used to complete a knowledge base [2], e.g. after being validated by an ontology engineer. Better results were obtained on the same task with different inductive methods (instance-based learning [8]). Yet, with DL-Foil we have the added value of having an intensional definition of the target concepts.

The elapsed time (not reported here) was very limited: about 0.5 hour for a whole ten-fold cross validation experiment including the time consumed by the reasoner to make the judgments.

5.3 Learned Concepts

For each ontology, we report examples of the concept descriptions that were learned during the experiments and compare them to the query concept that generated the examples and counterexamples.

BioPax
induced:

```
Or( And( physicalEntity protein) dataSource)
```

original:

```
Or( And( And( dataSource externalReferenceUtilityClass)
ForAll(ORGANISM ForAll(CONTROLLED phys icalInteraction))) protein)
```

NTN
induced:
`Or( EvilSupernaturalBeing Not(God))`
original:
`Not(God)`

Financial
induced:
`Or( Not(Finished) NotPaidFinishedLoan Weekly)`
original:
`Or( LoanPayment Not(NoProblemsFinishedLoan))`

These concepts totally overlap in terms of their extensions w.r.t. the known individuals.

Of course for a correct qualitative interpretation of the value of these concepts some familiarity is assumed with the ontologies. As mentioned, they are all freely available at standard repositories.

6 Conclusions and Outlook

In this work, we investigated learning expressive DLs supporting ontology languages such as OWL. We implemented a Foil-like algorithm in the DL-Foil system, that is an adaptation to the issues related to the different representation. The main components of this new system are represented by a set of refinement operators and by a different gain function which takes into account the open-world assumption. Namely many instances may be available which cannot be ascribed to the target concept nor to its negation. This requires a different setting and a special treatment of the unlabeled individuals.

A preliminary experimentation has been presented in this paper, applying the DL-Foil system to learning from individuals in real ontologies which represent a harder testbed w.r.t. the datasets employed for testing YinYang [13] and DL-Learner [18] that limited their scope to $\mathcal{ALC}$ ontologies.

The outcomes in terms of precision and recall were satisfactory. However, since these outcomes are not as meaningful as they might be in a standard (closed-world) setting, we recurred to different performance metrics, measuring the alignment of the classification decided by the concept descriptions induced by DL-Foil with the classification derived deductively by a DL reasoner. This allowed measuring the amount of unlabeled instances that may be ascribed to the newly induced concepts (or to their negations), which constituted a real added value brought by the inductive method. Actually these abductive conclusions should be evaluated by the expert who helped during the construction of the ontology which was not possible unless toy-ontologies were employed.

The experiments made on various ontologies showed that the method is quite effective, and its performance depends on the number (and distribution) of the available training instances. Besides, the procedure appears also robust to noise since commission errors were limited in the experiments carried out so far.

We plan to extend this work evaluating the benefits the algorithm can receive from the addition of other reasoning strategies such as abstraction and abduction. Another important issue is related to the employment of suitable distance or similarity measures that could influence chosen generalization strategy as well as example ordering in the training set. The measure may be applicable to other instance-based tasks which can be approached through machine learning techniques. Then the measure might be plugged in a hierarchical clustering algorithm where clusters would be formed grouping instances on the grounds of their similarity assessed through the measure.

References

[1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)

[2] Baader, F., Ganter, B., Sertkaya, B., Sattler, U.: Completing description logic knowledge bases using formal concept analysis. In: Veloso, M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, pp. 230–235 (2007)

[3] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)

[4] Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82(1–2), 353–367 (1996)

[5] Brandt, S., Küsters, R., Turhan, A.-Y.: Approximation and difference in description logics. In: Fensel, D., Giunchiglia, F., McGuinness, D., Williams, M.-A. (eds.) Proceedings of the International Conference on Knowledge Representation, pp. 203–214. Morgan Kaufmann, San Francisco (2002)

[6] Cohen, W.W., Hirsh, H.: Learnability of description logics. In: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Pittsburgh, PA. ACM Press, New York (1992)

[7] Cohen, W.W., Hirsh, H.: Learning the CLASSIC description logic. In: Torasso, P., Doyle, J., Sandewall, E. (eds.) Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning, pp. 121–133. Morgan Kaufmann, San Francisco (1994)

[8] d'Amato, C., Fanizzi, N., Esposito, F.: Query answering and ontology population: An inductive approach. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 288–302. Springer, Heidelberg (2008)

[9] Dean, M., Schreiber, G.: Web Ontology Language Reference. W3C recommendation, W3C (2004), http://www.w3.org/TR/owl-ref

[10] Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-intensive induction of terminologies from metadata. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004)

[11] Goldman, S.A., Kwek, S., Scott, S.D.: Learning from examples with unspecified attribute values. Information and Computation 180(2), 82–100 (2003)

[12] Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proceedings of the 12th international conference on World Wide Web, WWW 2003, pp. 48–57. ACM Press, New York (2003)

[13] Iannone, L., Palmisano, I., Fanizzi, N.: An algorithm based on counterfactuals for concept learning in the semantic web. Applied Intelligence 26(2), 139–159 (2007)

[14] Inuzuka, N., Kamo, M., Ishii, N., Seki, H., Itoh, H.: Tow-down induction of logic programs from incomplete samples. In: Muggleton, S. (ed.) ILP 1996. LNCS, vol. 1314, pp. 265–282. Springer, Heidelberg (1997)

[15] Kietz, J.-U.: Learnability of description logic programs. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 117–132. Springer, Heidelberg (2003)

[16] Kietz, J.-U., Morik, K.: A polynomial approach to the constructive induction of structural knowledge. Machine Learning 14(2), 193–218 (1994)

[17] Lehmann, J., Hitzler, P.: Foundations of refinement operators for description logics. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 161–174. Springer, Heidelberg (2008)

[18] Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 147–160. Springer, Heidelberg (2008)

[19] Lisi, F.A.: Principles of inductive reasoning on the Semantic Web: A framework for learning in $\mathcal{AL}$-Log. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 118–132. Springer, Heidelberg (2005)

[20] Quinlan, R.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)

[21] Rouveirol, C., Ventos, V.: Towards learning in CARIN-$\mathcal{ALN}$. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 191–208. Springer, Heidelberg (2000)

Feature Discovery with Type Extension Trees

Paolo Frasconi[2], Manfred Jaeger[1], and Andrea Passerini[2]

[1] Department for Computer Science, Aalborg University, Denmark
[2] Dipartimento di Sistemi e Informatica, Universitá degli Studi di Firenze, Italy

Abstract. We are interested in learning complex combinatorial features from relational data. We rely on an expressive and general representation language whose semantics allows us to express many features that have been used in different statistical relational learning settings. To avoid expensive exhaustive search over the space of relational features, we introduce a heuristic search algorithm guided by a generalized relational notion of information gain and a discriminant function. The algorithm succesfully finds interesting and interpretable features on artificial and real-world relational learning problems.

1 Introduction

A key component of relational data mining methods is the construction of relevant features. Whereas in conventional ("propositional") learning settings the set of possible features is usually given by the available attributes, one has in relational learning the ability to construct new features by considering the relational neighborhood of an entity. Taking into consideration related entities and their attributes, one obtains a large supply of potential features.

To illustrate the situation, consider Figure 1, which gives a graphical representation of an imaginary fragment of a movie database. Unfilled circles represent movies, shaded circles represent actors, arrows represent the binary relation *cast*, and stars identify movies that have won the best picture award. Suppose, now, that for movies m_1, m_2 it is not yet known whether they will win best picture award, and we want to predict this. For this prediction, awards won by movies with the same actors as m_1, m_2 may provide relevant information.

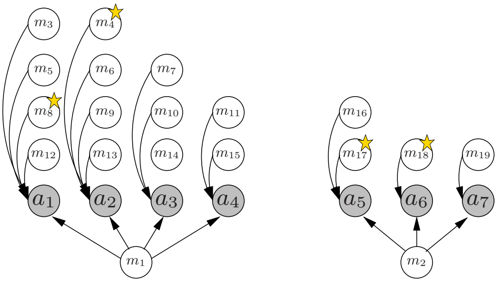


Fig. 1. Movie data fragment

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 122–139, 2008.

To express features in terms of properties of related objects, many different frameworks have been proposed. Some approaches are based on Boolean features corresponding to logical conditions like “there exists an actor in the cast who has also played in a movie that has won the best picture award.” Formal languages for expressing such features can be based on predicate logic (as in inductive logic programming), or use graphical representations [9].

Movies m_1, m_2 in Figure 1 can not be distinguished with such simple qualitative features, since both satisfy the same basic logical properties. Nevertheless, considering certain quantitative aspects of the relational neighborhoods of m_1, m_2, one might perhaps expect that m_2 has a higher chance of receiving an award than m_1: the two awards associated with m_1 come from cast members who have appeared in a relatively large number of movies, indicating that these are perhaps actors playing minor roles in many different movies, some of which happened to be successful. The awards associated with m_2, on the other hand, come from actors appearing in fewer movies, indicating that these could be major actors having leading roles in m_2.

One well established approach to express quantitative features is the use of *slot chains* and *aggregation*, which translate into database queries [5,13,14]. An example is the feature that counts the number of award-winning movies played by all cast members. This feature evaluates to 2 for both m_1 and m_2, and thus does not express the distinction described above. This distinction could be expressed by using proportion instead of count as a form of aggregation, yielding the feature that measures the proportion of award-winning movies among those played by cast members. It evaluates to 2/13 and 2/4 for m_1 and m_2, respectively, and thus makes the desired distinction. However, it is doubtful that this numeric feature captures exactly the most relevant quantitative information for making a prediction about m_1, m_2. Perhaps the count of actors with a proportion of $\geq 50\%$ award winning movies among the movies they participated in is the more relevant feature (here evaluating to 0 and 2, respectively, assuming that the movie to be classified is not considered in the count). Nested aggregates like this have been used in conjunction with first-order decision trees [1].

This very simple example illustrates three interesting aspects of relational features. First, the search space of possible features constructible by different combinations of aggregation, even for a single given slot chain, is quite big; second, each feature that we can construct in this way will probably only provide an approximation to the most relevant quantitative information for the prediction of the class label, and third, all these features are really summary statistics of a single, underlying more complex and fundamental feature: the complete specification of how many actors in the movie to be classified acted in how many other award winning and non award winning movies. The values of this *count of count* feature for m_1, m_2 are given by:

$$m_1 : \left\{ \begin{array}{l} \left\{ \begin{array}{r} a : 1 \\ \neg a : 3 \end{array} \right\} : 2 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 3 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 2 \end{array} \right\} : 1 \end{array} \right\} \quad m_2 : \left\{ \begin{array}{l} \left\{ \begin{array}{r} a : 1 \\ \neg a : 1 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 1 \\ \neg a : 0 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 1 \end{array} \right\} : 1 \end{array} \right\} \tag{1}$$

The value for m_1 shows that in the cast of m_1 there are two actors that acted in 1 award winning, and 3 non award winning movies. The formatting of the feature value emphasizes the hierarchical, combinatorial structure of these count of count values (which can also be nested deeper into count of count of count values, etc.). Note that count of count features in our sense are quite different from nested aggregates in the sense of [1]: unlike the latter, our counts of counts do not aggregate a multiset of values into a single number at each level of nesting.

In this paper, we study relational learning on the basis of complex count of count features. To this end we first review syntax and semantics of *Type extension trees (TETs)*, which are a representation language for such features first introduced in [7]. It has been shown in [7] that the TET language is very powerful, and many classes of features that have been used in relational learning can be derived from TET features.

However, the use of TET features in actual learning tasks has not been studied so far. In this paper, we introduce an algorithm for learning the structure of a TET from data in the supervised learning setting. A key component of this algorithm is a special kind of discriminant function that maps TET values into real numbers, and can be used as a simple but computationally efficient classifier. The TET learning algorithm uses heuristic search based on two types of evaluation heuristics: a generalized information gain measure for relational data, and a wrapper evaluation score based on empirical error of the discriminant function. Our experimental evaluation shows that we can learn TETs representing relevant and interpretable features.

2 TET Basics

In this section, we review the basic syntax and semantics definitions of TETs [7]. To simplify the definitions, we will assume fully Boolean domains, i.e. all attributes and relations are Boolean. All basic concepts can be quite easily generalized to multi-valued attributes and relations. In a Boolean domain, data can be viewed as a model in the sense of the following definition.

Definition 1. *Let R be a* relational signature *i.e. a set of relation symbols of different arities. A (finite) model for R, $\mathcal{M} = (M, I)$ consists of a finite domain M, and an interpretation function $I : r(\boldsymbol{a}) \rightarrow \{\text{true}, \text{false}\}$ defined for all* ground atoms $r(\boldsymbol{a})$ *constructible from relations $r \in R$ and arguments $\boldsymbol{a} \in M^{\text{arity}(r)}$.*

In logic programming terminology, $\mathcal{M}$ is a Herbrand interpretation for the vocabulary consisting of R and constant symbols for the elements of M. For convenience we may

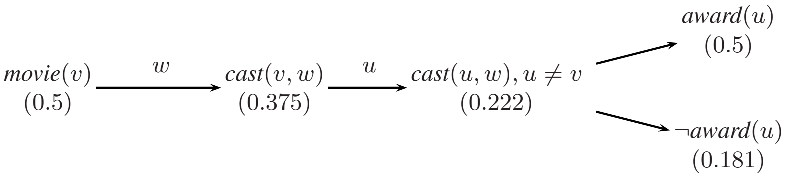


Fig. 2. A simple TET

assume that the domain M is partitioned into objects of different *types*, that arguments of relations are typed, and that I is only defined for ground atoms with arguments of appropriate types. We now proceed to define the formal syntax for expressing count of count features. The following definition is illustrated by Figure 2, which shows the TET defining the movie feature whose values for m_1, m_2 are shown in Eq. 1. The numbers in parantheses included in Figure 2 will be explained in Example 2 below.

Definition 2. *An* R-literal *is a (negated) atom* $r(\boldsymbol{v})$ *(*$r \in R \cup \{=\}$*,* $\boldsymbol{v}$ *a tuple of (possibly typed) variable symbols). An* R*-literal may not contain any constant symbols for elements* $a \in M$ *as arguments. We also allow the special literal* $\top(\boldsymbol{v})$*, which always evaluates to* true*. A* type *is a conjunction of* R*-literals.*

A type extension tree *(TET) over* R *is a tree whose nodes are labeled with* R*-types, and whose edges are labeled with (possibly empty) sets of variables.*

Note that the type of an object corresponds to a literal consisting of a single unary relation (e.g. *movie*(v)). Definition 2 generalizes this to types of tuples of objects, and to types which are expressed via a conjunction of literals. Labeled edges in a TET are related to quantifiers in predicate logic: like a quantifier, a labeled edge *binds* all occurrences of the variables associated with the edge in the subtree rooted at this edge. The *free variables* of a TET are all variables not bound by an edge label. The TET of Figure 2 has the single free variable v. We write $T(\boldsymbol{v})$ to denote a TET whose free variables are among the variables $\boldsymbol{v}$ (but does not necessarily contain all of them). We write

$$T(\boldsymbol{v}) = [\tau(\boldsymbol{v}), (\boldsymbol{w}_1, T_1(\boldsymbol{v}, \boldsymbol{w}_1)), \ldots, (\boldsymbol{w}_m, T_m(\boldsymbol{v}, \boldsymbol{w}_m))]$$

to denote a TET with a root labeled with $\tau(\boldsymbol{v})$, and m sub-trees $T_1(\boldsymbol{v}, \boldsymbol{w}_i)$ reached by edges labeled with variables $\boldsymbol{w}_i$ (possibly empty).

A TET $T(\boldsymbol{v})$ with free variables $\boldsymbol{v} = v_1, \ldots, v_k$ defines a *feature* for k-tuples of domain elements: for any model $\mathcal{M}$, and any $\boldsymbol{a} \in M^k$ the TET defines a *feature value* $V(T(\boldsymbol{a}))$. Eq 1 shows $V(T(m_1))$ and $V(T(m_2))$ for the TET T in Figure 2. We give the general definition of TET semantics in two steps: first we define the value space of nested counts associated with a given TET, and then the actual mapping $\boldsymbol{a} \mapsto V(T(\boldsymbol{a}))$.

Definition 3. *For any set* A *we denote with* multisets(A) *the set of all multisets over* A*. We denote with* $\{a_1 : k_1, \ldots, a_n : k_n\}$ *a multiset that contains* k_i *copies of* a_i*. The* pruned value space $\mathcal{V}(T)$ *of a TET T is inductively defined as follows:*

- *If* $T = [\tau]$ *consists of a single node, then* $\mathcal{V}(T) = \{\text{true}, \text{false}\}$.
- *If* $T = [\tau, (\boldsymbol{w}_1, T_1), \ldots, (\boldsymbol{w}_m, T_m)]$*, then*
$$\mathcal{V}(T) = \{\text{false}\} \cup \times_{i=1}^{m} \text{multisets}(\mathcal{V}(T_i)))$$

We call the $\mathcal{V}(T)$ of Definition 3 the pruned value space, because with these values we will only count objects that satisfy the conditions in the nodes. Full TET values, as originally defined in [7], treat *true* and *false* on an equal basis, and count both satisfying and non-satisfying objects.

Definition 4. *Let* $\mathcal{M} = (M, I)$ *be a model,* $T(v_1, \ldots, v_k)$ *a TET, and* $\boldsymbol{a} \in M^k$*. The value* $V(T(\boldsymbol{a})) \in \mathcal{V}(T)$ *is defined as follows:*

(i) If $T(\boldsymbol{v}) = [\tau(\boldsymbol{v})]$ *consists of a single node, then* $V(T(\boldsymbol{a})) := I(\tau(\boldsymbol{a}))$.

(ii) Otherwise:

(a) If $I(\tau(\boldsymbol{a})) = \text{false}$ *then* $V(T(\boldsymbol{a})) = \text{false}$.

(b) If $I(\tau(\boldsymbol{a})) = \text{true}$ *then*

$$V(T(\boldsymbol{a})) = (\mu(\boldsymbol{a}, \boldsymbol{w}_1, T_1), \ldots, \mu(\boldsymbol{a}, \boldsymbol{w}_m, T_m)),$$

with $\mu(\boldsymbol{a}, \boldsymbol{w}_i, T_i) \in \text{multisets}(\mathcal{V}(T_i))$ *given by*

$$\{\gamma :| \{\boldsymbol{b} \in M \mid V(T_i(\boldsymbol{a}, \boldsymbol{b})) = \gamma\} | \mid \gamma \in \mathcal{V}(T_i)\}.$$

Example 1. Eq. 1 shows $V(T(m_1))$ and $V(T(m_2))$. To explain the compuations of $V(T(m_1))$, we introduce abbreviations for sub-TETs of T:

$$\begin{aligned} T^{(1)}(u, w) &:= [\mathit{cast}(u, w), (\emptyset, [\mathit{award}(u)]), (\emptyset, [\neg\mathit{award}(u)])] \\ T^{(2)}(v, w) &:= [\mathit{cast}(v, w), (u, T^{(1)}(u, w))] \end{aligned}$$

Since $I(\mathit{movie}(m_1)) = \mathit{true}$, the computation of $V(T(m_1))$ follows case ii.b of Definition 4. That means we have to count for each $\gamma \in \mathcal{V}(T^{(2)})$ the number of actors a for which $V(T^{(2)}(m_1, a)) = \gamma$. We do this by computing for each of the four actors $a_1, \ldots, a_4$ the value $V(T^{(2)}(m_1, a_i))$. Consider a_1. Since $I(\mathit{cast}(m_1, a_1)) = \mathit{true}$, we again reach case ii.b, and now have to compute for each of the 13 movies $m' \neq m_1$ the value $V^{(1)}(m', a_1)$. For 9 out of the 13 movies this is just the *false* value (ii.a), because $I(\mathit{cast}(m', a_1)) = \mathit{false}$. For 3 movies (the ones without award), we obtain the value $(\{\mathit{false} : 1\}, \{\mathit{true} : 1\})$ (the pair of values $(V([\mathit{award}(m')]), V([\neg\mathit{award}(m')])))$, and for the award winning movie the value $(\{\mathit{true} : 1\}, \{\mathit{false} : 1\})$. Collecting these counts, we get

$$V(T^{(2)}(m_1, a_1)) = \left\{ \begin{array}{r} \mathit{false} : 9 \\ (\{\mathit{false} : 1\}, \{\mathit{true} : 1\}) : 3 \\ (\{\mathit{true} : 1\}, \{\mathit{false} : 1\}) : 1 \end{array} \right\}$$

This has been simplified to $\{a : 1, \neg a : 3\}$ in Equation 1. The same value is obtained for $V(T^{(2)}(m_1, a_2))$, giving a count of 2 for this value in $V(T(m_1))$.

A TET $T(\boldsymbol{v})$ represents a feature of object tuples $\boldsymbol{a}$. Several features might be expressed by TETs $T_1(\boldsymbol{v}), \ldots, T_m(\boldsymbol{v})$. Such a collection of features can always be combined into the single TET $[\top(\boldsymbol{v}), (\emptyset, T_1(\boldsymbol{v})), \ldots (\emptyset, T_m(\boldsymbol{v}))]$, which exactly provides the combined information of the T_i. In the following we will therefore focus on the scenario where a single TET represents all relevant features.

3 TET Discriminant Function

A TET alone only defines a feature of objects in a relational domain. TET-defined features can be incorporated in many ways into existing types of predictive or descriptive models. For example, one can define distance or kernel functions on TET value spaces $\mathcal{V}(T)$, thereby making TET features useable for standard clustering techniques, or SVM classifiers. In this section we briefly describe how to build a predictive model on a TET feature using simple *discriminant functions* on TET values, i.e. functions of the form

$$d : \mathcal{V}(T) \rightarrow \mathbb{R}.$$

To obtain a binary classification model (throughout we assume that class labels are $\{+,-\}$), we learn two discriminant functions d_+, d_-, and a threshold value t, and assign class label $+$ to a tuple $\boldsymbol{a}$ iff

$$d_+(V(T(\boldsymbol{a}))/d_-(V(T(\boldsymbol{a})) > t. \qquad (2)$$

We now introduce one simple type of TET-discriminant function. The motivation for the particular form of discriminant function we propose is twofold: first, for a given TET, our functions are efficient to learn and evaluate. Since we use the discriminant function in TET learning within a wrapper evaluation routine for candidate TETs, efficiency is an important issue. Second, in spite of their simplicity, one can show that "propositional TETs" (TETs with only unary relations and no edge labels) together with these discriminant functions provide a uniform generalization of the classic decision tree and Naive Bayes models. Due to space limitations we have to defer a full discussion of the relationship between our discriminant functions and other predictive models to a full paper.

Definition 5. *Let T be a TET. A* weight assignment $\boldsymbol{\beta}$ *for T assigns a nonnegative real number to all nodes of T. A weight assignment can be written as $(\beta^r, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m)$, where β^r is the weight assigned to the root, and $\boldsymbol{\beta}_i$ is the weight assignment to the ith sub-tree.*

For a TET T with node weights $\boldsymbol{\beta}$ we define the discriminant function $d^{\boldsymbol{\beta}}$ as follows. Let $\gamma \in \mathcal{V}(T)$:

- *If $\gamma = \text{false}$ define $d^{\boldsymbol{\beta}}(\gamma) := 0$.*
- *If $\gamma = \text{true}$ then $T = [\tau(\boldsymbol{v})]$ consists of a single node, and $\boldsymbol{\beta} = (\beta^r)$. Define $d^{\boldsymbol{\beta}}(\gamma) := \beta^r$.*
- *If $\gamma = (\mu_1, \ldots, \mu_m)$, $\mu_i \in \text{multisets}(\mathcal{V}(T_i))$, define*

$$d^{\boldsymbol{\beta}}(\gamma) := \beta^r \cdot \prod_{i=1}^{m} \prod_{\gamma' \in \mu_i, \gamma' \neq \text{false}} \frac{1}{\beta^r} d^{\boldsymbol{\beta}_i}(\gamma').$$

Example 2. The numbers shown in Figure 2 are a weight assignment. These are the actual weights defining the discriminant function d_+ we would learn using the method described in Section 4 for this TET, assuming that m_1 and m_2 of Figure 1 are given as a negative and positive example, respectively.

Now consider the value $\gamma_1 = V(T(m_1))$ as shown on the left in Eq. 1 and explained in Example 1. The (simplified) inner values $\{a : j, \neg a : k\} \in \mathcal{V}(T^{(2)})$ have the discriminant value

$$\begin{aligned} d(a : j, \neg a : k) &= 0.375 \cdot \left(\tfrac{0.222}{0.375} \tfrac{0.5}{0.222}\right)^j \cdot \left(\tfrac{0.222}{0.375} \tfrac{0.181}{0.222}\right)^k \\ &= 0.375 \cdot \left(\tfrac{0.5}{0.375}\right)^j \cdot \left(\tfrac{0.181}{0.375}\right)^k . \end{aligned}$$

This leads to

$$\begin{aligned} d(\gamma_1) &= 0.5 \cdot \left(\frac{d(a{:}1, \neg a{:}3))}{0.5}\right)^2 \left(\frac{d(a{:}0, \neg a{:}3))}{0.5}\right) \left(\frac{d((a{:}0, \neg a{:}2))}{0.5}\right) \\ &= 0.5 \cdot \left(\tfrac{0.375}{0.5}\right)^4 \cdot \left(\tfrac{0.5}{0.375}\right)^2 \cdot \left(\tfrac{0.181}{0.375}\right)^{11} . \end{aligned}$$

Similarly for $\gamma_2 = V(T(m_2))$:

$$d(\gamma_2) = 0.5 \cdot \left(\frac{0.375}{0.5}\right)^3 \cdot \left(\frac{0.5}{0.375}\right)^2 \cdot \left(\frac{0.181}{0.375}\right)^2 .$$

Intuitively, we can interpret this discriminant function as follows: because in the training data the positive example had a smaller cast than the negative example, the discriminant function multiplies the marginal probability of the positive class (0.5) for each actor in the cast with a penalty term $\frac{0.375}{0.5} < 1$. Furthermore, since in the negative example the actors in the cast appeared in a larger number of movies than in the positive example, another penalty factor $\frac{0.222}{0.375} < 1$ is added for every associated movie of any actor in the cast. Finally, a factor of $\frac{0.5}{0.222} > 1$ is added for every associated award movie, and a factor $\frac{0.181}{0.222} < 1$ for every non-award movie (cancelling out the 0.222 terms).

4 TET Learning

We first describe in more detail the learning problem we want to solve. Our data consists of a model $\mathcal{M}$ in the sense of Definition 1. In our implementation, $\mathcal{M}$ is given as a relational database containing one table for each $r \in R$, where the table for r contains all tupels $\boldsymbol{a} \in M^{\mathit{arity}(r)}$ for which $I(r(\boldsymbol{a})) =$ *true*. Furthermore, we are given an initial *target table*, i.e. a table consisting of a set of examples with $+/-$ class labels. For example, a learning problem given by the data depicted in Figure 1, and m_1, m_2 as a negative and positive example, respectively, would be given by the 4 leftmost tables in Table 4. Columns in the data tables are headed by synthetic identifiers *Arg*$_i$. Columns in the target table (other than the class label column) are headed by variable names, which will then become the names of the free variables in the TET we construct. Thus, given the input we will want to construct a TET $T(v)$ over the signature $R = \{\mathit{movie}, \mathit{cast}, \mathit{award}\}$ that captures features of v that are predictive for the class label given in *target*.

Note that in this representation of the learning task it is not visible that the class we want to predict is actually the *award* relation. Such a representation is appropriate, for example, when we have temporally stratified data, and the task is to predict for current movies m_1, m_2 whether they will receive an award – a prediction for which we may use the award label of older movies $m_3, \ldots, m_{19}$ (this is the scenario we have in our IMDB

Table 1. Data and Target Tables

Arg$_1$
m_1
m_2
⋮
m_{19}

movie

Arg$_1$	Arg$_2$
m_1	a_1
m_1	a_2
⋮	⋮
m_{19}	a_7

cast

Arg$_1$
m_4
m_8
m_{17}
m_{18}

award

v	Label
m_1	$-$
m_2	$+$

target

Input data

v	w	Label
m_1	a_1	$-$
⋮	⋮	⋮
m_1	a_4	$-$
m_2	a_5	$+$
m_2	a_6	$+$
m_2	a_7	$+$

target

Local target

experiment, Section 5). For prediction problems without such a stratification, notably when for the prediction of any object's class label one may use the class label of all other objects in the domain, the target table will essentially duplicate the class table (in our example, the award table would also contain m_2, and the target table would contain all movies, correctly labeled + or $-$). In such a case suitable syntactic constraints can be imposed in the TET construction that prevent the construction of "illegal" features ($\mathit{award}(v)$ as a feature for predicting the class of v).

Our general approach to TET learning is a recursive top-down construction that associates with each node a local discrimination task represented by a local target table. This makes our approach somewhat related to decision tree learning. In a fundamental difference to decision tree learning, however, it will generally not be the case that the target tables of child nodes partition the target table of their parent. In our running example, if starting with the input data in Table 4 we initially constructed the extension $\mathit{movie}(v) \xrightarrow{w} \mathit{cast}(v, w)$, then we would associate with the node $\mathit{cast}(v, w)$ the local target table shown on the right of Table 4. Rather than a reduction of the discrimination task to a smaller set of examples, as in decision tree learning, the construction of this new target table amounts to a problem transformation: the problem of predicting the label of a movie is transformed into predicting the label of movie/actor pairs in the new target table, which may be effected by taking into consideration attributes of both movies and actors, as well as additional relations between actors and movies (if any such exist in the data).

The exact specification of the construction of local target tables is as follows. Let n be a TET node associated with a local target table $tt_n(\boldsymbol{v}, L)$ with columns for variables $\boldsymbol{v}$ and label L. Let n' be a child of n labeled with type $\sigma(\boldsymbol{v}, \boldsymbol{w})$, and reached by an edge labeled with variables $\boldsymbol{w}$ (we include the possibility that $\boldsymbol{w}$ is the empty tuple, i.e. the edge is really unlabeled). Then n' is associated with a target table $tt_{n'}$ with columns for $\boldsymbol{v}, \boldsymbol{w}$ and L, defined as:

$$tt_{n'}(\boldsymbol{v}, \boldsymbol{w}, L) = \{(\boldsymbol{a}, \boldsymbol{b}, l) \in M^{|\boldsymbol{v}|+|\boldsymbol{w}|} \times \{+, -\} : (\boldsymbol{a}, l) \in tt_n; \mathcal{M} \models \sigma(\boldsymbol{a}, \boldsymbol{b})\}. \quad (3)$$

When building the TET we score candidate extensions based on a *generalized information gain (gig)* measure, which is a function of the old and new target table. The *gig* measure plays a central role in our TET learning method, and is more fully discussed in Section 4.1 below. In addition to *gig* scoring of candidate extensions (which is a local score for one node and a child), our learning method is driven by a global evaluation of the current full TET structure based on the accuracy of this TET in conjunction with the chosen classification model. In our implementation this is the discriminant function model. Using a different type of TET-based classifier (e.g. an SVM based on a TET-kernel) could obviously lead to different results in the TET learning. Thus, the TET learner does not perform "pure" feature discovery; rather, it constructs features that are useful in combination with a discriminant function classifier. However, our experimental results (Section 5) indicate that the features we obtain are not highly biased towards this particular classification model.

Table 2 shows the general structure of our TET learning method. The method is essentially implemented as a node constructor, that receives as arguments the data, a target table tt to be classified, a pointer r to the root of the TET in which this node is

Table 2. TET learning

```
$TET\_node(Data\,\mathcal{M}, Labeled\ table\ tt, TET\_node\ r, Type\ \tau(\boldsymbol{v}))$
1. this.type = $\tau(\boldsymbol{v})$
2. this.weight = $positive\_class\_frequency(tt)$
3. $TET\_node\ root$
4. if ($r$=$null$) $root$=this else $root$=$r$
5. $current\_score = predictive\_score(root)$
6. $EXT$:=$possible\_extensions$($root$,this)
7. for all $\sigma(\boldsymbol{v},\boldsymbol{w}) \in EXT$ compute $gig(tt,\sigma(\boldsymbol{v},\boldsymbol{w}))$
8. $CAND$:= $candidate\_extensions$($EXT$,$gig$-values)
9. for all $\sigma(\boldsymbol{v},\boldsymbol{w}) \in CAND$
10.    $tt' = construct\_tt(\mathcal{M}, tt, \sigma(\boldsymbol{v},\boldsymbol{w}))$
11.    add as child $c = TET\_node(\mathcal{M}, tt', root, \sigma(\boldsymbol{v},\boldsymbol{w}))$
12.    $new\_score = predictive\_score(root)$
13.    if $new\_score - current\_score < threshold$
14.      delete $c$
15.    else $current\_score$=$new\_score$
```

constructed, and the type $\tau(\boldsymbol{v})$ with which the node is to be labeled. A TET is learned by the call

$$root\_of\_learned\_TET = TET\_node(\mathcal{M}, tt, null, \top(\boldsymbol{v}))$$

(because of initialization issues, all our TETs have the vacuous type $\top(\boldsymbol{v})$ in the root).

The construction works as follows: Line 1 labels the node under construction with its type. Line 2 sets the weight for the discriminant function d_+ for this node. It is just the relative frequency of positive examples in the target table associated with this node (the weight for d_- being one minus this weight). Thus, the discriminant function here is learned (at little extra cost) in parallel with the TET construction. If a different classification model was used for the TET construction, then line 2 would be omitted. Lines 3-4 set a local *root* variable pointing to the root of the TET under construction.

The function *predictive_score* called in lines 5 and 12 performs the global evaluation of the current TET based on its predictive performance in conjunction with the chosen classification model. If a model other than the discriminant function here is used, then calls to *predictive_score* may require computationally expensive model training for the current TET.

Lines 6-8 are crucial: here a subset of all the possible extensions of the current node is constructed for further exploration. This operation is analogous to refinement operators in ILP. Our construction is in two steps: in the first step the set of possible extensions for the current node is constructed by the function *possible_extensions*. This function can implement various constraints and a language bias. In our implementation, we restrict possible extensions to contain only one literal, and to introduce at most one new variable. The function can also take TILDE-style user-defined *rmode* declarations, that can force certain arguments of the new literal to be filled with variables already present in the parent node (input variable), or with a new variable introduced by this extension (output variable). In all our experiments we use rmode declarations that force the argument of any unary relation to be an input variable. Furthermore, *possible_extensions* is used to implement a termination condition: if the depth of the current node in the TET has reached a (user specified) maximum depth, then *possible_extensions* will return an empty set. In a second step, the generalized information gain is computed for all possible extensions; the function *candidate_extensions* then performs a selection based on *gig* values. Our current implementation of *candidate_extensions* selects all extensions whose *gig* value exceeds user defined thresholds (we use different thresholds depending on whether a candidate extension introduces a new variable; the *gig* values for these two types of extensions are incomparable, see Section 4.1).

For each candidate extension then a child node is created. The function *construct_tt* constructs the local target table for the child according to (3). After line 11 is executed, a whole new subtree is rooted at the new node c. Lines 12-15 then evaluate the extension of the old TET with this new subtree, and either accept or reject the subtree based on a user defined threshold for the required global score improvement.

4.1 Generalized Information Gain

To select promising candidate extensions in lines 7-8 of the TET learner, we use a *generalized information gain* measure that we now introduce. Given is a TET node labeled with a target table $tt(\boldsymbol{v}, L)$. We distinguish extensions with unlabeled and labeled edges, which we also call *condition introductions* and *object introductions*, respectively.

In a condition introduction we add a child node labeled with a type $\sigma(\boldsymbol{v})$ that puts additional constraints on the tuples in $tt(\boldsymbol{v}, L)$, and the new target table obtained from *tt* and $\sigma(\boldsymbol{v})$ via (3) is just a subset of *tt*. We denote this sub-table $tt_{\sigma(\boldsymbol{v})}$. Let $H(tt)$ denote the entropy of the class label distribution in *tt*. Standard information gain

$$ig(\sigma(\boldsymbol{v})) := H(tt) - \frac{|tt_{\sigma(\boldsymbol{v})}|}{|tt|} H(tt_{\sigma(\boldsymbol{v})}) - \frac{|tt \setminus tt_{\sigma(\boldsymbol{v})}|}{|tt|} H(tt \setminus tt_{\sigma(\boldsymbol{v})}) \qquad (4)$$

can therefore be used to measure the informativeness of a candidate condition introduction.

When evaluating candidate object introductions, we have to consider two different aspects: first, extending a node in the current TET by an object introduction refines the feature represented by the TET and modifies the discriminant function, which may directly lead to improved predictive performance. In this case we say that the object introduction is *directly informative*. Secondly, an object introduction can be *potentially informative* when the problem transformation represented by the new target table enables additional strategies for discriminating between positive and negative examples.

To illustrate these two aspects, consider the candidate extension $\xrightarrow{w}$ *cast*(v, w) of the root $\top(v)$ for the data in Table 4. This extension introduces the number of actors in a movie as a feature. Since in the data movies in the positive class have fewer actors than movies in the negative class (i.e. there is *degree disparity* [8]), this extension is directly informative. Note that due to the fact that every movie has at least one actor, *cast*(v, w) is a *non-discriminating relation* [4] for standard ILP systems without aggregation functions. The extension also is potentially informative, because, as the new local target table indicates, movies in the positive class have different actors in their cast than movies in the negative class, and therefore the classes can also be distinguished if we succeed in separating the two sets of actors by subsequent extensions of the new *cast*(v, w) node.

Our goal is to find an evaluation measure for both the direct and potential informativeness of a candidate object introduction $\xrightarrow{\boldsymbol{w}} \sigma(\boldsymbol{v}, \boldsymbol{w})$. Measuring the potential informativeness serves similar goals as *lookahead* [2,4] in ILP systems, but differs in that it tries to achieve this goal by considering only one-step extensions (i.e. single literals in the context of our learner), rather than the combinatorial search space of possible multi-step refinements.

Consider a specific tuple of objects $\boldsymbol{a}$ that can be substituted for the variables $\boldsymbol{w}$ in $\sigma(\boldsymbol{v}, \boldsymbol{w})$. Then we can consider $\sigma(\boldsymbol{v}, \boldsymbol{a})$ as a condition introduction (it imposes on the tuples in $tt(\boldsymbol{v}, L)$ the condition of being related to the fixed objects $\boldsymbol{a}$ in the way specified by $\sigma(\boldsymbol{v}, \boldsymbol{a})$), and we can measure $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$ according to (4). If objects $\boldsymbol{a}$ are either mostly associated with $\boldsymbol{v}$'s from the positive class, or with $\boldsymbol{v}$'s from the negative class, then $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$ will tend to be larger than when no such association exists. If this association exists for a significant proportion of tuples $\boldsymbol{a}$, then this will be reflected in the generalized information gain defined as

$$gig(\sigma(\boldsymbol{v}, \boldsymbol{w})) := \frac{1}{|\,bindings(\boldsymbol{w})\,|} \sum_{\boldsymbol{a} \in bindings(\boldsymbol{w})} ig(\sigma(\boldsymbol{v}, \boldsymbol{a})),$$

where $bindings(\boldsymbol{w})$ is the set of tuples of objects that can be substituted for $\boldsymbol{w}$. The values one obtains for $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ must be interpreted with some care: in most situations $|\,tt_{\sigma(\boldsymbol{v},\boldsymbol{a})}\,|/|\,tt\,|$ will be small for all $\boldsymbol{a}$, and therefore also $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ will be a small number. It therefore does not make sense to compare $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ against the information gain $ig(\sigma(\boldsymbol{v}))$ of a simple condition introduction. However, $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ provides a meaningful measure to compare several candidate object introductions against each other.[1]

So far, we have motivated *gig* by its ability to measure the potential informativeness of an object introduction. However, direct informativeness from degree disparity, too, will increase *gig*: when $\boldsymbol{v}$'s in the positive class, on average, have more associated $\boldsymbol{w}$'s than $\boldsymbol{v}$'s in the negative class, then, conversely, for many $\boldsymbol{a}$'s in $bindings(\boldsymbol{w})$ the distribution of $\boldsymbol{v}$'s within $tt_{\sigma(\boldsymbol{v},\boldsymbol{a})}$ must show a higher proportion of the positive class than in the base table tt. This leads to higher values of $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$, for many $\boldsymbol{a}$'s, and thus a higher value of $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$.

5 Experiments

Our experiments focus on the capability of TET learning as a feature discovery method. We are interested in whether the TET learner will produce known relevant and/or new useful and interpretable features. We also investigate the predictive performance of the learned TETs in conjunction with the discriminant function classifier. We compare the TET learner with a range of other relational learning systems: *relational probability trees* [11] as implemented in the *Proximity* system (kdl.cs.umass.edu/proximity),

[1] *gig* in Table 2 refers to either *gig* when object introductions are scored, or standard *ig* for scoring condition introductions.

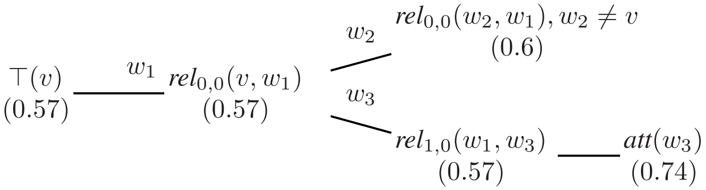
TILDE [3] (a relational decision tree learner driven by traditional info-gain), and the *Alchemy* system (alchemy.cs.washington.edu) for Markov Logic Networks [15].

Slotchain data (synthetic). An important type of relational features consists of attribute values of other objects that are reachable via a certain sequence of relations, also called *slotchain*. The challenge posed by slotchain features lies in the fact that they consist of a conjunction of literals, none of which is informative individually. Thus, slotchain features are useful prototype features for testing the effectiveness of tools like lookahead or our generalized information gain.

We generated data representing slotchain dependencies as follows: the domain consists of 300 objects each of 3 different types $\mathit{type}_0, \ldots, \mathit{type}_2$. Between objects of type i and $i+1$ ($i = 0, 1$) 2 different binary relations $\mathit{rel}_{i,k}$ ($k = 0, 1$) are generated by randomly selecting for each object o of type i exactly 2 distinct objects as $\mathit{rel}_{i,k}$-successors. Objects of type type_2 have a boolean attribute *att*. Objects of type type_0 have a class label *class*. The probability that $o \in \mathit{type}_0$ is in the positive class is 0.8 if o is connected via the chain of relations $\mathit{rel}_{0,0}, \mathit{rel}_{1,0}$ to an object $o' \in \mathit{type}_2$ with $\mathit{att}(o) = \mathit{true}$; and 0.1 otherwise. The *att* relation is true for $o' \in \mathit{type}_2$ with probability 0.2. The relevant feature for predicting the class label therefore is

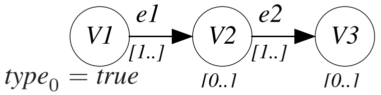
$$\top(v) \overset{w_0}{\text{---}} \mathit{rel}_{0,0}(v, w_0) \overset{w_1}{\text{---}} \mathit{rel}_{1,0}(w_0, w_1) \text{---}\, \mathit{att}(w_1)$$

The TET learner returned the weighted TET


$\top(v)$
(0.57)
w_1
$\mathit{rel}_{0,0}(v, w_1)$
(0.57)
w_2
$\mathit{rel}_{0,0}(w_2, w_1), w_2 \neq v$
(0.6)
w_3
$\mathit{rel}_{1,0}(w_1, w_3)$
(0.57)
$\mathit{att}(w_3)$
(0.74)

While this TET contains a spurious branch apart from the correct feature, this branch does not contribute much to the discriminant function, because the weight of its root is very close to the weight of the root's parent. This weighted TET achieves a predictive accuracy of 0.86 and AUC(ROC) value of 0.87 on a test set.

A comparison with the *Proximity* implementation of relational probability trees (Prox-RPTs) is made difficult by the fact that Proximity does not operate directly on the relational database, but on set of examples consisting of small subgraphs of the full data graph. These subgraphs have to be constructed by a user-defined query, the formulation of which already amounts to a large part of the feature discovery process. Queries are formulated in a language of graph patterns. To retrieve suitable subgraphs for the slotchain learning task, we used the query

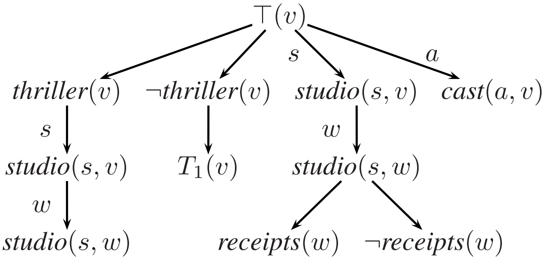


This query retrieves for every object $o \in \mathit{type}_0$ the subgraph consisting of all nodes and edges that are at most two steps away from o. The feature language employed by Prox-RPTs uses count and aggregation operators for attributes of node and edge elements in the query graph. The RPT we obtained had in its root the feature $\mathit{prop}(\mathit{V3.att} = \mathit{true}) \geq 0.875$, and similar features referring to *V3.att* in its other nodes. Thus, while the Prox-RPT learner determined the relevance of the *att* attribute in nodes two steps away, it did not refine this in terms of the exact slotchain by which these nodes are reached. Accuracy (0.65) and AUC(ROC) (0.633), therefore, were much lower than in the TET-based model.

TILDE without lookahead, as expected, was unable to learn the slotchain (it returns the empty clause). Only when the correct slotchain is given as a user-defined lookahead clause, can this feature be learned.

Alchemy structure learning did not terminate on the slotchain data within 6 hours (whereas the TET learner took about 30 seconds, and Proximity less than 10 seconds). However, Alchemy tries to solve the much harder task of learning a full generative model for all relations in the data, rather than a predictive model for the *class* attribute.

IMDB. Our second experiment uses the real world Internet Movie Database (IMDb). In our experimental setup we follow [10]. The problem is to predict whether a movie will earn more than $2 million in its opening weekend (expressed by a *receipts* class label). Available data includes attributes of movies, e.g. a *genre* attribute, binary relations between movies and actors, movies and producing studios, and movies and directors, as well as an *award* attribute of actors. Movie data from 5 successive years 1996-2000 was used, and the 4 separate prediction problems for the class labels of movies from the years 1997,...,2000 were considered. When classifying movies from a given year, then the *receipts* label of movies from the preceding year is available. In all 4 versions of the experiment we obtained quite similar TETs, a representative one being



Because of space limitations, the subtree $T_1(v)$ here is not shown. It consists of further condition introduction nodes (10 in total) testing different genre attributes of movies. The most interesting feature encoded by this TET (which is present in all 4 versions of the experiment) is the third branch from the left, which counts the *receipt* attribute values among the movies produced (in previous years) by the same studio as the movie to be classified. Interestingly, this is essentially the underlying count statistic of the feature that Neville and Jensen [10] obtain at the root of a relational probability tree they construct for the same classification problem:

$$\text{Studio Movie } \mathit{Prop}(\mathit{receipts} = y) > 0.6$$

The TETs we construct, together with the simple discriminant function as the classification model, yields accuracy results (84.4, 78.8, 80.7, and 86.4%, respectively, for the 4 different years) which are competitive with the results reported by Neville and Jensen.

CORA. Our third experiment uses the CORA citation data. We consider the entity resolution problem as investigated by Singla and Domingos [15]; we use the dataset available at alchemy.cs.washington.edu. The predictive tasks are to decide whether two strings found in the *author*, respectively *venue* field in two different bibliographic entries refer to the same entity, and whether two different bibliographic entries refer to the same paper. Types of objects in the database include *author*, *venue*, *bib*, *word*,... Available relations include the binary relations *hasWordVenue*(v, w) representing that venue v contains the word w, *bibToVenue*(b, v) representing that bibliographic item b contains v in the venue field, and similar relations for the fields 'author' and 'title'.

We ran the TET learner on each of the 5 different training sets defined by a 5-fold division of the source data. In all 3 tasks the TETs constructed over the different folds exhibited a quite stable structure, with some branches being present in all folds. A representative TET for the *venue* resolution task is shown at the left in Figure 3.

The left branch (which was constructed in all 5 runs) represents a central feature for entity resolution: whether two entities (i.e. strings) are the same depends on the number of words that appear in both of them. This simple fact is called *reverse predicate equivalence* in [15], and hand-coded into the prediction model via logical axioms of the form

$$\mathit{hWV}(v_1, w) \wedge \mathit{hWV}(v_2, w) => \mathit{sameVenue}(v_1, v_2)$$

The right branch first introduces the bib-entries that have the string v_1 in the venue field). The left sub-branch tests whether b_1 also contains v_2 in the venue field. In effect, this is a test whether v_1 and v_2 are identical strings – an obvious feature to consider, yet one that is not expressible in a more straightforward manner with the given relational vocabulary! The right sub-branch introduces the title of the bib entry b_1, and then counts the number of bib-entries b_2 that also have this title, and that contain venue v_2 in the venue field. In effect, this whole branch represents a feature that considers how often v_1 and v_2 appear in bib-entries b_1, b_2 that have the same title.

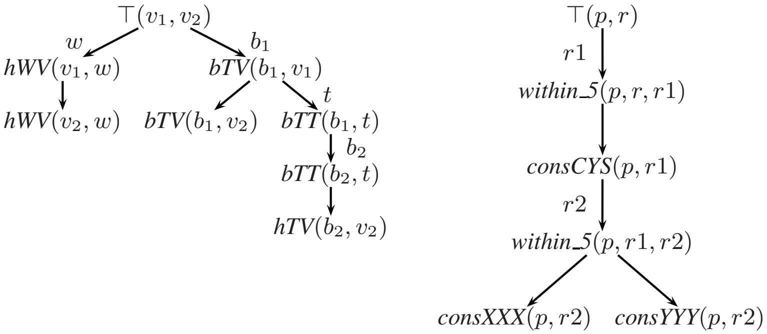


Fig. 3. TETs for Venue resolution (left) and metal binding sites (right)

For the three prediction tasks, author resolution, venue resolution, and bib-entry resolution, we achieved with our weighted TETs AUC(PR) values of 0.987, 0.771, 0.938, respectively (micro-averaged over the 5 folds). This is slightly better for author and venue than the values 0.980, 0.743, 0.971 reported in [15] for a Markov Logic Network model that (like our TET) is language independent, i.e. does not contain rules referring to specific strings occurring in the data. The worse performance on Bib can probably be explained by the fact that [15] perform multi-task classification, and that Bib resolution benefits more from the multi-task setting than Author and Venue (there is no direct evidence for two Bibs being the same: they tend to be the same if their authors and venues are the same).

Metal binding sites. In our last experiments we predict metal binding residues in proteins, using sequence information only. Metal ions are essential for Life, performing structural, catalytical and regulatory roles in the cell. About one third of the known proteins is believed to bind metal ions in their native conformation. A single metal ion is typically coordinated by a number of residues ranging from one to eight, each metal having a preferred coordination number and a range of possible alternatives. The most common metal ligands are CYS and HIS, followed by ASP and GLU which are however far more abundant in proteins. Metal binding sites typically show many regularities in terms of number, type, and distance of ligands and surrounding residues. Biologists have tried to encode such regularities in motifs identifying specific metal binding sites, as well as other biologically relevant portions of proteins. PROSITE [6] motifs consist of either regular expressions or position-specific profiles with amino acid weights and ggvgap costs. The former are extremely specific but have a very low recall, the latter show a more balanced precision/recall ratio but their performance is still rather unsatisfactory. While providing interpretable explanations, such motifs fail to capture much of the information provided by the sequence, especially when enriched by evolutionary information in the form of multiple alignment profiles, as shown by the performance achieved by a supervised learning architecture [12] fed by such inputs. Our TET learning algorithm discovered relevant and interpretable combinatorial features that outperform the knowledge-based regular expressions defined in PROSITE. Furthermore, counts-of-counts capabilities indeed provide additional discriminative power, as shown by experimental comparisons with Tilde and with TETs based on regular expression-like features only.

We learned two separate TETs for CYS and HIS respectively. Objects in the domain consist of proteins and residues within proteins. Residue attributes are extracted from multiple alignment profiles, and consist of the binarized conservation of either relevant residue types such as CYS, HIS, ASP, GLU or PRO, or relevant residue classes such as *small*, *hydrophobic* or *negative*. Relations have the form *Within_n(p,r1,r2)* and *Plus_n(p,r1,r2)*, and represent pairs of residues *(r1,r2)* in a certain protein *p* whose sequence separation is at most or exactly *n*, respectively. Note that despite their similarity, these two types of relations express quite different features: once one of the two residues is given, *Plus_n(p,r1,r2)* is satisfied by at most one other residue, and will thus lead to regular expression-like features similar to PROSITE patterns (but typically less specific). On the other hand, *Within_n(p,r1,r2)* allows to collect all residues in the neighbourhood of a given residue, and naturally generates counts-of-counts types of features.

Table 3. Microaveraged area under the ROC curve (%) for CYS and HIS metal bonding state prediction

Task	TET$_{full}$	TET$_{regexp}$	PROSITE$_{motifs}$	PROSITE$_{patterns}$	TILDE	SVM-BRNN
CYS	85.6±0.8	81.6±0.9	76.5±0.9	64.4±1.0	83.8±0.8	93.2±0.6
HIS	80.1±1.0	77.7±1.1	67.3±1.2	60.5±1.2	71.1±1.1	88.6±0.8

We ran a five-fold cross validation procedure on the same folds employed in [12]. Table 3 reports microaveraged area under the ROC results for TET with (TET$_{full}$) and without (TET$_{regexp}$) the *Within_n(p,r1,r2)* relation, PROSITE patterns and motifs and TILDE, as well as the state-of-the-art performance for this task [12]. Figure 4 reports significance of performance difference between learners, computed by a two-tailed Hanley-McNeil test [16] on areas under the ROC curve. TILDE was run with the same declarative bias used for learning TETs (rmode declarations that set one of the residues in *Plus_n* and *Within_n* relations to input, and the other to output). However, it was unable to exploit the *Within_n(p,r1,r2)* relations, which are *non-discriminating relations* [4] as any residue has at least one neighbour, while *Plus_n(p,r1,r2)* relations happen to be slightly discriminant as residues at sequence boundaries do not satisfy the relation and are typically non-binding residues. Forcing TILDE to explore *Within_n(p,r1,r2)* relations by extensive lookahead produced worse results and increased computational costs. Area under the ROC results for TILDE were calculated assigning to each example the probability of binding of the leaf node which classified it.

The best reported performance for this task (93.2% and 88.6% AUC for CYS and HYS, respectively) was previously obtained with SVM-BRNN, a highly engineered architecture based on support vector machines and bidirectional recurrent neural networks, fed by amino acid windows [12]. However, predictions obtained by such an architecture are not interpretable. While the TET-based classifier does not reach the same levels of accuracy, it significantly outperforms annotations curated by experts and effectively suggests novel and more compact explanations: TETs produced by our learning algorithm had 22 nodes on average, while there are 467 PROSITE patterns and 305

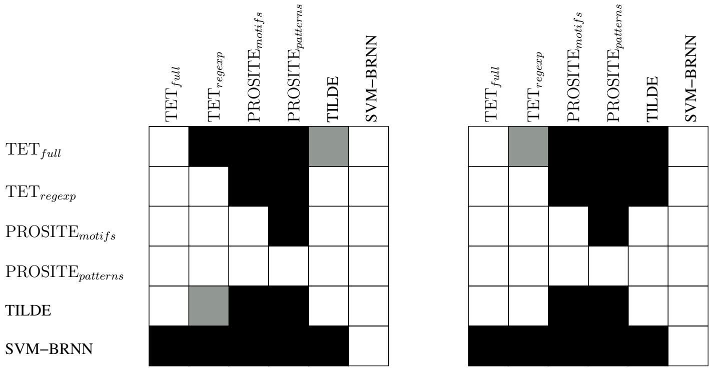


Fig. 4. Significance of performance difference between learners for the MBS data set. A black (grey) box indicates that the learner on the row is significantly better than that on the column at p=0.05 (p=0.1).

position-specific profiles that match at least one residue in the dataset. Counts-of-counts features do contribute to the overall TETs performance, as shown by the improvements over TILDE and TETs which rely on regular expression-like features only.

Figure 3 (right) shows a TET structure learned for CYS. Inspecting the structure, we note that the first branch learned in all folds encodes counts-of-counts features exploring the surrounding of the candidate residue, where XXX and YYY can be: HIS or *negative* identifying candidate co-ligands (ASP and GLU are both negatively charged); *polar* or *positive* identifying hydrophilic residues, an indication that the region is exposed to the exterior and thus apt to contain a binding site; *small* as small residues often provide the needed flexibility for the 3D conformation of the site. A similar branch is learned most of the time for the left context of the target residue too. Note that by concatenating the relation *within_5* twice, learned TETs manage to explore a wide context of the target residue. Empirical evidence in [12] shows that wide contexts (in a range of up to 15 residues) are useful for accurate prediction. Branches encoding standard regular expressions such as *CXXC* (two CYS separated by any two residues) are added at the latest stages only, i.e. they only play a minor refinement role.

The learning experiments reported here took in the range of 30 seconds (slotchain) and 19 hours (Bib-resolution; per fold) to complete on an Intel Xeon 3.2 GHz platform, with metal binding sites (3 h) and venue resolution (5 h; per fold) in between. By far the most time is spent on data retrieval from the underlying MySQL database.

6 Conclusion

The TET language is a very simple yet highly expressive language for representing features in relational data. Unlike previous feature representation frameworks a TET represents raw count of count statistics, not simple numeric or Boolean features obtained by aggregation of such counts. Our current TET implementation is restricted to Boolean attributes and relations only. However, on the representational, semantic level, there is no problem with extending the TET framework to multi-valued and numeric datatypes. However, numeric data poses additional challenges for transforming the complex TET values into more manageable, condensed features that then can be used in standard predictive models.

We have defined a discriminant function that enables prediction directly on the basis of TET values. Using this discriminant function and a new generalized information gain measure, we have developed a TET learner that in our experiments has been able to discover relevant and interpretable features in a variety of learning settings that differ significantly both with regard to the structure of the data, and the type of learning task.

The discriminant function we used in this paper is motivated by its grounding in classical prediction models, and the fact that it is easy to learn and evaluate, which makes it suitable for use in a wrapper evaluation procedure. In spite of its simplicity, we achieve with the discriminant function in our experiments predictive performance that is competitive with other state-of-the-art models, and sometimes (Cora) outperforms models that have been built using hand-coded domain knowledge. It must be emphasized, though, that the TET feature language is not tied to this discriminant function. Future work, therefore, will be directed towards constructing refined TET-based classification models using tools for collective and multi-task classification, as well as integrating TET features into other existing supervised learning techniques, e.g. kernel methods or relational probability trees. Also, in future work, we will further investigate "relational" information gain measure (our *gig* at this point being a somewhat ad-hoc solution) in order to obtain better theoretical justifications for the current *gig*, or improved versions thereof.

References

1. Van Assche, A., Vens, C., Blockeel, H., Dzeroski, S.: First order random forests: Learning relational classifiers with complex aggregates. Machine Learning 64, 149–182 (2006)
2. Blockeel, H., De Raedt, L.: Lookahead and discretization in ilp. In: Proc. of the 7th Int. Workshop on ILP, pp. 77–84 (1997)
3. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence (101), 285–297 (1998)
4. Castillo, L.P., Wrobel, S.: A comparative study on methods for reducing myopia of hill-climbing search in multirelational learning. In: Proc. of ICML 2004 (2004)
5. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proc. of IJCAI 1999 (1999)
6. Hulo, N., Sigrist, C.J.A., Le Saux, V., Langendijk-Genevaux, P.S., Bordoli, L., Gattiker, A., De Castro, E., Bucher, P., Bairoch, A.: Recent improvements to the prosite database. Nucleic Acids Research 32(Database-Issue), 134–137 (2004)
7. Jaeger, M.: Type extension trees: a unified framework for relational feature construction. In: Proceedings of Mining and Learning with Graphs (MLG 2006) (2006)
8. Jensen, D., Neville, J., Hay, M.: Avoiding bias when aggregating relational data with degree disparity. In: Proc. of ICML 2003 (2003)
9. Knobbe, A.J., Siebes, A., van der Wallen, D.: Multi-relational decision tree induction. In: Żytkow, J.M., Rauch, J. (eds.) PKDD 1999. LNCS (LNAI), vol. 1704, pp. 378–383. Springer, Heidelberg (1999)
10. Neville, J., Jensen, D.: Collective classification with relational dependency networks. In: Proc. of 2nd Int. Workshop on Multi-Relational Data Mining, pp. 77–91 (2003)
11. Neville, J., Jensen, D., Friedland, L., Hay, M.: Learning relational probability trees. In: Proceedings of SIGKDDD 2003 (2003)
12. Passerini, A., Punta, M., Ceroni, A., Rost, B., Frasconi, P.: Identifying cysteines and histidines in transition-metal-binding sites using support vector machines and neural networks. Proteins 65(2), 305–316 (2006)
13. Perlich, C., Provost, F.: Aggregation-based featrue invention and relational concept classes. In: Proc. of SIGKDD 2003 (2003)
14. Popescul, A., Ungar, L.H.: Feature generation and selection in multi-relational statistical learning. In: Getoor, L., Taskar, B. (eds.) Statistical Relational Learning. MIT Press, Cambridge (2007)
15. Singla, P., Domingos, P.: Entity resolution with markov logic. In: Perner, P. (ed.) ICDM 2006. LNCS (LNAI), vol. 4065. Springer, Heidelberg (2006)
16. Hanley, J.A., McNeil, B.J.: A method of comparing the areas under receiver operating characteristic curves derived from the same cases. Radiology 148(3), 839–843 (1983)

Feature Construction Using Theory-Guided Sampling and Randomised Search

Sachindra Joshi[1], Ganesh Ramakrishnan[1], and Ashwin Srinivasan[1,2]

[1] IBM India Research Laboratory, 4-C, Vasant Kunj Institutional Area
New Delhi 110070, India
[2] Dept. of Computer Science and Engineering & Centre for Health Informatics,
University of New South Wales, Sydney
{jsachind,ganramkr,ashwin.srinivasan}@in.ibm.com

Abstract. It has repeatedly been found that very good predictive models can result from using Boolean features constructed by an an Inductive Logic Programming (ILP) system with access to relevant relational information. The process of feature construction by an ILP system, sometimes called "propositionalization", has been largely done either as a pre-processing step (in which a large set of possibly useful features are constructed first, and then a predictive model is constructed) or by tightly coupling feature construction and model construction (in which a predictive model is constructed with each new feature, and only those that result in a significant improvement in performance are retained). These represent two extremes, similar in spirit to filter and wrapper-based approaches to feature selection. An interesting, third perspective on the problem arises by taking search-based view of feature construction. In this, we conceptually view the task as searching through subsets of all possible features that can be constructed by the ILP system. Clearly an exhaustive search of such a space will usually be intractable. We resort instead to a randomised local search which repeatedly constructs randomly (but non-uniformly) a subset of features and then performs a greedy local search starting from this subset. The number of possible features usually prohibits an enumeration of all local moves. Consequently, the next move in the search-space is guided by the errors made by the model constructed using the current set of features. This can be seen as sampling non-uniformly from the set of all possible local moves, with a view of selecting only those capable of improving performance. The result is a procedure in which a feature subset is initially generated in the pre-processing style, but further alterations are guided actively by actual model predictions. We test this procedure on language processing task of word-sense disambiguation. Good models have previously been obtained for this task using an SVM in conjunction with ILP features constructed in the pre-processing style. Our results show an improvement on these previous results: predictive accuracies are usually higher, and substantially fewer features are needed.

1 Introduction

Most machine-learning techniques for constructing models from sample data are feature-based. By this, we mean that they expect data in which objects are

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 140–157, 2008.

encoded as vectors of values for a pre-specified set of attributes or features, with the object of the model often being to predict the value of one of these features using the values of the others.[1] One machine-learning approach that clearly does not fit into this category is Inductive Logic Programming (ILP), which deals instead with learning from instances of objects represented in a relational form. For example, suppose we are concerned with building models for identifying toxic chemicals. An appropriate relational representation may be the atoms in the chemical, their location in 3-dimensional space, the bonds amongst them, the different structural and functional group locations (benzene rings, methyl groups *etc.*) and so on. A feature-based representation, on the other hand, may use the bulk properties of the chemical (its molecular weight, the number of atoms of a particular type *etc.*), some structural features (the presence of fused benzene rings, for example), and so on. ILP systems are able to construct models using the former representation, while methods like decision trees, neural networks, support vector machines require the latter.

Despite an ILP system's ability to handle more complex representations, there has long flourished a somewhat heretical strand of ILP activity that has sought to convert (in some efficient manner) the relational representation into a feature-based one, with a view of using well-established feature-based model construction methods. While a theoretical case could be made for some of this work—positive learnability results for specific forms were shown nearly 15 years ago by Dzeroski *et. al* [16]—the primary motivations have been practical. First, feature-based model constructors are computationally efficient. Second, with appropriate features, they are able to construct significantly complex predictive models. Third, both data analysts and domain experts alike are more familiar with these methods than they are with ILP. This form of model construction, with an ILP system providing some or all of the features, and a feature-based learner constructing the actual model has repeatedly been shown to be remarkably effective (see [2,17,18] for some examples).

If we are indeed committed to using a feature-based learner, then a case can be made for ILP as a natural choice for the automatic construction of new features, if the relational information can be encoded in a logical form. The techniques essentially fall into one of two categories: those that construct features independent of the subsequent model-constructor; and those that inter-leave feature and model-construction. The earliest example of the former is LINUS [1], that, given a set of relations, constructed all features possible within some syntactic restrictions. Essentially a similar idea, but using a much more efficient technique for generating the features is executed by [3]. Also in the same category are approaches like those exemplified by [19], which impose additional semantic constraints on the features (using modern terminology, these constraints effectively place minimum requirements on the support and confidence values). The inter-leaved approach sequentially builds up a set of features, in a manner

[1] The terminology, from statistical modelling, of predicting a dependent variable using a set of independent variables is related, but we avoid it here, since independence amongst the features is often not as apparent.

similar to forward-selection techniques used by a statistical model constructor. That is, features constructed by the ILP engine are given one at a time to the model constructor, and only those that actually improve predictive performance are retained. The k-FOIL system [4] and the SAYU procedure proposed in [5] are recent examples of this. The reader will recognise that the two approaches are similar in spirit to filter and wrapper-based approaches to feature selection [20]: the difference being, of course, that we do not already have the features to select from here.

There is an interesting, third perspective to be gained by accepting the fact that what is being done is actually a search for the best subset of features that can be constructed by an ILP engine. Clearly, for any realistic problem, this space would be too large to search exhaustively. In this paper, we examine instead the use of a well-established method of randomised search for exploring this space. There are three specific advantages we gain from adopting this perspective: (1) Randomised search techniques, although not provably optimal, have been shown to be very effective in searching extremely large searce spaces. We are now able to apply them to the task of feature construction; (2) The pre-processing and inter-leaved forms of feature construction can be seen as special cases of a randomised algorithm searching through the feature-subset space; (3) We are able to understand better some of the steps of feature construction in terms of generating a non-uniform sample of local moves for the randomised search. In this paper, we implement the simplest form of randomised search inspired by the GSAT algorithm [21], and investigate its performance on language processing tasks for which ILP-constructed features (generated in the pre-processing style) have been found to be useful.

The rest of the paper is organised as follows. Section 2 formulates feature-construction by an ILP engine as a search process. The use of a randomised local search procedure to execute this search is described in Section 2.1. The basic procedure is usually impractical to use directly. Modifications to the basic randomised procedure in the form of using the errors made by the model-constructor to guide sampling of local moves are in Section 2.2. We examine the possibility of reducing the computational burden further by using models that assign weights to features. This is described in Section 2.3. An empirical evaluation of the approach for the word-sense disambiguation problems is presented in Section 3. Section 5 concludes the paper.

2 Feature Construction as Search

We motivate the approach we propose using the "trains" problem, originally proposed by Ryzhard Michalski. The task, familiar to many readers, is to construct a model that can discriminate between eastbound and westbound trains, using properties of their carriages, and the loads carried (see Fig. 1).

We will assume that the trains can be adequately described by background predicates that will become evident shortly. Further, let us assume that the 10 trains shown in the figure are denoted $t1, t2, \ldots, t10$ and that their

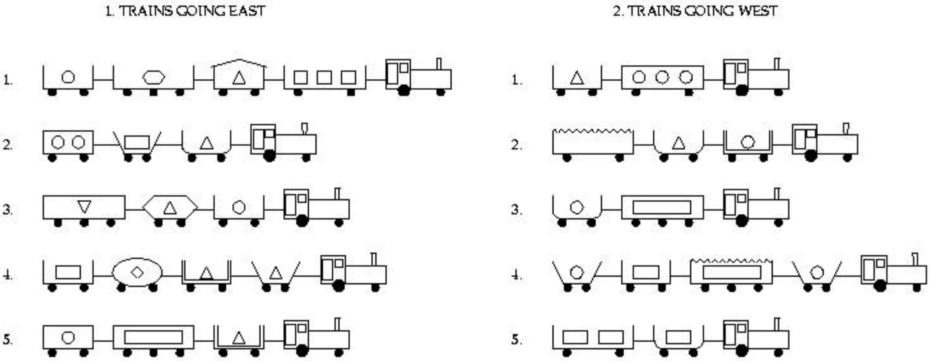
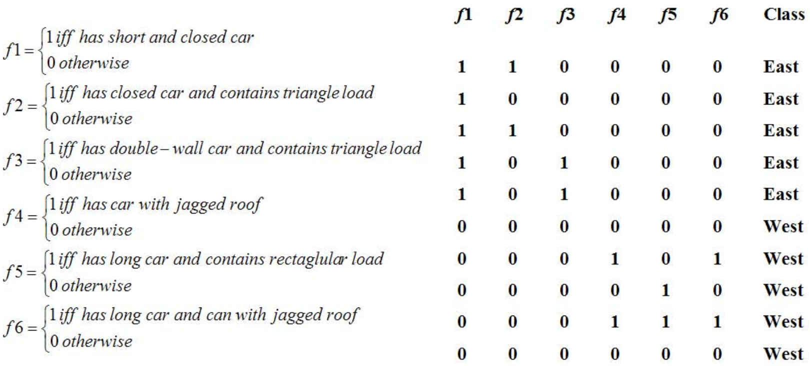


Fig. 1. The trains problem. Trains are classified either as "eastbound" or "westbound". They have open or closed carriages of different shapes, lengths, and so on. The carriages contain loads of different shapes and numbers. The task is to construct a model that, given the description of a train, can predict whether it will be eastbound or westbound.

classifications are encoded (in the Prolog language) as a set of examples of the form: $class(t1, eastbound)$, $class(t2, eastbound)$, ..., $class(t10, westbound)$. As is quite normal in the use of ILP for feature-construction, we will assume features to be Boolean valued, and obtained from some clause identified by the ILP program. For example, Fig. 2 shows five such features, found by an ILP engine, and the corresponding tabular representation of the 10 examples in Fig. 1.

Suppose that these 6 features are the only features that can be constructed by the ILP engine, and further, that it is our task to find the best subset of these that can result in the best model. Clearly, if we simply evaluated models obtained with each of the 63 subsets of the set $\{f1, f2, f3, f4, f5, f6\}$ and return the subset that returned the best model, we would be done. Now, let us consider



$$f1 = \begin{cases} 1 \; \textit{iff has short and closed car} \\ 0 \; \textit{otherwise} \end{cases}$$

$$f2 = \begin{cases} 1 \; \textit{iff has closed car and contains triangle load} \\ 0 \; \textit{otherwise} \end{cases}$$

$$f3 = \begin{cases} 1 \; \textit{iff has double-wall car and contains triangle load} \\ 0 \; \textit{otherwise} \end{cases}$$

$$f4 = \begin{cases} 1 \; \textit{iff has car with jagged roof} \\ 0 \; \textit{otherwise} \end{cases}$$

$$f5 = \begin{cases} 1 \; \textit{iff has long car and contains rectaglular load} \\ 0 \; \textit{otherwise} \end{cases}$$

$$f6 = \begin{cases} 1 \; \textit{iff has long car and can with jagged roof} \\ 0 \; \textit{otherwise} \end{cases}$$

f1	f2	f3	f4	f5	f6	Class
1	1	0	0	0	0	East
1	0	0	0	0	0	East
1	1	0	0	0	0	East
1	0	1	0	0	0	East
1	0	1	0	0	0	East
0	0	0	0	0	0	West
0	0	0	1	0	1	West
0	0	0	0	1	0	West
0	0	0	1	1	1	West
0	0	0	0	0	0	West

Fig. 2. Some Boolean features for the trains problem, and a corresponding tabular representation of the trains in Fig. 1

a more practical situation. Suppose the features that can be constructed by an ILP system are not in the 10s, but in the 1000s or even 100s of 1000s. This would make it intractable to construct models with all possible subsets of features. Further, suppose constructing each feature is not straightforward, computationally speaking, making it impractical to even use the ILP engine to construct all the features in the first place. Are we able to nevertheless able to determine the subset that would yield the best model (which we will now interpret to mean the model with the highest classification accuracy). The problem to be addressed is shown in Fig. 3.

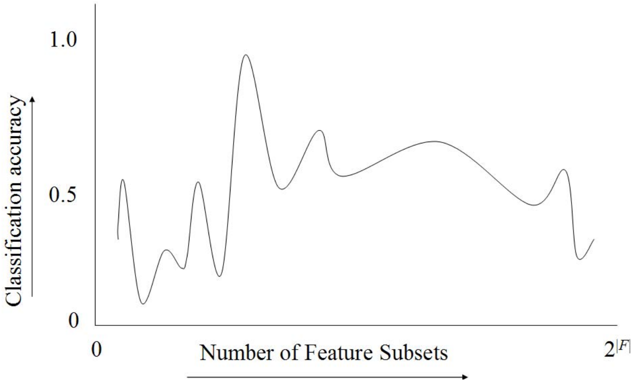


Fig. 3. Identifying the best subset of features for a model-construction algorithm A. The X-axis enumerates the different subsets of features that can be constructed by an ILP engine ($\mathcal{F}$ denotes the set of all possible features that can be constructed by the engine). The Y-axis shows the probability that an an instance drawn randomly using some pre-specified distribution will be correctly classified by a model constructed by A, given the corresponding subset on the X-axis. We wish to identify the subset $k*$ that yields the highest probability, without actually constructing all the features in $\mathcal{F}$.

Readers will recognise this as somewhat similar to the problem addressed by a randomised procedure for distribution-estimation like Gibb's sampling. There, if the $\mathcal{F}$ features are given (or at least can be enumerated), then it is possible to converge on the best-performing subset without examining the entire space of $2^{|\mathcal{F}|}$ elements. Clearly, if we are unable to generate all possible features in $\mathcal{F}$ beforehand, we are not in a position to use these methods. Instead, we resort a randomised local search procedure inspired by a randomised procedure for checking satisfiability of Boolean formulae.

2.1 A Randomised Local Search Procedure

Randomised local search procedures are some of the most effective methods proposed for addressing the hard problem of determining the satisfiability of propositional formulae. The basic search procedure embodied in a technique like GSAT [21] is straightforward (Fig. 4, taken from [6]).

(Here R and M represent the number of random restarts and moves allowed.) This kind of search procedure has been adapted successfuly to the ILP problem of identifying a set of clauses that, along with some background knowledge, entail a set of examples [15]. Here, we examine its use for feature construction (Fig. 5).

Existing techniques for feature construction can be re-cast as special cases of the procedure in Fig. 5, with appropriate values assigned to R and M; and definitions of a starting point (Step 3a) and local moves (Step 3(f)i). For example, "LINUS-inspired" methods that use an ILP engine to construct a large number of features independent of the model constructor can be seen as an instance of the randomised procedure with $R = 1$ and $M = 0$. Some additional constraints may be imposed on the "starting subset" selected in Step 3a, which could be encapsulated in the distribution used for random selection. Any further selection amongst features in this subset are then upto the model-constructor. On the other hand, SAYU-like procedures can be emulated with $R = 1$ and large values of M; along with restrictions that force the search to start from an empty subset, and local moves only to add a single feature at a time.

```
1. currentbest:= 0 ("0" is some conventional default answer)
2. for i = 1 to R do begin
   (a) current:= randomly selected starting point
   (b) if current is better than currenbest then currentbest:= current
   (c) for j = 1 to M do begin
         i. next:= best local move from current
        ii. if next is better than currenbest then currentbest:= next
       iii. current:= next
   (d) end
3. end
4. return currentbest
```

Fig. 4. A basic randomised local search procedure

```
1. bestfeatures:= {}
2. bestaccuracy:= 0.0
3. for i = 1 to R do begin
   (a) currentfeatures:= randomly selected set of features
   (b) currentmodel:= model constructed with currentfeatures
   (c) accuracy:= estimated accuracy of currentmodel
   (d) if accuracy > bestaccuracy then begin
         i. bestfeatures:= currentfeatures
        ii. bestaccuracy:= accuracy
   (e) end
   (f) for j = 1 to M do begin
         i. nextfeatures:= best local move from currentfeatures
        ii. nextmodel:= model constructed with nextfeatures
       iii. accuracy:= estimated accuracy of nextmodel
        iv. if accuracy > bestaccuracy then begin
              A. bestfeatures:= nextfeatures
              B. bestaccuracy:= accuracy
         v. end
        vi. currentfeatures:= nextfeatures
   (g) end
4. end
5. return bestfeatures
```

Fig. 5. The basic randomised local search procedure, adapted to the task of feature construction

We consider now the more general procedure shown. In this, R and M can take on any value from the set of natural numbers (including 0). In addition, the starting subset is assumed to be drawn using some distribution that need not be known; and a local move is one that either adds a single new feature to the existing subset of features, or drops a feature from the existing subset. We are now immediately confronted with two issues that make it impractical to use the procedure as shown. First, we have the same difficulty that prevented us from using an enumerative technique like a Gibb's sampler: generating the local neighbourhood requires us to obtain all possible single-feature additions. Second, for each local move, we need to construct a model, which can often be computationally expensive. We address each of these in turn.

2.2 Reducing Local Moves Using Theory-Guided Sampling

In this section, we consider a modification of the search procedure in Fig. 5 that results in only examining a small sample of all the local moves possible before deciding on the next move in Step 3(f)i. Ideally, we are interested in obtaining a sample that, with high probability, contains the best local move possible. Assuming there are no ties, and that the number of possible local moves is very large, it would clearly be undesirable to select the sample using a uniform distribution over local moves. We propose instead a selection that uses the errors made by the model-constructor to obtain a sample of local moves. As a result, features in the local neighbourhood that are relevant to the errors are more likely to be selected. In some sense, this is somewhat reminiscent of boosting methods: here, instead of increasing the weights of examples incorrectly classified, the representation language is enriched in a way that is biased to classify these examples correctly on subsequent iterations.

Recall that at any point, a local move from a feature-subset F is obtained by either dropping an existing feature in F or adding a new feature to F. We are specifically concerned with the addition step, since in principle, all possible features that can be constructed by the ILP engine could be considered as candidates. We curtail this in the following ways. First, we restrict ourselves to samples of features that are relevant to examples misclassifed by the model-constructor using the current set of features F (by "relevant" we mean those that are TRUE for at least one of the examples in error). Second, in our implementation, we restrict ourselves to a single new feature for each such example (by "new", we mean a feature not already in F).[2] The altered procedure is in Fig. 6.

2.3 Reducing Models Constructed Using Feature Weights

Step 3(f)v although now considering fewer moves, still has to construct a model for each move before deciding on the best one. For some model-construction

[2] In the implementation, we select this feature using its discriminatory power given the original set of examples.

```
1. bestfeatures:= {}
2. bestaccuracy:= 0.0
3. for i = 1 to R do begin
   (a) currentfeatures:= randomly selected set of features
   (b) currentmodel:= model constructed with currentfeatures
   (c) accuracy:= estimated accuracy of currentmodel
   (d) if accuracy > bestaccuracy then begin
       i. bestfeatures:= currentfeatures
       ii. bestaccuracy:= accuracy
   (e) end
   (f) for j = 1 to M do begin
       i. $F^-$:= set of feature subsets obtained by dropping a feature from currentfeatures
       ii. $F_{new}$:= sample of new features that are TRUE for errors made by currentmodel
       iii. $F^+$:= set of feature subsets obtained by adding a feature in $F_{new}$ to currentfeatures
       iv. localmoves:= $F^- \cup F^+$
       v. nextfeatures:= best subset in localmoves
       vi. nextmodel:= model constructed with nextfeatures
       vii. accuracy:= estimated accuracy of nextmodel
       viii. if accuracy > bestaccuracy then begin
             A. bestfeatures:= nextfeatures
             B. bestaccuracy:= accuracy
       ix. end
       x. currentfeatures:= nextfeatures
       xi. currentmodel:= nextmodel
   (g) end
4. end
5. return bestfeatures
```

Fig. 6. The randomised local search procedure for feature construction, modified using theory-guided sampling of local moves

methods, this may also be computationally too expensive. We examine the possibility of using feature weights to reduce the computational burden further.

The principal purpose of constructing and evaluating models in the local neighbourhood is to decide on the best next move to make. This will necessarily involve either an addition of a new feature to the existing set of features, or the deletion of an existing feature from the current set of features. That is, we are looking to find the best new feature to add, or the worst old feature to drop (given the other features in the set, of course). Correctly, we would form models with each old feature omitted in turn from the current set and each new feature added in turn to the current set. The best model would then determine the next move. Using a model-constructor that assigns weights to features allows us to adopt the following sub-optimal procedure instead. First, we find the feature with the lowest weight in the current model: this is taken to be the worst old feature. Next, we construct a single model with all features (old and new). Let us call this "extended model". The best new feature is taken to be the new feature with the highest weight in the extended model. The procedure in Fig. 6 with this further modification is shown in Fig. 7. It is evident that the number of additional models constructed at any point in the search space is now reduced to just 3: the price we pay is that we are not guaranteed to obtain the same result as actually performing the individual additions and deletions of features.

It is the randomised local search procedure in Fig. 7, with one small difference, that we implement and evaluate empirically in this paper. The difference arises from the comparison of models: in the procedure shown, this is always done using

```
bestfeatures:=
bestaccuracy:= 0.0
for i = 1 to R do begin
    currentfeatures:= randomly selected set of features
    currentmodel:= model constructed with currentfeatures
    accuracy:= estimated accuracy of currentmodel
    if accuracy > bestaccuracy then begin
        bestfeatures:= currentfeatures
        bestaccuracy:= accuracy
    end
    for j = 1 to M do begin
        $F_{new}$:= sample of new features that are TRUE for errors made by currentmodel
        extendedmodel:= model constructed using currentfeatures and $F_{new}$
        $f_{worst}$:= feature in currentfeatures with lowest weight in currentmodel
        $f_{best}$:= feature in $F_{new}$ with highest weight in extendedmodel
        $F^-$:= set with feature subset obtained by dropping $f_{worst}$ from currentfeatures
        $F^+$:= set with feature subset obtained by adding $f_{best}$ to currentfeatures
        localmoves:= $F^- \cup F^+$
        nextfeatures:= best subset in localmoves
        nextmodel:= model constructed with nextfeatures
        accuracy:= estimated accuracy of nextmodel
        if accuracy > bestaccuracy then begin
            bestfeatures:= nextfeatures
            bestaccuracy:= accuracy
        end
        currentfeatures:= nextfeatures
        currentmodel:= nextmodel
    end
end
return bestfeatures
```

Fig. 7. The randomised local search procedure for feature construction, modified using theory-guided sampling of local moves and the use of feature-weights to reduce model construction

estimated accuracies only. In our implementation, if estimated accuracies for a pair of models are identical, then the model using fewer features is prefered (that is, comparisons are done on the pair (A, F) where A is the estimated accuracy of the model and F is the number of features used in the model).

One final point is worth clarifying. This concerns how the procedure in Fig. 7 is to avoid over-fitting the data. There are three principal ways in which we see this can be achieved: (1) Estimated accuracies of the model constructed, if unbiased, should give the procedure a way of halting before over-fitting; (2) The feature-constructor—here an ILP learner—can ensure that features have some minimal support (in the data mining sense of the word); and (3) The model constructor can perform some appropriate trade-off of fit-versus-complexity to avoid over-fitting.

3 Empirical Evaluation

Our objective is to evaluate empirically the effectiveness of the approach based on randomised search for feature construction. Specifically, we intend to compare the performance of following two kinds of feature-construction techniques on word-sense disambiguation problems in language processing, for which ILP-based feature construction methods have been previously studied in the literature [2]:

1. *Features constructed prior to model construction.* An ILP system first constructs all interesting features that are consistent with the constraints provided by the background knowledge. Models are then constructed using these features. Any feature selection that may be necessary is done prior to model construction.
2. *Features constructed using randomised local search and theory-guided sampling.* This is the procedure we have described in this paper (specifically, the one in Fig. 7). Models are constructed using the same model construction procedure as above, but no separate step of feature-selection is done.

3.1 Materials

Data and Background Knowledge. We use two benchmark data sets for Word Sense Disambiguation (WSD) problem introduced in [2] to investigate the effectiveness of our proposed method. WSD is an important problem that needs to be solved in many natural language tasks such as machine translation, information retrieval, speech and text processing and so on. Although complete descriptions of the data are available in [2], we describe them briefly here for completeness.

Monolingual task. This data consist of the 32 verbs from the SENSEVAL-3 competition [7]. SENSEVAL[3] is a joint evaluation effort for WSD and related tasks. We use all the verbs of the English lexical sample task from the competition. The number of examples for each verb varies from 40 to 398 (average of 186). The number of senses varies from 3 to 12 with an average of 7 senses. The average accuracy of the majority class is about 55%. The benchmark identifies 66% of the data that can be used for model construction. The rest are used for testing models.

Bilingual task. This task consists of 7 highly ambiguous verbs in machine translation from English to Portuguese. The sample corpus comprises around 200 English sentences for each verb extracted from a corpus of fiction books. In that corpus, the number of translations varies from 5 to 17, with an average of 11 translations. The average accuracy of the majority class is about 54%.

In [2], 9 categories of background predicates were introduced. Of these, the category B0 consists of some simple hand-crafted features. The background predicates specifically of relevance to an ILP-based feature constructor are in categories B1–B8. These consist of: (B1) the local context of the verb in a sentence; (B2) lemmas of 5 content words to the right and left of the verb, lemmatized by MINIPAR [9]; (B3) part-of-speech tags of 5 content words to the right and left of the verb (obtained using MXPOST [10]); (B4) subject and object syntactic relations with respect to the verb, obtained from MINIPAR parses; (B5) collocations with respect to the verb, of the form: 1st preposition to the right, 1st

[3] www.senseval.org

and 2nd words to the left and right, 1st noun, 1st adjective, and 1st verb to the left and right; (B6) verb restrictions, in terms of the semantic features of their arguments in the sentence, extracted using LDOCE [11]; (B7) dictionary definitions, being a relative count of the overlapping words in dictionary definitions of each of the possible translations of the verb and the words; and (B8) phrasal verbs possibly occurring in a sentence.

Fig. 8 tabulates the number of facts in each of B1–B8 for the two disambiguation tasks.

Background	Monolingual Task	Bilingual Task
B1	61175	4244
B2	33969	0
B3	33969	4870
B4	6523	1598
B5	66092	10514
B6	1308	1922
B7	1591	150
B8	0	418
All	204627	23716

Fig. 8. Ground facts comprising definitions for B1–B8. B8 does not appear in the monolingual task, since SENSEVAL-3 data do not consider senses of the verbs occurring in phrasal verbs. B2 was not included in the bilingual task because it was considered to be covered by B5, since we had shorter sentences than in the monolingual task.

Algorithms. We distinguish here between three implementations that, together, comprise the randomised local search approach: (1) The implementation of the procedure in Fig. 7). This is implemented here in the Prolog language, with some associated executable files; (2) The ILP implementation used to construct features using B1–B8. We use the ILP system Aleph [6]; and (3) The model constructor. We use a linear SVM: the specific implementation used is the one provided in the WEKA toolbox called SMO.[4]

3.2 Method

Our method is straightforward:

> For each verb in each task (that is, 32 verbs in the monolingual task and 7 verbs in the bilingual task):

[4] http://www.cs.waikato.ac.nz/~ml/weka/. We obtain weights for features by using the support vector machine with linear kernel. Specifically, let $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$ represent an n-dimensional input vector. In the case of linear kernel, the predictor function can be described as: $prediction(x_i) = sgn[b + w^T x_i]$, where $w = \sum_i \alpha_i x_i$ and b is the interceptor. The linear classifier categorizes a new data point x to the positive class if linear combination $w_1x_1+w_2x_2+\ldots, w_nx_n$ is above a given threshold and to the negative class if the linear combination is below the threshold. We use the absolute value $|w_j|$ as the weight of the j^{th} feature. These weights have also been used for the feature selection problem in [13].

1. Construct a set of good features using the ILP engine. Select a subset of these as being relevant for the prediction task. Construct a model using the feature-subset selected. Call this the “model with pre-specified features” (or PreSpec model).
2. Identify a set of features using the randomised local-search procedure in Fig. 7. Obtain a model using these features. Call this the “model with features using randomised search” (or RandSearch model).
3. Compare the performance of the PreSpec model against the performance of the RandSearch model.

The following details are relevant:

(a) We use the same “training-test” splits of the data as those used in [2]. Performance will be measured by the accuracy of prediction on the test set (that is, the percentage of test examples whose sense is predicted correctly).

(b) The ILP learner constructs a set of definite clauses in line with the usual specifications for predictive ILP systems (see [14] for a statement of these). Positive examples for the ILP learner are provided by the correct sense (or translation in the bilingual case) of verbs in a sentence. Negative examples are generated automatically using all other senses (or translations). The translation of a clause into a Boolean feature is straightforward: the feature corresponding to a definite clause is a function that returns the value “true” for an example if and only if the body of the definite clause is true, given the background knowledge. The ILP engine is constrained to return only a single feature that discriminates best between the positive and negative examples. On each iteration of Fig. 7, the ILP engine is forced to construct features that are true for errors made by the current model.

(c) Although we are able to generate PreSpec models using the randomised procedure in Fig. 7 (by simply setting $R = 1$ and $M = 0$), we will take the models reported in [2] to denote the PreSpec models. We also consider a slight variation by performing some amount of experimentation to determine (locally) optimal settings for two parameters: the C parameter used by the linear SVM and F, the number of features to be selected. We determine the appropriate setting by systematic variation of the C parameter over the range 0.0001 to 10000 in multiple of 10s and F over the same range examined in [2]. The predictive accuracy with each (C, F) setting is estimated and the values that yield the best results are used to construct the final model (the predictive accuracy estimate is obtained using an average over 5 repeats of predictions on 20% of the training data sampled to form a “validation” set). We refer to these models as PreSpec* models.

(d) For RandSearch models, we use a value of $R = 10$ and $M = 5$ for the number of random restarts and iterations of local moves. We are not able to offer domain-independent guidelines on values for these parameters. In the context of using a similar approach in search for clauses [15], short periods of local search with many random restarts were found to be effective. While the values we have selected here are arbitrary, some more principled approach may be possible by a systematic empirical exploration of reasonable values. We also consider a variant that performs no local search, and returns a set of features, of the same size as those obtained by RandSearch, but constructed randomly. We call this procedure “Random”. Both cases require a “start point”, which is provided by features obtained from (repeated, if $R > 1$) random samples of 30 examples.

(e) Comparison of performance will be done using the Wilcoxon signed-rank test. The test is a non-parametric test of the null hypothesis that there is no significant difference between the median performance of a pair of algorithms. The test works by ranking the absolute value of the differences observed in performance of the pair of algorithms. Ties are discarded and the ranks are then given signs depending on whether the performance of the first algorithm is higher or lower than that of the second. If the null hypothesis holds, the sum of the signed ranks should be approximately 0. The probabilities of observing the actual signed rank sum can be obtained by an exact calculation (if the number of entries is less than 10), or by using a normal approximation. We are interested in all cases in the the directional hypothesis that the performance of RandSearch models are better than those of PreSpec models.

4 Results and Discussion

Fig. 9 and 10 tabulate the comparative performance of the PreSpec, PreSpec*, Random and RandSearch models on the two disambiguation tasks. Also included is the performance of a classifier that simply predicts the most frequent sense of the verb (as assessed on the training set). The average number of features used in each case is in Fig. 11. We note at the outset that there appears to be little to choose between “Random” and “Majority Class”, and between “PreSpec” and “PreSpec*”. Therefore, in what follows, we will not include any further discussion of either Random or PreSpec*. With this caveat, the principal details in the tabulations are these: (1) Majority Class performs worse than the other procedures; (2) For both tasks, the accuracies of the PreSpec models are usually lower than the RandSearch models. Discarding ties, the PreSpec has the highest accuracy for 10 of the 32 verbs for the monolingual task and for 1 of the 7 verbs for the bilingual task; and (3) RandSearch models use substantially fewer features than PreSpec models.

We turn now to the question of whether the differences in accuracies observed between the models are in fact significant. The probabilities calculated by using the Wilcoxon test are shown in Fig. 12.[5] The tabulations further support the position that both PreSpec and RandSearch models are significantly better than a simple majority class guesser. RandSearch is significantly better on the bilingual task, and there is reasonable evidence to believe that it is better on the

[5] These were obtained from the program kindly provided by Richard Lowry at `http://faculty.vassar.edu/lowry/wilcoxon.html`.

Verb	**Senses**	**Accuracy**				
		Majority Class	PreSpec	PreSpec*	RandSearch	Random
activate	5	82.46 ±3.56	83.33 ±3.49	82.45 ±3.56	**92.98** ±2.39	82.46 ±3.56
add	6	45.80 ±4.35	82.44 ±3.32	**83.21** ±3.27	74.81 ±3.79	46.56 ±4.36
appear	3	44.70 ±4.33	71.21 ±3.94	71.21 ±3.94	**87.12** ±2.92	43.94 ±4.32
ask	6	27.78 ±3.99	50.00 ±4.45	53.17 ±4.44	**60.32** ±4.36	29.37 ±4.06
begin	4	59.74 ±5.59	**74.03** ±5.00	72.73 ±5.07	71.43 ±5.15	59.74 ±5.59
climb	5	55.22 ±6.08	83.58 ±4.53	**86.57** ±4.17	82.09 ±4.68	55.22 ±6.08
decide	4	67.74 ±5.94	77.42 ±5.31	**80.64** ±5.02	77.42 ±5.31	67.74 ±5.94
eat	7	**88.37** ±3.46	87.21 ±3.60	**88.37** ±3.46	**88.37** ±3.46	**88.37** ±3.46
encounter	4	50.77 ±6.20	72.31 ±5.55	72.30 ±5.55	**73.85** ±5.45	36.92 ±5.99
expect	3	74.36 ±4.94	**92.31** ±3.02	**92.31** ±3.02	89.74 ±3.44	74.36 ±4.94
express	4	69.09 ±6.23	**72.73** ±6.01	67.27 ±6.32	70.91 ±6.12	69.09 ±6.23
hear	7	46.88 ±8.82	**65.63** ±8.40	62.5 ±8.55	40.62 ±8.68	46.88 ±8.82
lose	9	52.78 ±8.32	**58.33** ±8.22	52.7 ±8.32	47.22 ±8.32	52.78 ±8.32
mean	7	52.50 ±7.90	70.00 ±7.25	**75.0** ±6.85	**75.00** ±6.85	52.50 ±7.90
miss	8	33.33 ±8.61	33.33 ±8.61	36.66 ±8.80	**56.67** ±9.05	33.33 ±8.61
note	3	38.81 ±5.95	**88.06** ±3.96	**88.06** ±3.96	82.09 ±4.68	56.72 ±6.05
operate	5	16.67 ±8.78	77.78 ±9.80	72.22 ±10.55	**88.89** ±7.41	38.89 ±11.49
play	12	46.15 ±6.91	53.85 ±6.91	**55.77** ±6.89	**55.77** ±6.89	46.15 ±6.91
produce	6	52.13 ±5.15	67.02 ±4.85	65.96 ±4.88	**77.66** ±4.30	52.13 ±5.15
provide	6	85.51 ±4.24	89.86 ±3.63	86.96 ±4.05	**91.30** ±3.39	82.61 ±4.56
receive	9	88.89 ±6.05	88.89 ±6.05	88.89 ±6.05	**92.59** ±5.04	88.89 ±6.05
remain	3	78.57 ±4.90	87.14 ±4.00	85.71 ±4.18	**95.71** ±2.42	78.57 ±4.90
rule	5	50.00 ±9.13	83.33 ±6.80	83.33 ±6.80	**90.00** ±5.48	40.00 ±8.94
smell	7	40.74 ±6.69	**77.78** ±5.66	75.92 ±5.82	74.07 ±5.96	40.74 ±6.69
suspend	7	35.94 ±6.00	57.81 ±6.17	56.25 ±6.20	**68.75** ±5.79	39.06 ±6.10
talk	9	72.60 ±5.22	73.97 ±5.14	73.97 ±5.14	**75.34** ±5.04	72.60 ±5.22
treat	9	28.07 ±5.95	47.37 ±6.61	**57.89** ±6.53	49.12 ±6.62	28.07 ±5.95
use	5	71.43 ±12.07	**92.86** ±6.88	**92.86** ±6.88	**92.86** ±6.88	71.43 ±12.07
wash	12	67.65 ±8.02	**73.53** ±7.57	61.76 ±8.33	58.82 ±8.44	67.65 ±8.02
watch	7	**74.51** ±6.10	**74.51** ±6.10	72.54 ±6.24	**74.51** ±6.10	**74.51** ±6.10
win	7	44.74 ±8.07	60.53 ±7.93	57.89 ±8.00	**63.16** ±7.83	42.11 ±8.01
write	8	26.09 ±9.16	34.78 ±9.93	39.13 ±10.17	**52.17** ±10.42	34.78 ±9.93
Mean	7	55.31	71.97	71.63	74.10	56.06
Median	6	52.31	74.03	72.63	74.90	52.31

Fig. 9. Estimates of accuracies of disambiguation models on the monolingual task. "Senses" refers to the numbers of possible senses of each verb. The column labelled "Majority class" gives the accuracy of models that simply predict the most common sense of each verb (as estimated from the training data).

Verb	**Translations**	**Accuracy**				
		Majority class	PreSpec	PreSpec*	RandSearch	Random
come	11	50.30±7.62	76.74±6.44	75.56 ±6.41	**86.67** ±5.07	55.56 ±7.41
get	17	21.00±6.70	40.54±8.07	**64.1** ±7.68	51.28 ±8.00	20.51 ±6.47
give	5	88.80±4.81	95.35±3.21	**100** ±0.00	97.78 ±2.20	97.78 ±2.20
go	11	68.50±6.78	78.72±5.97	77.55 ±5.96	**83.67** ±5.28	71.43 ±6.45
look	7	50.30±7.45	82.22±5.70	76.59 ±6.17	**82.98** ±5.48	59.57 ±7.16
make	11	70.00±7.25	75.00±6.85	78.57 ±6.33	**80.95** ±6.06	73.81 ±6.78
take	13	28.50±8.24	**60.00**±8.94	56.25 ±8.77	56.25 ±8.77	12.50 ±5.85
Mean	11	53.91	72.65	75.51	77.08	55.88
Median	11	50.30	76.74	76.60	82.98	59.57

Fig. 10. Estimates of accuracies of disambiguation models on the bilingual task. "Translations" refers to the numbers of possible translations of each verb into Portuguese.

Model	**Avg. Features Used**	
	Monolingual	Bilingual
Majority class	0	0
PreSpec	250	500
RandSearch	29	27

Fig. 11. Average numbers of features required to construct models. The values for PreSpec are those reported in [2] after feature-selection was performed.

	Majority class	PreSpec	RandSearch
Majority class	–	–	–
PreSpec	< 0.001, 0.010	–	–
RandSearch	< 0.001, 0.010	0.08, 0.05	–

Fig. 12. Probablities of observing the differences in accuracies for the monolingual and bilingual tasks, under the null hypothesis that median accuracies of the pair of algorithms being compared are equal. Each entry consists of a pair of probability estimates, corresponding to the mono and bilingual tasks. The alternate hypothesis is each case is that the column-model performance is better than the row-model performance. Thus a value of 0.08 in the (RandSearch,PreSpec) entry means that there is an 8% chance that the performance of the two models are actually the same, given the performance observed on the monolingual task.

monolingual one as well (with the usual caveat that probabilities from repeated cross-comparisons should be interpreted with caution).

For the monolingual task, we are also able to add to the comparisons reported in [2], based on average accuracies for each verb. This is shown in Fig. 13. The algorithms in this table use model construction methods that include bagged decision trees (the Syntalex family), Naive Bayes (CLaC1), maximum entropy modelling (CLaC2), a multiclass perceptron (MC-WSD), and ILP. It is evident that the RandSearch models are comparable to the state-of-the-art in the field.

Models	**Accuracy**
Majority class	55.31
Syntalex-1	67.00
Syntalex-2	66.50
Syntalex-3	67.60
Syntalex-4	65.30
CLaC1	67.00
CLaC2	66.00
MC-WSD	72.50
ILP	69.15
ILP-assisted	71.97
RandSearch	74.10

Fig. 13. Comparative average accuracies of the best models reported for the SENSEVAL-3 competition. All rows except the last are as in [2]. There "ILP" refers to using an ILP engine to construct rules for disambiguation (as opposed to using a feature-based learner); and "ILP-assisted" is what we have called PreSpec here.

The superior performance of RandSearch here suggests that adequate models for the WSD tasks can be constructed with fairly small numbers of features. We note however that this may not always be the case: depending on the background knowledge available to the ILP learner, adequate models may require very large numbers of features. In this case, we believe PreSpec-like models may perform better. The distinction between PreSpec and RandSearch models is somewhat illusory: we can clearly obtain the former using the latter with an R value of 1 and an M value of 0. A related question is whether the local search is beneficial at all. We have some evidence for this: on the 32 verbs in the mono-lingual task, not performing a local search yielded a better model only on 6 verbs (there were 10 ties, and local search gave better models on the remaining 16 verbs).

5 Concluding Remarks

In this paper we investigate the applicability of using a randomised search technique for feature construction using ILP. The search method we propose can be seen as a gneralisation of some of the existing approaches to using ILP to extend a feature-based representation. A direct implementation of randomised approach is, however, computationally expensive, and we have described a number of additional modifications for practical use. Chief amongst these is the use of the errors made by a model constructed using an existing set of features to select amongst future moves in the search. Results from an empirical evaluation on some standard datasets in language processing are promising: we find predictive accuracies of models constructed are usually higher and use substantially fewer features. In cases where we are able to compare against the state-of-the-art, we find average prediction accuracies are higher than those reported earlier.

There are a number of ways in which the work here can be improved and extended. We list the main limitations here under three categories. On the theoretical front, it is evident that we have not provided any guarantees of optimality on the feature-subset constructed. While this is typical of randomised methods of the type proposed here, it would nevertheless be useful to obtain some performance bounds, however loose.

On the implementation front, our implementation is based on the simplest kind of randomised search (GSAT). Better methods exist and need to be investigated (for example, WalkSat). Further, we could consider other neighbourhood definitions for the local search such as adding or dropping upto k features. Of course, we are not restricted to use SVMs, or even the specific variant of SVM here, as our model constructor. Our experiments on the monolingual task here suggest that there is no significant difference between a “1-norm” SVM and the standard approach we have used here, but other model construction techniques may yield better results.

On the application front, we have evaluated our search procedure on a specific set of problems (namely, those concerned with word-sense disambiguation); and against a specific kind of feature-construction (in which all features are constructed before model construction). Clearly, testing on other datasets is desirable. We also need to extend the comparative study to include methods like kFOIL [4] that perform "dynamic propositionalisation" (that is, generate features incrementally).

These limitations notwithstanding, we believe there is sufficient evidence to believe that the randomised approach used here could provide an interesting way to interleave the construction of features and their associated models. We note that when used in conjunction with a statistical model constructor (as we have done here), we are effectively performing a form of statistical relational learning. This aspect of the search-based approach needs to be investigated further.

Acknowledgements

The authors would like to thank Lucia Specia for generously providing the data used in the tasks here. Jimmy Foulds, at the University of Waikato, helped beyond the call of duty in getting an executable version of code that constructed models using a 1-norm SVM within WEKA.

References

1. Lavrac, N., Dzeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with LINUS. Technical report, Jozef Stefan Institute (1990)
2. Specia, L., Srinivasan, A., Ramakrishnan, G., Nunes, G.V.: Word Sense Disambiguation Using Inductive Logic Programming. In: Muggleton, S., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 409–423. Springer, Heidelberg (2007)
3. Zelezny, F.: Efficient Construction of Relational Features. In: Proceedings of the 4th Int. Conf. on Machine Learning and Applications, pp. 259–264. IEEE Computer Society Press, Los Angeles (2005)
4. Landwehr, N., Passerini, A., Raedt, L.D., Frasconi, P.: kFOIL: Learning Simple Relational Kernels. In: Gil, Y., Mooney, R. (eds.) Proc. Twenty-First National Conference on Artificial Intelligence (AAAI 2006) (2006)
5. Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., Costa, V.S.: Change of representation for statistical relational learning. In: Proc. IJCAI 2007 (2007)
6. Srinivasan, A.: The Aleph Manual (1999), http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
7. Mihalcea, R., Chklovski, T., Kilgariff, A.: The SENSEVAL-3 English Lexical Sample Task. In: SENSEVAL-3: Third International Workshop on the Evaluation of Systems for Semantic Analysis of Text, Barcelona, pp. 25–28 (2004)
8. Specia, L., Nunes, M.G.V., Stevenson, M.: Exploiting Parallel Texts to Produce a Multilingual Sense-tagged Corpus for Word Sense Disambiguation. In: RANLP 2005, Borovets, pp. 525–531 (2005)
9. Lin, D.: Principle based parsing without overgeneration. In: 31st Annual Meeting of the Association for Computational Linguistics, Columbus, pp. 112–120 (1993)
10. Ratnaparkhi, A.: A Maximum Entropy Part-Of-Speech Tagger. In: Empirical Methods in NLP Conference, University of Pennsylvania (1996)
11. Procter, P.(ed.): Longman Dictionary of Contemporary English. Longman Group, Essex (1978)

12. Parker, J., Stahel, Password, M.: English Dictionary for Speakers of Portuguese. Martins Fontes, São Paulo (1998)
13. Brank, J., Grobelnik, M., Milic-Frayling, N., Mladenic, D.: Feature Selection Using Linear Support Vector Machines Technical report, Micorsoft Research, MSR-TR-2002-63
14. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods. J. Log. Program 19/20, 629–679 (1994)
15. Zelezny, F., Srinivasan, A., Page Jr., C.D.: Randomised restarted search in ILP. Machine Learning 64(1-3), 183–208 (2006)
16. Dzeroski, S., Muggleton, S., Russell, S.J.: PAC-Learnability of Determinate Logic Programs. In: COLT 1992, pp. 128–135 (1992)
17. King, R.D., Karwath, A., Clare, A., Dehaspe, L.: Genome scale prediction of protein functional class from sequence using data mining. In: KDD 2000, pp. 384–389 (2000)
18. Ramakrishnan, G., Joshi, S., Balakrishnan, S., Srinivasan, A.: Using ILP to Construct Features for Information Extraction from Semi-structured Text. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 211–224. Springer, Heidelberg (2008)
19. Srinivasan, A., King, R.D.: Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. Data Mining and Knowledge Discovery 3(1), 37–57 (1999)
20. Kohavi, R., John, G.H.: Wrappers for Feature Subset Selection. Artif. Intell. 97(1-2), 273–324 (1997)
21. Selman, B., Levesque, H.J., Mitchell, D.G.: A New Method for Solving Hard Satisfiability Problems. In: AAAI 1992, pp. 440–446 (1992)

Foundations of Onto-Relational Learning

Francesca A. Lisi and Floriana Esposito

Dipartimento di Informatica, Università degli Studi di Bari
Via E. Orabona 4, 70125 Bari, Italy
{lisi,esposito}@di.uniba.it

Abstract. ILP is a major approach to Relational Learning that exploits previous results in concept learning and is characterized by the use of prior conceptual knowledge. An increasing amount of conceptual knowledge is being made available in the form of ontologies, mainly formalized with Description Logics (DLs). In this paper we consider the problem of learning rules from observations that combine relational data and ontologies, and identify the ingredients of an ILP solution to it. Our proposal relies on the expressive and deductive power of the KR framework $\mathcal{DL}$+log that allows for the tight integration of DLs and disjunctive DATALOG with negation. More precisely we adopt an instantiation of this framework which integrates the DL $\mathcal{SHIQ}$ and positive DATALOG. We claim that this proposal lays the foundations of an extension of Relational Learning, called Onto-Relational Learning, to account for ontologies.

1 Motivation

Due to the close relation between Logic Programming and Relational Databases [5], ILP has established itself as a major approach to Relational Learning. Indeed, the function-free fragment of Horn Clausal Logic (HCL) known as DATALOG [4] is the most widely used in ILP for Knowledge Representation (KR) purposes. Most ILP exploits previous results in concept learning. A distinguishing feature of ILP is the use of prior conceptual knowledge. Though this feature has been widely recognized as one of the strongest points of ILP, the background knowledge in ILP is often not organized around a well-formed conceptual model. This practice seems to ignore the growing demand for an ontological foundation of knowledge in intelligent systems. Indeed, an increasing amount of conceptual knowledge is being made available in the form of *ontologies*. In Artificial Intelligence, an ontology refers to an engineering artifact (more precisely, produced according to the principles of *Ontological Engineering* [12]), constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. This set of assumptions has usually the form of a first-order logical (FOL) theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. In the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 158–175, 2008.

intended interpretation. More formally, an ontology is a formal explicit specification of a shared conceptualization for a domain of interest [13]. Among the other things, this definition emphasizes the fact that an ontology has to be specified in a language that comes with a formal semantics. Only by using such a formal approach ontologies provide the machine interpretable meaning of concepts and relations that is expected when using an ontology-based approach. Among the formalisms proposed by Ontological Engineering, the most currently used are *Description Logics* (DLs) [1]. Note that DLs are decidable fragments of FOL that are incomparable with HCL as regards the expressive power [2] and the semantics [27]. Yet, DLs and HCL can be combined according to some limited forms of hybridization. Therefore the adoption of such hybrid KR systems can help overcoming the current difficulties in accommodating ontologies in ILP.

In this paper we consider the problem of learning rules from observations that combine relational data and ontologies, and identify the ingredients of an ILP solution to it. Our proposal relies on the expressive and deductive power of the KR framework $\mathcal{DL}$+log [28] that allows for the tight integration of DLs and disjunctive Datalog with negation (Datalog$^{\neg\vee}$) [8]. More precisely, we adopt an instantiation of this framework obtained by integrating the DL $\mathcal{SHIQ}$ [15] and positive Datalog [4]. We claim that this proposal lays the foundations of an extension of Relational Learning, called Onto-Relational Learning, to account for ontologies.

The paper is organized as follows. Section 2 provides background information on the integration of ontologies and relational data from both the KR and the ILP perspective. Section 3 introduces the KR framework of $\mathcal{DL}$+log. Section 4 defines the ILP framework for learning $\mathcal{SHIQ}$+log rules. Section 5 concludes the paper with final remarks. More details of Datalog$^{\neg\vee}$ and DLs can be found in Appendix A and B, respectively.

2 Background

2.1 Ontologies and Relational Data

The integration of ontologies and relational databases follows the tradition of KR research on *hybrid systems*, i.e. those systems which are constituted by two or more subsystems dealing with distinct portions of a single KB by performing specific reasoning procedures [9]. The motivation for investigating and developing such systems is to improve on two basic features of KR formalisms, namely *representational adequacy* and *deductive power*, by preserving the other crucial feature, i.e. *decidability*. In particular, combining DLs with HCL can easily yield to undecidability if the interface between them is not reduced. The resulting KR systems will be referred to as DL-HCL hybrid systems in the rest of the paper.

$\mathcal{AL}$-log [7] is a hybrid KR system that integrates $\mathcal{ALC}$ [30] and Datalog [4]. In particular, variables occurring in the body of rules may be constrained with $\mathcal{ALC}$ concept assertions to be used as 'typing constraints'. This makes rules applicable only to explicitly named objects. Reasoning for $\mathcal{AL}$-log knowledge bases is based on *constrained SLD-resolution*, i.e. an extension of SLD-resolution with a tableau calculus for $\mathcal{ALC}$ to deal with constraints. Constrained SLD-resolution is *decidable* and runs in single non-deterministic exponential time. Constrained SLD-refutation is a complete and sound method for answering *ground* queries.

A comprehensive study of the effects of combining DLs and HCL can be found in [17]. Here the family Carin of hybrid languages is presented. Special attention is devoted to the DL $\mathcal{ALCNR}$. The results of the study can be summarized as follows: (i) answering conjunctive queries over $\mathcal{ALCNR}$ TBoxes is decidable, (ii) query answering in a logic obtained by extending $\mathcal{ALCNR}$ with non-recursive Datalog rules, where both concepts and roles can occur in rule bodies, is also decidable, as it can be reduced to computing a union of conjunctive query answers, (iii) if rules are recursive, query answering becomes undecidable, (iv) decidability can be regained by disallowing certain combinations of constructors in the logic, and (v) decidability can be regained by requiring rules to be *role-safe*, where at least one variable from each role literal must occur in some non-DL-atom. As in $\mathcal{AL}$-log, query answering is decided using constrained resolution and a modified version of tableau calculus.

Besides decidability, another relevant issue is *DL-safeness* [27]. A safe interaction between the DL and the HCL part of an hybrid KB allows to solve the semantic mismatch between DLs and HCL, namely the OWA for DLs and the CWA for HCL[1]. In this respect, $\mathcal{AL}$-log is DL-safe whereas Carin is not.

2.2 Ontologies and Relational Learning

Two ILP frameworks have been proposed so far that adopt a hybrid DL-HCL representation for both hypotheses and background knowledge. They can be considered as previous attempts at using ontologies in Relational Learning.

The framework proposed in [29] focuses on discriminant induction and adopts the ILP setting of learning from interpretations. Hypotheses are represented as Carin-$\mathcal{ALN}$ non-recursive rules with a Horn literal in the head that plays the role of target concept. The coverage relation of hypotheses against examples adapts the usual one in learning from interpretations to the case of hybrid Carin-$\mathcal{ALN}$ BK. The generality relation between two hypotheses is defined as an extension of generalized subsumption. Procedures for testing both the coverage relation and the generality relation are based on the existential entailment algorithm of Carin. Following [29], Kietz studies the learnability of Carin-$\mathcal{ALN}$, thus providing a pre-processing method which enables ILP systems to learn Carin-$\mathcal{ALN}$ rules [16].

In [18], the representation and reasoning means come from $\mathcal{AL}$-log. Hypotheses are represented as constrained Datalog clauses that are linked, connected (or range-restricted), and compliant with the bias of Object Identity (OI)[2]. Note that this framework is general, meaning that it is valid whatever the scope of

[1] Note that the OWA and CWA have a strong influence on the results of reasoning.

[2] The OI bias can be considered as an extension of the UNA from the semantic level to the syntactic one of $\mathcal{AL}$-log. It can be the starting point for the definition of either an equational theory or a quasi-order for constrained Datalog clauses.

induction (description/prediction) is. Therefore the literal in the head of hypotheses represents a concept to be either discriminated from others (*discriminant induction*) or characterized (*characteristic induction*). The generality relation for one such hypothesis language is an adaptation of generalized subsumption [3], named $\mathcal{B}$-subsumption, to the $\mathcal{AL}$-log KR framework. It gives raise to a quasi-order and can be checked with a decidable procedure based on constrained SLD-resolution [20]. Coverage relations for both ILP settings of learning from interpretations and learning from entailment have been defined on the basis of query answering in $\mathcal{AL}$-log [19]. As opposite to [29], the framework has been partially implemented in an ILP system [22]. More precisely, an instantiation of it for the case of *characteristic induction from interpretations* has been considered. Indeed, the system supports a variant of a very popular data mining task - frequent pattern discovery - where rich prior conceptual knowledge is taken into account during the discovery process in order to find patterns at multiple levels of description granularity. The search through the space of patterns represented as unary conjunctive queries in $\mathcal{AL}$-log and organized according to $\mathcal{B}$-subsumption is performed by applying an ideal downward refinement operator [21].

3 Integrating Ontologies and Rules with $\mathcal{DL}$+log

The KR framework of $\mathcal{DL}$+log [28] allows for the tight integration of DLs [1] and Datalog$^{\neg\vee}$ [8]. More precisely, it allows a DL knowledge base (DL-KB) to be extended with *weakly-safe* Datalog$^{\neg\vee}$ rules. Note that the condition of weak safeness allows to overcome the main representational limits of the approaches based on the DL-safeness condition, e.g. the possibility of expressing conjunctive queries and unions of conjunctive queries, by keeping the integration scheme still decidable. In a certain extent, $\mathcal{DL}$+log is between $\mathcal{AL}$-log and Carin.

3.1 Syntax

We start from three mutually disjoint predicate alphabets:

- an alphabet of concept names P_C;
- an alphabet of role names P_R;
- an alphabet of Datalog predicates P_D.

We call a predicate p a *DL-predicate* if either $p \in P_C$ or $p \in P_R$. Then, we denote by $\mathcal{C}$ a countably infinite alphabet of constant names.

An *atom* is an expression of the form $p(X)$, where p is a predicate of arity n and X is a n-tuple of variables and constants. If no variable symbol occurs in X, then $p(X)$ is called a *ground atom* (or *fact*). If $p \in P_C \cup P_R$, the atom is called a *DL-atom*, while if $p \in P_D$, it is called a Datalog *atom*.

Given a description logic $\mathcal{DL}$, a $\mathcal{DL}$- KB with weakly-safe Datalog$^{\neg\vee}$ rules ($\mathcal{DL}$+log-KB for short) $\mathcal{B}$ is a pair (Σ, Π), where:

- Σ is a $\mathcal{DL}$-KB, i.e., a pair $(\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is the TBox and $\mathcal{A}$ is the ABox;
- Π is a set of Datalog$^{\neg\vee}$ rules, where each rule R has the form

$$p_1(\boldsymbol{X_1}) \vee \ldots \vee p_n(\boldsymbol{X_n}) \leftarrow$$
$$r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m}), s_1(\boldsymbol{Z_1}), \ldots, s_k(\boldsymbol{Z_k}), \neg u_1(\boldsymbol{W_1}), \ldots, \neg u_h(\boldsymbol{W_h})$$

$n \geq 0$, $m \geq 0$, $k \geq 0$, $h \geq 0$, each $p_i(\boldsymbol{X_i})$, $r_j(\boldsymbol{Y_j})$, $s_l(\boldsymbol{Z_l})$, $u_k(\boldsymbol{W_k})$ is an atom and:

- each p_i is either a DL-predicate or a Datalog predicate;
- each r_j, u_k is a Datalog predicate;
- each s_l is a DL-predicate;
- (Datalog safeness) every variable occurring in R must appear in at least one of the atoms $r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m}), s_1(\boldsymbol{Z_1}), \ldots, s_k(\boldsymbol{Z_k})$;
- (weak safeness) every head variable of R must appear in at least one of the atoms $r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m})$.

We remark that the above notion of weak safeness allows for the presence of variables that only occur in DL-atoms in the body of R. On the other hand, the notion of DL-safeness of variables adopted in previous approaches [25,23,26] can be expressed as follows: every variable of R must appear in at least one of the atoms $r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m})$. Therefore, DL-safeness forces every variable of R to occur also in the Datalog atoms in the body of R, while weak safeness allows for the presence of variables that only occur in DL-atoms in the body of R. Without loss of generality, we can assume that in a $\mathcal{DL}$+log-KB (Σ, Π) all constants occurring in Σ also occur in Π.

Example 1. Let us consider a $\mathcal{DL}$+log-KB $\mathcal{B}$ (adapted from [28]) integrating the following DL-KB Σ (ontology about persons)

[$A1$] PERSON $\sqsubseteq \exists$ FATHER$^-$.MALE
[$A2$] MALE $\sqsubseteq$ PERSON
[$A3$] FEMALE $\sqsubseteq$ PERSON
[$A4$] FEMALE $\sqsubseteq \neg$MALE
MALE(Bob)
PERSON(Mary)
PERSON(Paul)
FATHER(John,Paul)

and the following Datalog$^{\neg\vee}$ program Π (rules about students):

[$R1$] boy(X) ← enrolled(X,c1,bsc), PERSON(X), ¬girl(X)
[$R2$] girl(X) ← enrolled(X,c2,msc), PERSON(X)
[$R3$] boy(X)∨ girl(X) ← enrolled(X,c3,phd), PERSON(X)
[$R4$] FEMALE(X) ← girl(X)
[$R5$] MALE(X) ← boy(X)
[$R6$] man(X) ← enrolled(X,c3,phd), FATHER(X,Y)
enrolled(Paul,c1,bsc)
enrolled(Mary,c1,bsc)
enrolled(Mary,c2,msc)
enrolled(Bob,c3,phd)
enrolled(John,c3,phd)

Note that the rules mix DL-literals and DATALOG-literals. Notice that the variable Y in rule $R6$ is weakly-safe but not DL-safe, since Y does not occur in any DATALOG predicate in $R6$.

3.2 Semantics

For $\mathcal{DL}$+log two semantics have been defined: a first-order logic (FOL) semantics and a nonmonotonic (NM) semantics. In particular, the latter extends the stable model semantics of DATALOG$^{\neg\vee}$ [10].

According to the NM semantics, DL-predicates are still interpreted under the classical open-world assumption (OWA), while DATALOG predicates are interpreted under a closed-world assumption (CWA). Notice that, both under the FOL semantics and the NM semantics, entailment can be reduced to satisfiability, since it is possible to express constraints in the DATALOG program. More precisely, under both semantics, it is immediate to verify that (Σ, Π) entails $p(c)$ iff $(\Sigma, \Pi \cup \{\leftarrow p(c)\}$ is unsatisfiable. In a similar way, it can be seen that *conjunctive query answering* can be reduced to satisfiability in $\mathcal{DL}$+log. Consequently, Rosati [28] concentrates on the satisfiability problem in $\mathcal{DL}$+log-KBs.

It has been shown that, when the rules are positive disjunctive, i.e., there are no negated atoms in the bodies of rules, the above two semantics are equivalent with respect to the satisfiability problem. In particular, FOL-satisfiability can always be reduced (in linear time) to NM-satisfiability. Hence, the satisfiability problem under the NM semantics is in the focus of interest.

Example 2. With reference to Example 1, it can be easily verified that all NM-models for $\mathcal{B}$ satisfy the following ground atoms:

- boy(Paul) (since rule R1 is always applicable for {X/Paul} and R1 acts like a default rule, which can be read as follows: if X is a person enrolled in course c1, then X is a boy, unless we know for sure that X is a girl);
- girl(Mary) (since rule R2 is always applicable for {X/Mary});
- boy(Bob) (since rule R3 is always applicable for {X/Bob}, and, by rule R4, the conclusion girl(Bob) is inconsistent with Σ);
- MALE(Paul) (due to rule R5);
- FEMALE(Mary) (due to rule R4).

Notice that $\mathcal{B} \models_{NM}$FEMALE(Mary), while $\Sigma \not\models_{FOL}$ FEMALE(Mary). In other words, adding rules has indeed an effect on the conclusions one can draw about DL-predicates. Moreover, such an effect also holds under the FOL semantics of $\mathcal{DL}$+log-KBs, since it can be immediately verified that $\mathcal{B} \models_{FOL}$FEMALE(Mary) in this case.

3.3 Reasoning

The problem statement of satisfiability for finite $\mathcal{DL}$+log-KBs requires some preliminary definitions. We start by introducing Boolean conjunctive queries

(CQs) and Boolean unions of conjunctive queries (UCQs), and the containment problem for such queries.

A *Boolean UCQ* over a predicate alphabet P is a first-order sentence of the form $\exists \boldsymbol{X}.conj_1(\boldsymbol{X}) \vee \ldots \vee conj_n(\boldsymbol{X})$, where $\boldsymbol{X}$ is a tuple of variable symbols and each $conj_i(\boldsymbol{X})$ is a set of atoms whose predicates are in P and whose arguments are either constants or variables from $\boldsymbol{X}$. A *Boolean CQ* corresponds to a Boolean UCQ in the case when $n = 1$.

Given a $\mathcal{DL}$-TBox $\mathcal{T}$, a Boolean CQ Q_1 and a Boolean UCQ Q_2 over the alphabet $P_C \cup P_R$, Q_1 is contained in Q_2 with respect to $\mathcal{T}$, denoted by $\mathcal{T} \models Q_1 \subseteq Q_2$, iff, for every model $\mathcal{I}$ of $\mathcal{T}$, if Q_1 is satisfied in $\mathcal{I}$ then Q_2 is satisfied in $\mathcal{I}$. In the following, we call the problem of deciding $\mathcal{T} \models Q_1 \subseteq Q_2$ the *Boolean CQ/UCQ containment problem.*[3]

Besides the Boolean CQ/UCQ containment problem, it is important to clarify how the grounding operation used in stable model semantics is adapted to the $\mathcal{DL}$+log case. Given a $\mathcal{DL}$+log KB $\mathcal{B} = (\Sigma, \Pi)$, the DL-grounding of Π, denoted as $gr_p(\Pi)$, is a set of Boolean CQs. Note that $gr_p(\Pi)$ constitutes a partial grounding of the conjunctions of DL-atoms that occur in Π with respect to the constants in $\mathcal{C}_\Pi$, since the variables that only occur in DL-atoms in the body of rules arenot replaced by constants in $gr_p(\Pi)$.

The algorithm NMSAT-$\mathcal{DL}$+log for deciding NM-satisfiability of $\mathcal{DL}$+log-KBs has a very simple structure, since it decides satisfiability by looking for a guess (G_P, G_N) of the Boolean CQs in $gr_p(\Pi)$ that is consistent with the $\mathcal{DL}$-KB Σ (Boolean CQ/UCQ containment problem) and such that the Datalog$^{\neg\vee}$ program $\Pi(G_P, G_N)$ has a stable model. Notice that $\Pi(G_P, G_N)$ is a ground Datalog$^{\neg\vee}$ program over P_D, i.e. no DL-predicate occurs in such a program. The decidability of reasoning in $\mathcal{DL}$+log depends on the decidability of the Boolean CQ/UCQ containment problem in $\mathcal{DL}$.

Theorem 1. *[28] For every description logic $\mathcal{DL}$, satisfiability of $\mathcal{DL}$+log-KBs (both under FOL semantics and under NM semantics) is decidable iff Boolean CQ/UCQ containment is decidable in $\mathcal{DL}$.*

The decidability of ground query answering follows from Theorem 1.

Corollary 1. *Given a $\mathcal{DL}$+log KB (Σ, Π) and a ground atom α, $(\Sigma, \Pi) \models \alpha$ iff $(\Sigma, \Pi \cup \{\leftarrow \alpha\})$ is unsatisfiable.*

Thus ground queries can be answered by means of the abovementioned algorithm NMSAT-$\mathcal{DL}$+log for deciding NM-satisfiability of $\mathcal{DL}$+log-KBs.

From Theorem 1 and from previous results on query answering and query containment in DLs, we are able to state decidability of reasoning in several instantiations of $\mathcal{DL}$+log. In particular, for the DL $\mathcal{SHIQ}$ it is known that Boolean CQ/UCQ containment is decidable [11]. Since $\mathcal{SHIQ}$ is the most expressive DL for which this property has been proved, we consider $\mathcal{SHIQ}$+log (i.e. $\mathcal{SHIQ}$ extended with weakly-safe positive Datalog rules) as the KR framework in our study of Onto-Relational Learning.

[3] This problem was called *existential entailment* in [17].

4 Towards Onto-Relational Learning in $\mathcal{SHIQ}$+log

4.1 Defining the Learning Problem

We consider the problem of learning rules from ontologies and relational data. At this stage of work the scope of induction does not matter. So we assume that the predicate in the rule head represents a concept to be either discriminated from others (*discriminant induction*) or characterized (*characteristic induction*). Therefore the term 'observation' is to be preferred to the term 'example'. We choose to work within the setting of *learning from interpretations* [6] which requires an observation to be represented as a set of ground unit clauses.

We assume that the data are represented as a $\mathcal{SHIQ}$+log KB $\mathcal{B}$ where the intensional part $\mathcal{K}$ (i.e., the TBox $\mathcal{T}$ plus the set Π_R of rules) plays the role of *background knowledge* and the extensional part (i.e., the ABox $\mathcal{A}$ plus the set Π_F of facts) contributes to the definition of *observations*. Therefore ontologies may appear as input to the learning problem of interest.

Example 3. Suppose we have a $\mathcal{SHIQ}$+log KB (adapted from [28]) consisting of the following intensional knowledge $\mathcal{K}$:

$[A1]$ $\mathtt{RICH}\sqcap\mathtt{UNMARRIED} \sqsubseteq \exists\ \mathtt{WANTS\text{-}TO\text{-}MARRY}^{-}.\top$
$[A2]$ $\mathtt{WANTS\text{-}TO\text{-}MARRY} \sqsubseteq \mathtt{LIKES}$
$[R1]$ $\mathtt{RICH(X)} \leftarrow \mathtt{famous(X)}, \mathtt{scientist(X,us)}$

and the following extensional knowledge $\mathcal{F}$:

```
UNMARRIED(Mary)
UNMARRIED(Joe)
famous(Mary)
famous(Paul)
famous(Joe)
scientist(Mary,us)
scientist(Paul,us)
scientist(Joe,it)
```

that can be split into the sets $\mathcal{F}_{\mathtt{Paul}} = \{\mathtt{famous(Paul)}, \mathtt{scientist(Paul,us)}\}$, $\mathcal{F}_{\mathtt{Mary}} = \{\mathtt{UNMARRIED(Mary)}, \mathtt{famous(Mary)}, \mathtt{scientist(Mary,us)}\}$, and $\mathcal{F}_{\mathtt{Joe}} = \{\mathtt{UNMARRIED(Joe)}, \mathtt{famous(Joe)}, \mathtt{scientist(Joe,it)}\}$.

The language $\mathcal{L}$ of hypotheses must allow for the generation of $\mathcal{SHIQ}$+log rules starting from three disjoint alphabets $P_C(\mathcal{L}) \subseteq P_C(\mathcal{B})$, $P_R(\mathcal{L}) \subseteq P_R(\mathcal{B})$, and $P_D(\mathcal{L}) \subseteq P_D(\mathcal{B})$. More precisely, we consider linked[4] and range-restricted[5] definite clauses of the form

[4] Let H be a clause. A term t in some literal $l_i \in H$ is *linked* with linking-chain of length 0, if t occurs in *head*(H), and is linked with linking-chain of length $d+1$, if some other term in l_i is linked with linking-chain of length d. The link-depth of a term t in some $l_i \in H$ is the length of the shortest linking-chain of t. A literal $l_i \in H$ is linked if at least one of its terms is linked.

[5] A clause H is *range-restricted* or *connected* if each variable occurring in *head*(H) also occur in *body*(H).

$$p(\boldsymbol{X}) \leftarrow r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m}), s_1(\boldsymbol{Z_1}), \ldots, s_k(\boldsymbol{Z_k})$$

where $m \geq 0$, $k \geq 0$, each $p(X)$, $r_j(\boldsymbol{Y_j})$, $s_l(\boldsymbol{Z_l})$ is an atom, and:

- p is either a $\mathcal{SHIQ}$-predicate or a Datalog-predicate;
- each r_j is a Datalog-predicate;
- each s_l is a $\mathcal{SHIQ}$-predicate.

Note that the literal $p(\boldsymbol{X})$ in the head represents the target concept, denoted as c if p is a Datalog-predicate and as C if p is a $\mathcal{SHIQ}$-predicate. In the following we provide examples for these two cases of rule learning, one aimed at inducing $c(\boldsymbol{X}) \leftarrow$ rules and the other $C(\boldsymbol{X}) \leftarrow$ rules. The former kind of rule will enrich the Datalog part of the KB, whereas the latter will extend the DL part (i.e., the input ontology).

Example 4. Suppose that the Datalog-predicate happy is the target concept and the set $P_D(\mathcal{L}^{\texttt{happy}}) \cup P_C(\mathcal{L}^{\texttt{happy}}) \cup P_R(\mathcal{L}^{\texttt{happy}}) = \{\texttt{famous/1}\} \cup \{\texttt{RICH/1}\} \cup \{\texttt{WANTS-TO-MARRY/2}, \texttt{LIKES/2}\}$ provides the building blocks for the language $\mathcal{L}^{\texttt{happy}}$. The following $\mathcal{SHIQ}$+log rules

$H_1^{\texttt{happy}}$ happy(X) ← RICH(X)
$H_2^{\texttt{happy}}$ happy(X) ← famous(X)
$H_3^{\texttt{happy}}$ happy(X) ← famous(X), WANTS-TO-MARRY(Y,X)

belonging to $\mathcal{L}^{\texttt{happy}}$ can be considered hypotheses for the target concept happy. Note that $H_3^{\texttt{happy}}$ is weakly-safe.

Example 5. Suppose now that the target concept is the DL-predicate LONER. If $\mathcal{L}^{\texttt{LONER}}$ is defined over $P_D(\mathcal{L}^{\texttt{LONER}}) \cup P_C(\mathcal{L}^{\texttt{LONER}}) = \{\texttt{famous/1}, \texttt{scientist/2}\} \cup \{\texttt{UNMARRIED/1}\}$, then the following $\mathcal{SHIQ}$+log rules

$H_1^{\texttt{LONER}}$ LONER(X) ← scientist(X,Y)
$H_2^{\texttt{LONER}}$ LONER(X) ← scientist(X,Y), UNMARRIED(X)
$H_3^{\texttt{LONER}}$ LONER(X) ← scientist(X,Y), famous(X)

belong to $\mathcal{L}^{\texttt{LONER}}$ and represent hypotheses for the target concept LONER.

4.2 Testing the Hypothesis Coverage of Observations

An observation $o_i \in O$ is represented as a couple $(p(\boldsymbol{a_i}), \mathcal{F}_i)$ where $\mathcal{F}_i$ is a set containing ground facts concerning the individual $\boldsymbol{a_i}$. We assume $\mathcal{K} \cap O = \emptyset$.

Definition 1. *Let $H \in \mathcal{L}$ be a hypothesis, $\mathcal{K}$ a background knowledge and $o_i \in O$ an observation. We say that H* covers *o_i* under interpretations w.r.t. *$\mathcal{K}$ iff $\mathcal{K} \cup \mathcal{F}_i \cup H \models p(a_i)$.*

Note that the coverage test can be reduced to query answering in $\mathcal{SHIQ}$+log KBs which in its turn can be reformulated as a satisfiability problem of the KB.

Example 6. The hypothesis $H_3^{\texttt{happy}}$ mentioned in Example 4 covers the observation $o_{\texttt{Mary}} = (\texttt{happy(Mary)}, \mathcal{F}_{\texttt{Mary}})$ because $\mathcal{K} \cup \mathcal{F}_{\texttt{Mary}} \cup H_3^{\texttt{happy}} \models \texttt{happy(Mary)}$. Indeed, all NM-models for $\mathcal{B} = \mathcal{K} \cup \mathcal{F}_{\texttt{Mary}} \cup H_3^{\texttt{happy}}$ satisfy:

- $\texttt{famous(Mary)}$ (trivial!);
- $\exists\ \texttt{WANTS-TO-MARRY}^{-}.\top\texttt{(Mary)}$, due to the axiom A1 and to the fact that both $\texttt{RICH(Mary)}$ and $\texttt{UNMARRIED(Mary)}$ hold in every model of $\mathcal{B}$;
- $\texttt{happy(Mary)}$, due to the above conclusions and to the rule R1. Indeed, since $\exists\texttt{WANTS-TO-MARRY}^{-}.\top\texttt{(Mary)}$ holds in every model of $\mathcal{B}$, it follows that in every model there exists a constant $\texttt{x}$ such that $\texttt{WANTS-TO-MARRY(x,Mary)}$ holds in the model, consequently from rule R1 it follows that $\texttt{happy(Mary)}$ also holds in the model.

Note that $H_3^{\texttt{happy}}$ does not cover the observations $o_{\texttt{Joe}} = (\texttt{happy(Joe)}, \mathcal{F}_{\texttt{Joe}})$ and $o_{\texttt{Paul}} = (\texttt{happy(Paul)}, \mathcal{F}_{\texttt{Paul}})$. More precisely, $\mathcal{K} \cup \mathcal{F}_{\texttt{Joe}} \cup H_3^{\texttt{happy}} \not\models \texttt{happy(Joe)}$ because $\texttt{scientist(Joe,it)}$ holds in every model of $\mathcal{B} = \mathcal{K} \cup \mathcal{F}_{\texttt{Joe}} \cup H_3^{\texttt{happy}}$, thus making the rule R1 not applicable for $\{\texttt{X/Joe}\}$, therefore $\texttt{RICH(Joe)}$ not derivable. Finally, $\mathcal{K} \cup \mathcal{F}_{\texttt{Paul}} \cup H_3^{\texttt{happy}} \not\models \texttt{happy(Paul)}$ because $\texttt{UNMARRIED(Paul)}$ is not forced to hold in every model of $\mathcal{B} = \mathcal{K} \cup \mathcal{F}_{\texttt{Paul}} \cup H_3^{\texttt{happy}}$, therefore $\exists\texttt{WANTS-TO-MARRY}^{-}.\top\texttt{(Paul)}$ is not forced by A1 to hold in every such model.

It can be proved that $H_1^{\texttt{happy}}$ covers $o_{\texttt{Mary}}$ and $o_{\texttt{Paul}}$, while $H_2^{\texttt{happy}}$ all the three observations.

Example 7. With reference to Example 5, the hypothesis $H_3^{\texttt{LONER}}$ covers the observation $o_{\texttt{Mary}} = (\texttt{LONER(Mary)}, \mathcal{F}_{\texttt{Mary}})$ because all NM-models for $\mathcal{B} = \mathcal{K} \cup \mathcal{F}_{\texttt{Mary}} \cup H_3^{\texttt{LONER}}$ do satisfy $\texttt{scientist(Mary,us)}$ and $\texttt{famous(Mary)}$. Note that it covers the observations $o_{\texttt{Paul}} = (\texttt{LONER(Paul)}, \mathcal{F}_{\texttt{Paul}})$ and $o_{\texttt{Joe}} = (\texttt{LONER(Joe)}, \mathcal{F}_{\texttt{Joe}})$ for analogous reasons. It can be proved that $H_2^{\texttt{LONER}}$ covers $o_{\texttt{Mary}}$ and $o_{\texttt{Joe}}$ while $H_1^{\texttt{LONER}}$ all three observations.

4.3 Structuring the Hypothesis Space

The definition of a generality relation for hypotheses in $\mathcal{L}$ can disregard neither the peculiarities of $\mathcal{SHIQ}$+log nor the methodological apparatus of ILP. Roughly speaking, we propose to adapt generalized subsumption [3] to the case of $\mathcal{SHIQ}$+log rules. The resulting generality relation will be called *$\mathcal{K}$-subsumption*, briefly $\succeq_{\mathcal{K}}$, from now on. First we provide a model-theoretic definition of $\succeq_{\mathcal{K}}$ based on the notion of covering[6], then a more operational characterization that relies on the reasoning tasks known for $\mathcal{SHIQ}$+log.

Definition 2. *Let $H_1, H_2 \in \mathcal{L}$ be two hypotheses and $\mathcal{K}$ a background knowledge. We say that $H_1 \succeq_{\mathcal{K}} H_2$ if for every model $\mathcal{J}$ of $\mathcal{K}$ and every ground atom α such that H_2 covers α under $\mathcal{J}$, we have that H_1 covers α under $\mathcal{J}$.*

[6] A definite clause C *covers* a ground atom α under an interpretation $\mathcal{J}$ if there is a ground substitution θ for C ($C\theta$ is ground) such that $body(C)\theta$ is true under $\mathcal{J}$ and $head(C)\theta = \alpha$. This notion can be easily extended to the case of $\mathcal{SHIQ}$+log rules, assuming that α can be either a $\mathcal{SHIQ}$-atom or a DATALOG-atom.

It can be proved that $\succeq_{\mathcal{K}}$ is a quasi-order (i.e. it is a reflexive and transitive relation) for $\mathcal{SHIQ}$+log rules.

A procedure for deciding $\succeq_{\mathcal{K}}$ can be derived from the following characterization of $\mathcal{K}$-subsumption.

Definition 3. *Let $H_1, H_2 \in \mathcal{L}$ be two hypotheses standardized apart, $\mathcal{K}$ a background knowledge, and σ a Skolem substitution*[7] *for H_2 with respect to $\{H_1\} \cup \mathcal{K}$. We say that $H_1 \succeq_{\mathcal{K}} H_2$ iff there exists a ground substitution θ for H_1 such that (i) $head(H_1)\theta = head(H_2)\sigma$ and (ii) $\mathcal{K} \cup body(H_2)\sigma \models body(H_1)\theta$.*

Note that condition (ii) is a variant of the Boolean CQ/UCQ containment problem because $body(H_2)\sigma$ and $body(H_1)\theta$ are both Boolean CQs. The difference between (ii) and the original formulation of the problem is that $\mathcal{K}$ encompasses not only a TBox but also a set of rules. Nonetheless this variant can be reduced to the satisfiability problem for finite $\mathcal{SHIQ}$+log KBs. Indeed the skolemization of $body(H_2)$ allows to reduce the Boolean CQ/UCQ containment problem to a CQ answering problem. Due to the aforementioned link between CQ answering and satisfiability, checking (ii) can be reformulated as proving that the KB $(\mathcal{T}, \Pi_R \cup body(H_2)\sigma \cup \{\leftarrow body(H_1)\theta\})$ is unsatisfiable. Once reformulated this way, (ii) can be solved by applying the algorithm NMSAT-$\mathcal{DL}$+log.

Example 8. Let us consider the hypotheses

$H_1^{\texttt{happy}}$ `happy(A)` $\leftarrow$ `RICH(A)`
$H_2^{\texttt{happy}}$ `happy(X)` $\leftarrow$ `famous(X)`

reported in Example 4 up to variable renaming. We want to check whether $H_1^{\texttt{happy}} \succeq_{\mathcal{K}} H_2^{\texttt{happy}}$ holds. Let $\sigma = \{\texttt{X}/\texttt{a}\}$ a Skolem substitution for $H_2^{\texttt{happy}}$ with respect to $\mathcal{K} \cup H_1^{\texttt{happy}}$ and $\theta = \{\texttt{A}/\texttt{a}\}$ a ground substitution for $H_1^{\texttt{happy}}$. The condition (i) is immediately verified. The condition (ii) $\mathcal{K} \cup \{\texttt{famous(a)}\} \models \texttt{RICH(a)}$ is nothing else that a ground query answering problem in $\mathcal{SHIQ}$+log. It can be proved that the query `RICH(a)` can not be satisfied because the rule R1 is not applicable for `a`. Thus, $H_1^{\texttt{happy}} \not\succeq_{\mathcal{K}} H_2^{\texttt{happy}}$. Since $H_2^{\texttt{happy}} \not\succeq_{\mathcal{K}} H_1^{\texttt{happy}}$, the two hypotheses are incomparable with respect to $\mathcal{K}$-subsumption. Conversely, it can be proved that $H_2^{\texttt{happy}} \succeq_{\mathcal{K}} H_3^{\texttt{happy}}$ but not viceversa.

Example 9. Let us consider the hypotheses

$H_1^{\texttt{LONER}}$ `LONER(A)` $\leftarrow$ `scientist(A,B)`
$H_2^{\texttt{LONER}}$ `LONER(X)` $\leftarrow$ `scientist(X,Y),UNMARRIED(X)`

reported in Example 5 up to variable renaming. We want to check whether $H_1^{\texttt{LONER}} \succeq_{\mathcal{K}} H_2^{\texttt{LONER}}$ holds. Let $\sigma = \{\texttt{X}/\texttt{a}, \texttt{Y}/\texttt{b}\}$ a Skolem substitution for $H_2^{\texttt{LONER}}$ with respect to $\mathcal{K} \cup H_1^{\texttt{LONER}}$ and $\theta = \{\texttt{A}/\texttt{a}, \texttt{B}/\texttt{b}\}$ a ground substitution for $H_1^{\texttt{LONER}}$. The condition (i) is immediately verified. The condition

[7] Let $\mathcal{B}$ be a clausal theory and H be a clause. Let $X_1, \ldots, X_n$ be all the variables appearing in H, and $a_1, \ldots, a_n$ be distinct constants (individuals) not appearing in $\mathcal{B}$ or H. Then the substitution $\{X_1/a_1, \ldots, X_n/a_n\}$ is called a *Skolem substitution* for H w.r.t. $\mathcal{B}$.

(ii) $\mathcal{K} \cup \{\texttt{scientist(a,b)}, \texttt{UNMARRIED(a)}\} \models \{\texttt{scientist(a,b)}\}$

is a ground query answering problem in $\mathcal{SHIQ}$+log. It can be easily proved that all NM-models for $\mathcal{K} \cup \{\texttt{scientist(a,b)}, \texttt{UNMARRIED(a)}\}$ satisfy $\texttt{scientist(a,b)}$. Thus, $H_1^{\texttt{LONER}} \succeq_{\mathcal{K}} H_2^{\texttt{LONER}}$. The viceversa does not hold.

It is straightforward to prove that the decidability of $\mathcal{K}$-subsumption follows from the decidability of $\mathcal{SHIQ}$+log.

4.4 Searching the Hypothesis Space

The space $(\mathcal{L}, \succeq_{\mathcal{K}})$ is a quasi-ordered set, therefore it can be searched by refinement operators.

Definition 4. *Let $\mathcal{L}$ be a language of hypotheses built out of the three finite and disjoint alphabets $P_C(\mathcal{L})$, $P_R(\mathcal{L})$, and $P_D(\mathcal{L})$, and*

$$p(\boldsymbol{X}) \leftarrow r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m}), s_1(\boldsymbol{Z_1}), \ldots, s_k(\boldsymbol{Z_k})$$

be a hypothesis belonging to $\mathcal{L}$. A downward refinement operator *ρ for $(\mathcal{L}, \succeq_{\mathcal{K}})$ is defined such that the set $\rho(H)$ contains all $H_s \in \mathcal{L}$ that can be obtained from H by applying one of the following refinement rules:*

$\langle AddDataLit\rangle$ $body(H_s) = body(H) \cup \{r_{m+1}(\boldsymbol{Y_{m+1}})\}$, where $r_{m+1} \in P_D$ and $r_{m+1}(\boldsymbol{Y_{m+1}}) \notin body(H)$.
$\langle AddOntoLit\rangle$ $body(H_s) = body(H) \cup \{s_{k+1}(\boldsymbol{Z_{k+1}})\}$, where $s_{k+1} \in P_C \cup P_R$ and it does not exist any $s_l \in body(H)$ such that $s_{k+1} \sqsubseteq s_l$.
$\langle SpecOntoLit\rangle$ $body(H_s) = (body(H) \setminus \{s_l(\boldsymbol{Z_l})\}) \cup s'_l(\boldsymbol{Z_l})$ where $s'_l \in P_C \cup P_R$ and $s'_l \sqsubseteq s_l$.

An upward refinement operator *δ for $(\mathcal{L}, \succeq_{\mathcal{K}})$ is defined such that the set $\delta(H)$ contains all $H_g \in \mathcal{L}$ that can be obtained from H by applying one of the following refinement rules:*

$\langle DelDataLit\rangle$ $body(H_g) = body(H) \setminus \{r_j(\boldsymbol{Y_j})\}$.
$\langle DelOntoLit\rangle$ $body(H_g) = body(H) \setminus \{s_l(\boldsymbol{Z_l})\}$.
$\langle GenOntoLit\rangle$ $body(H_g) = (body(H) \setminus \{s_l(\boldsymbol{Z_l})\}) \cup s'_l(\boldsymbol{Z_l})$ where $s'_l \in P_C \cup P_R$ and $s_l \sqsubseteq s'_l$.

All the rules of ρ (resp. δ) are correct, i.e. the H_s's (resp. H_g's) obtained by applying any of the rules of ρ (resp. δ) to $H \in \mathcal{L}$ are such that $H \succeq_{\mathcal{K}} H_s$ (resp. $H_g \succeq_{\mathcal{K}} H$). This can be proved intuitively by observing that they act only on *body*(H). Thus condition (i) of Definition 2 is satisfied. Furthermore, it is straightforward to notice that the application of any of the rules of ρ to H reduces (resp. augments) the number of models of H. In particular, as for $\langle SpecOntoLit\rangle$ and $\langle GenOntoLit\rangle$, this intuition follows from the semantics of $\mathcal{SHIQ}$. So condition (ii) also is fulfilled.

Example 10. With reference to Example 5, the hypotheses $H_2^{\texttt{LONER}}$ and $H_3^{\texttt{LONER}}$ are obtained from $H_1^{\texttt{LONER}}$ by applying the refinement rules $\langle AddDataLit \rangle$ and $\langle AddOntoLit \rangle$ respectively. The rule $\langle AddOntoLit \rangle$ also specializes the hypothesis $H_2^{\texttt{happy}}$ of Example 4 into $H_3^{\texttt{happy}}$, which in its turn can be obtained as refinement via $\langle SpecOntoLit \rangle$ from the following clause

$$H_4^{\texttt{happy}} = \texttt{happy(X)} \leftarrow \texttt{famous(X)}, \texttt{LIKES(Y,X)}$$

also belonging to $\mathcal{L}^{\texttt{happy}}$.

If the bottom-up search is adopted, we have that $H_1^{\texttt{LONER}}$ can be obtained from $H_2^{\texttt{LONER}}$ by applying $\langle DelDataLit \rangle$ and from $H_3^{\texttt{LONER}}$ by applying $\langle DelOntoLit \rangle$. Also, $H_3^{\texttt{happy}}$ can be generalized into $H_2^{\texttt{happy}}$ or $H_4^{\texttt{happy}}$ according to whether $\langle DelOntoLit \rangle$ or $\langle GenOntoLit \rangle$ is applied.

Ideal refinement operators have been proven not to exist for clausal languages ordered by θ-subsumption or stronger orders but can be approximated by dropping the requirement of properness or by bounding the language. We choose the latter option because it guarantees that, if $(\mathcal{L}, \succeq)$ is a quasi-ordered set, $\mathcal{L}$ is finite and $\succeq$ is decidable, then there always exists an ideal refinement operator for $(\mathcal{L}, \succeq)$. In our case, since $\succeq_{\mathcal{K}}$ is a decidable quasi-order, we only need to bound $\mathcal{L}$ in a suitable manner. From Definition 4 we know that the alphabets $P_C(\mathcal{L})$, $P_R(\mathcal{L})$, and $P_D(\mathcal{L})$ are finite. Having Datalog as basis for the CL part of $\mathcal{SHIQ}$+log avoids the generation of infinite terms. Yet, the expressive power of $\mathcal{SHIQ}$+log requires several other bounds to be imposed on $\mathcal{L}$ in order to guarantee its finiteness. It is necessary to introduce a complexity measure for $\mathcal{SHIQ}$+log rules, as a pair of two different coordinates. Considering that the complexity of a $\mathcal{SHIQ}$+log rule resides in its body, the former coordinate is the size (i.e. the difference between the number of symbol occurrences and the number of distinct variables) of the biggest literal in $body(H)$, while the latter is the number of literals in $body(H)$. To keep $\mathcal{L}$ finite, we need first to set a maximum value for these two coordinates. Second, it is necessary to set the maximum number of specialization/generalization steps of the DL literals so that the search in the ontology is also depth-bounded.

Ideal refinement operators are mainly of theoretical interest, because in practice they are often very inefficient. More constructive - though possibly improper - refinement operators are usually to be preferred over ideal ones. Optimal refinement operators can be easily derived from those proposed in this paper.

5 Conclusions and Future Work

Using ontologies in Relational Learning raises several challenges to the ILP community. The approach followed in this paper aims at extending Relational Learning to account for ontologies in a clear, well-founded and systematic way analogously to what has been done in Statistical Relational Learning. More precisely, our contribution to the upcoming Onto-Relational Learning adopts a decidable KR framework, $\mathcal{SHIQ}$+log, that is the most powerful among the ones

Table 1. Comparison between ILP frameworks for Onto-Relational Learning

	Learning in Carin-$\mathcal{ALN}$ [29]	**Learning in $\mathcal{AL}$-log [18]**	**Learning in $\mathcal{SHIQ}$+log**
prior knowledge	Carin-$\mathcal{ALN}$ KB	$\mathcal{AL}$-log KB	$\mathcal{SHIQ}$+log KB
ontology lang.	$\mathcal{ALN}$	$\mathcal{ALC}$	$\mathcal{SHIQ}$
hypothesis lang.	Carin-$\mathcal{ALN}$ non-recursive rules	constrained Datalog clauses	$\mathcal{SHIQ}$+log non-recursive rules
target concept	Horn literal	Datalog literal	$\mathcal{SHIQ}$/Datalog literal
observations	interpretations	interpretations/implications	interpretations
induction	predictive	predictive/descriptive	predictive/descriptive
generality order	extension of [3] to Carin-$\mathcal{ALN}$	extension of [3] to $\mathcal{AL}$-log	extension of [3] to $\mathcal{SHIQ}$+log
coverage test	Carin-$\mathcal{ALN}$ query answering	$\mathcal{AL}$-log query answering	$\mathcal{SHIQ}$+log query answering
ref. operators	no	downward	downward
implementation	no	partially	no
application	no	yes	no

currently available for the integration of DLs and CLs. It differs from the related proposals [29] and [18] (illustrated in Section 2.2) in several respects as summarized in Table 1, notably the following three. First, it relies on a more expressive DL (i.e., $\mathcal{SHIQ}$) - thus getting closer to the current standard ontology languages. Second, it allows for inducing a definition for DL concepts (i.e., rules with a $\mathcal{SHIQ}$ literal in the head) - thus having ontology elements not only as input but also as output of the learning process. Third, it adopts a tighter form of integration between the DL part and the CL part of rules (i.e., the weakly-safe one) - thus enabling the decidability of the Boolean CQ/UCQ containment problem. Similarities also emerge from Table 1 such as the definition of a *semantic* generality relation for hypotheses in order to accommodate ontologies in ILP. Note that generalized subsumption is chosen for adaptation because in all three ILP frameworks definite clauses, though enriched with DL literals, are still used.

Though this paper can be considered as a feasibility study of learning in $\mathcal{SHIQ}$+log, yet it provides the core ingredients of an ILP framework for Onto-Relational Learning. We would like to emphasize that these ingredients will be still valid for any other upcoming decidable instantiation of $\mathcal{DL}$+log, provided that positive Datalog is still considered. Anyway, having preliminary results for learning in $\mathcal{SHIQ}$+log is already valuable from an application viewpoint. Indeed, $\mathcal{SHIQ}$ was the starting point for the design of the ontology language OWL for the Semantic Web [14]. Also, the Semantic Web offers several use cases for rules (built on top of OWL ontologies) among which we can choose in order to see our ILP approach to Onto-Relational Learning at work. As next step towards any practice, we plan to define ILP algorithms starting from the ingredients identified in this paper. Also we would like to investigate the impact of having Datalog$^{\neg\vee}$ both in the language of hypotheses and in the language for the background theory. The inclusion of the nonmonotonic features of $\mathcal{SHIQ}$+log *full* will strengthen the ability of our ILP framework to deal with incomplete knowledge by performing some form of commonsense reasoning. One such ability can turn out to be useful in application domains, such as the Semantic Web, that require reasoning with uncertainty and under inconsistency. Speaking of which and as a final remark, we would like to point out that Onto-Relational Learning is not alternative but complementary to Statistical Relational Learning.

Acknowledgements. We are grateful to Riccardo Rosati for his precious advice on $\mathcal{DL}$+log and Diego Calvanese for his valid support on the Boolean CQ/UCQ containment problem. Also we acknowledge the financial support of the COFIN-PRIN 2006 project "Learning Hierarchical, Abstract Models from Temporal or Spatial Data" funded by the *Ministero dell'Istruzione, dell'Università, e della Ricerca Scientifica* and the project "Modelli e tecniche di Apprendimento Statistico Relazionale" funded by the *Università degli Studi di Bari*.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
2. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82(1–2), 353–367 (1996)
3. Buntine, W.: Generalized subsumption and its application to induction and redundancy. Artificial Intelligence 36(2), 149–176 (1988)
4. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Transactions on Knowledge and Data Engineering 1(1), 146–166 (1989)
5. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (1990)
6. De Raedt, L., Džeroski, S.: First order jk-clausal theories are PAC-learnable. Artificial Intelligence 70, 375–392 (1994)
7. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: $\mathcal{AL}$-log: Integrating Datalog and Description Logics. Journal of Intelligent Information Systems 10(3), 227–252 (1998)
8. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems 22(3), 364–418 (1997)
9. Frisch, A.M., Cohn, A.G.: Thoughts and afterthoughts on the 1988 workshop on principles of hybrid reasoning. AI Magazine 11(5), 84–87 (1991)
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–386 (1991)
11. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. Journal of Artificial Intelligence Research 31, 151–198 (2008)
12. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering. Springer, Heidelberg (2004)
13. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220 (1993)
14. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
15. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3), 239–263 (2000)
16. Kietz, J.-U.: Learnability of description logic programs. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 117–132. Springer, Heidelberg (2003)

17. Levy, A.Y., Rousset, M.-C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104, 165–209 (1998)
18. Lisi, F.A.: Building Rules on Top of Ontologies for the Semantic Web with Inductive Logic Programming. Theory and Practice of Logic Programming 8(03), 271–300 (2008)
19. Lisi, F.A., Esposito, F.: Efficient Evaluation of Candidate Hypotheses in $\mathcal{AL}$-log. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 216–233. Springer, Heidelberg (2004)
20. Lisi, F.A., Malerba, D.: Bridging the Gap between Horn Clausal Logic and Description Logics in Inductive Learning. In: Cappelli, A., Turini, F. (eds.) AI*IA 2003. LNCS, vol. 2829, pp. 49–60. Springer, Heidelberg (2003)
21. Lisi, F.A., Malerba, D.: Ideal Refinement of Descriptions in $\mathcal{AL}$-log. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 215–232. Springer, Heidelberg (2003)
22. Lisi, F.A., Malerba, D.: Inducing Multi-Level Association Rules from Multiple Relations. Machine Learning 55, 175–210 (2004)
23. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. Journal on Web Semantics 3(1), 41–60 (2005)
24. Reiter, R.: Equality and domain closure in first order databases. Journal of ACM 27, 235–249 (1980)
25. Rosati, R.: Towards expressive KR systems integrating DATALOG and description logics: preliminary report. In: Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., Patel-Schneider, P.F. (eds.) Proceedings of the 1999 International Workshop on Description Logics (DL 1999). CEUR Workshop Proceedings (1999)
26. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. Journal of Web Semantics 3(1) (2005)
27. Rosati, R.: Semantic and computational advantages of the safe integration of ontologies and rules. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 50–64. Springer, Heidelberg (2005)
28. Rosati, R.: $\mathcal{DL}$+log: Tight integration of description logics and disjunctive datalog. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. of Tenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 68–78. AAAI Press, Menlo Park (2006)
29. Rouveirol, C., Ventos, V.: Towards Learning in CARIN-$\mathcal{ALN}$. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 191–208. Springer, Heidelberg (2000)
30. Schmidt-Schauss, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)

A Disjunctive Datalog

Disjunctive DATALOG (DATALOG$^{\vee}$) is a powerful database query language based on disjunctive logic programming [8]. Briefly, it is a variant of DATALOG where disjunctions may appear in the rule heads. Advanced versions (DATALOG$^{\neg\vee}$) also allow for negation in the bodies, which can be handled according to a semantics for negation in disjunctive logic programming.

More formally, a DATALOG$^{\neg\vee}$ *rule* R is an expression of the form

$$p_1(\boldsymbol{X_1}) \vee \ldots \vee p_n(\boldsymbol{X_n}) \leftarrow r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m}), \neg u_1(\boldsymbol{W_1}), \ldots, \neg u_h(\boldsymbol{W_h})$$

such that $n \geq 0$, $m \geq 0$, $h \geq 0$, each $p_i(\boldsymbol{X_i})$, $r_j(\boldsymbol{Y_j})$, $u_k(\boldsymbol{W_k})$ is an atom and every variable occurring in R must appear in at least one of the atoms $r_1(\boldsymbol{Y_1}), \ldots, r_m(\boldsymbol{Y_m})$. This last condition is known as the Datalog *safeness* condition for variables. The variables occurring in the atoms $p_1(\boldsymbol{X_1}) \vee \ldots \vee p_n(\boldsymbol{X_n})$ are called the head variables of R. If $n = 0$, we call R a constraint.

A Datalog$^{\neg\vee}$ *program* Π is a set of Datalog$^{\neg\vee}$ rules. If, for all $R \in \Pi$, $n \leq 1$, Π is called a Datalog$^{\neg}$ program. If, for all $R \in \Pi$, $h = 0$, Π is called a positive Datalog$^{\vee}$ program. If, for all $R \in \Pi$, $n \leq 1$ and $h = 0$, Π is called a positive Datalog program. If there are no occurrences of variable symbols in Π, Π is called a *ground* program.

Defining the semantics of a Datalog$^{\neg\vee}$ program is complicated by the presence of disjunction in the rules' heads because it makes the underlying disjunctive logic programming inherently nonmonotonic, i.e. new information can invalidate previous conclusions. Among the many alternatives, one widely accepted semantics for Datalog$^{\neg\vee}$ is the extension to the disjunctive case of the *stable model semantics* [10]. According to this semantics, a Datalog$^{\neg\vee}$ program may have several alternative models (but possibly none), each corresponding to a possible view of the reality.

B Description Logics

DLs are a family of decidable FOL fragments that allow for the specification of knowledge in terms of classes (*concepts*), binary relations between classes (*roles*), and instances (*individuals*) [2]. Complex concepts can be defined from atomic concepts and roles by means of constructors (see Table 2). E.g., concept descriptions in the basic DL $\mathcal{AL}$ are formed according to only the constructors of atomic negation, concept conjunction, value restriction, and limited existential restriction. The DLs $\mathcal{ALC}$ and $\mathcal{ALN}$ are members of the $\mathcal{AL}$ family. The former extends $\mathcal{AL}$ with (arbitrary) concept negation (also called complement and equivalent to having both concept union and full existential restriction), whereas the latter with number restriction. The DL $\mathcal{ALCNR}$ adds to the constructors inherited from $\mathcal{ALC}$ and $\mathcal{ALN}$ a further one: role intersection (see Table 2). On the contrary, in the DL $\mathcal{SHIQ}$ [15] it is allowed to invert roles and to express qualified number restrictions of the form $\geq nS.C$ and $\leq nS.C$ where S is a simple role (see Table 2).

A DL knowledge base (KB) can state both is-a relations between concepts (*axioms*) and instance-of relations between individuals (resp. couples of individuals) and concepts (resp. roles) (*assertions*). Concepts and axioms form the so-called TBox whereas individuals and assertions form the so-called ABox[8]. A $\mathcal{SHIQ}$ KB encompasses also a role box. A role box (RBox) $\mathcal{R}$ consists of a finite set of transitivity axioms, and role inclusion axioms of the form $R \sqsubseteq S$ where R and S are abstract roles. Therefore hierarchies can be defined over not only

[8] When a DL-based ontology language is adopted, an ontology is nothing else than a TBox. If the ontology is populated, it corresponds to a whole DL KB, i.e. encompassing also an ABox.

concepts but also roles. The semantics of DLs is defined through a mapping to FOL. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a DL KB consists of a domain $\Delta^{\mathcal{I}}$ and a mapping function $\cdot^{\mathcal{I}}$. In particular, individuals are mapped to elements of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (*Unique Names Assumption* (UNA) [24]). Yet in $\mathcal{SHIQ}$ UNA does not hold. Thus individual equality (inequality) assertions may appear in a $\mathcal{SHIQ}$ KB (see Table 2). As a consequence, the (Boolean) CQ/UCQ containment problem for $\mathcal{SHIQ}$ boils down to the (Boolean) CQ/UCQ answering problem. Also the KB represents many different interpretations, i.e. all its models. This is coherent with the *Open World Assumption* (OWA) that holds in FOL semantics. The main reasoning task for a DL KB is the *consistency check* that is performed by applying decision procedures based on tableau calculus.

Table 2. Syntax and semantics of DLs

bottom (resp. top) concept	$\bot$ (resp. $\top$)	$\emptyset$ (resp. $\Delta^{\mathcal{I}}$)
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
(abstract) role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
(abstract) inverse role	R^-	$(R^{\mathcal{I}})^-$
(abstract) individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
concept intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
concept union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
at least number restriction	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \|\{y \| (x,y) \in R^{\mathcal{I}}\}\| \geq n\}$
at most number restriction	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \|\{y \| (x,y) \in R^{\mathcal{I}}\}\| \leq n\}$
at least qualif. number restriction	$\geq nS.C$	$\{x \in \Delta^{\mathcal{I}} \mid \|\{y \in C^{\mathcal{I}} \| (x,y) \in S^{\mathcal{I}}\}\| \geq n\}$
at most qualif. number restriction	$\leq nS.C$	$\{x \in \Delta^{\mathcal{I}} \mid \|\{y \in C^{\mathcal{I}} \| (x,y) \in S^{\mathcal{I}}\}\| \leq n\}$
role intersection	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
concept equivalence axiom	$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
concept subsumption axiom	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
role equivalence axiom	$R_1 \equiv R_2$	$R_1^{\mathcal{I}} = R_2^{\mathcal{I}}$
role inclusion axiom	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
concept assertion	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$\langle a, b \rangle : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
individual equality assertion	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality assertion	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

L-Modified ILP Evaluation Functions for Positive-Only Biological Grammar Learning

Thierry Mamer[1,⋆], Christopher H. Bryant[2], and John McCall[1]

[1] School of Computing, The Robert Gordon University,
St. Andrews Street, AB25 1HG, Aberdeen, Scotland, UK
tm@comp.rgu.ac.uk
http://www.comp.rgu.ac.uk

[2] School of Computing, Science and Engineering, Newton Building,
University of Salford, Salford, Greater Manchester, M5 4WT, UK

Abstract. We identify a shortcoming of a standard positive-only clause evaluation function within the context of learning biological grammars. To overcome this shortcoming we propose L-modification, a modification to this evaluation function such that the lengths of individual examples are considered. We use a set of bio-sequences known as neuropeptide precursor middles (NPP-middles). Using L-modification to learn from these NPP-middles results in induced grammars that have a better performance than that achieved when using the standard positive-only clause evaluation function. We also show that L-modification improves the performance of induced grammars when learning on short, medium or long NPPs-middles. A potential disadvantage of L-modification is discussed. Finally, we show that, as the limit on the search space size increases, the greater is the increase in predictive performance arising from L-modification.

Keywords: Inductive Logic Programming (ILP), Biological Grammar Induction, Machine Learning, Minimum Description Length (MDL).

1 Introduction

This work aims to improve the automated learning of biological grammars using Inductive Logic Programming (ILP) tools.

1.1 Biological Grammars

Biological grammars (BG) are patterns in the form of grammars that model biological sequences: among others, protein sequences. Linguistic approaches can be used for representing the structure of proteins [11] because their primary structure can be represented as a sequence of characters from a well defined chemical

[⋆] Corresponding Author: Thierry Mamer, School of Computing, The Robert Gordon University, St. Andrews Street, AB25 1HG, Aberdeen, Scotland, United Kingdom; webpage: http://www.comp.rgu.ac.uk/staff/tm/

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 176–191, 2008.

alphabet of only 20 different amino-acids (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y). These sequences can be of any length, from very small, up to hundreds of characters long. Formal grammars can define dependencies in biological sequences because of their declarative and hierarchical nature (e.g. biological sequences folding up in three dimensional space lead to dependencies between distant parts). Using grammars to model biological sequences brings two main advantages to the biologist: first, grammars can be used to annotate sequences whose function is yet unknown and thus suggest a likely function; second, because the grammar structure represents common points between sequences of similar functions, they could help biologists to understand biological functions. See Figure 1 for a basic example of a BG parsing a protein sequence. BGs can take several forms depending on the approach taken. In our experiments, BGs take the form of *context free grammars* (CFG) (see Section 2.2) and describe a specific family of proteins (see Section 2.1).

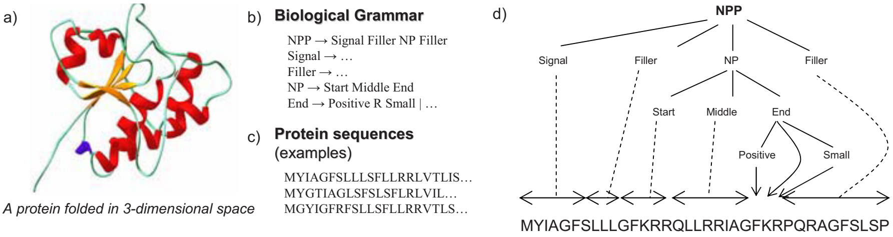


Fig. 1. Figure 1a) shows an example of how a protein might look like in 3 dimensional space. Figure 1c) shows a few examples of protein sequences. These are sequences of the amino acids which make up the proteins, but ignoring the 3 dimensional structure. Figure 1b) gives an indication of what a biological grammar describing an Neuropeptide Precursor Protein (NPP) might look like; a set of production rules that can parse a protein sequence. Figure 1d) shows a very basic example of a parse tree, illustrating how an NPP sequence could be parsed by an NPP grammar.

In our experiments we learn BGs from examples of protein sequences belonging to a certain family of proteins (see Section 2.1). Protein sequences generally have highly variable length, which is what sets our task of learning BG apart from most traditional ILP learning tasks. It is this variable length of examples in the training data given to the ILP system that is the main focus of this work.

1.2 Biological Grammar Learning with ILP

Muggelton et al [8] first investigated Chomski-like grammar representations for learning cost-effective, comprehensible predictors of members of biological sequence families. They used the ILP tool CProgol [6] to learn grammars describing neuropeptide precursors (NPPs). Their best predictor made the search for novel NPPs more than one hundred times more efficient than randomly selecting proteins for synthesis and testing them for biological activity. Our work takes its roots in the approach of [8] as we use *definite clause grammars* DCG [9] (see Section 2.2) and we also use a subset of the NPP dataset used in [8](see Section 2.1). Bryant et al. [1,3,2], which also takes its roots in the approach of [8] used the ILP tool Aleph [12] because Aleph is a modular system, it is easier to modify and gives a large number of options. Our work also uses Aleph as we also need the ability to customize the clause evaluation functions and the clause coverage computation (see Section 3.4).

1.3 Hypothesis

The hypothesis that we want to test in this work is as follows:

When using the generic ILP system Aleph to learn biological grammars describing proteins of the NPP family, then the predictive performance of these grammars can be improved by estimating the quality of a learned clause through an evaluation function that takes into account the length of the examples in the training data.

1.4 Justification

To decide between two models that equally well describe the data, the minimum description length principle (MDL) [10] suggests that $I(M|E)$ (which is the description length I of the model M, given the examples E) should be minimized. In our experiments we consider $I(M|E)$ to be the sum of the description lengths of the model $I(M)$ and the examples encoded through the model $I(E|M)$. An approximation to this principle is to estimate the compression that would be achieved by encoding the examples with the model [13,5]. To estimate this compression, the size of the examples that would be encoded by the model has to be known. Under the assumption that each example has the same length, which is usually the case, their individual length can be neglected and a simple count of covered examples can be used to estimate the compression achieved. However if we are learning BGs, the examples given are biological sequences, which often have variable length. In that case the length of each individual example has an undeniable impact on the total size of examples encoded through the model. It then becomes clear that the length of individual examples should not be neglected while estimating the compression achieved by a model. We want to apply this idea to an evaluation function that is to evaluate a grammar clause during learning. Such an evaluation function would use the size of the clause and the size of all examples that are covered by that clause to evaluate the performance of the clause. To the knowledge of the authors there has been no previous work on positive-only clause evaluation functions, used in ILP, that consider the length of training examples. Evaluation functions that estimate the compression achieved by a model are often called compression measures.

2 Experiment Design

In our experiments we run the ILP tool Aleph [12] on the NPP datasets described in Section 2.1. For each of those datasets we use several different clause evaluation functions (as described in Section 3) and observe the differences in performance. The results of the training and testing are recorded and subsequently summarized in Section 4.

2.1 Data Sets

Dataset 1 - Whole set
For the first set of experiments we used a part of the human Neuropeptide Precurser Protein (NPP) dataset which was used in [8]. The experiments conducted by [8] involved, among others, the inference of BGs on a set of NPP sub-sequences called *middle* (henceforth denoted as *NPP-middles*). In this work, we will only consider the NPP-middles because these are most interesting in relation to our hypothesis (Section 1.3). First, the NPP-middles are considerably longer than any of the other NPP parts which makes the induction of a grammar describing them a more challenging task, and second, they are also the only NPP subsequences that display high variations in length; our NPP-middles range from 5 to 95 amino acids. This dataset consists of 76 positive examples, 2908 random examples and some Background Knowledge (BK).

Dataset 2 - Training data grouped by length
To see if the evaluation functions perform differently for longer or shorter examples we decided to split up the dataset into several parts and conduct a second set of experiments. We took the dataset discussed in the previous paragraph and split it into three disjoint subsets, based on the length of positive examples, effectively creating three separate datasets. We denote these three subsets as follows: NPP-middles-short, NPP-middles-medium and NPP-middles-long. The intention was to split the set of positive examples into three sets containing more or less the same number of examples. The first subset, NPP-middles-short, contains 24 examples each of length (number of characters) $l < 13$. The reason why this set contains only 24 examples instead of 25 or 26, as seems logical with 76 total positive examples, is because we set a length threshold of 13 in order to prevent the examples of length 13 being split between two subsets. The second subset, NPP-middles-medium contains 26 examples with $13 \leq l \leq 29$. The third subset, NPP-long contains 26 examples with $l > 29$. The random examples were split up in the same way according to length of their examples, using the same cut-off values that were used for the positive examples. This results in NPP-middles-short containing 908 random examples, NPP-middles-medium containing 864 random examples and NPP-middles-long containing 1136 random examples.

Background Knowledge (BK)
The BK in this dataset consists of general molecular biology knowledge which can be considered relevant for any protein grammar inference process. The BK

contains amino acid letters and their physio-chemical properties (as first proposed by [4], and also used by [8]) and gaps. The purpose of gaps is to match parts of the protein sequence that are not directly relevant to the function or which cannot be characterized by the provided background predicates, but which still participate in the overall structure of the molecule [2]. This dataset, including the BK, was also used in [3] and [1]. (The datasets and BK can be found at: www.comp.rgu.ac.uk/staff/tm/materials/ILP08/) The set of random examples may contain protein sequences that would be positive, but at the time this dataset was collected, were still undiscovered as such. Consequently we cannot treat them as negative examples so we have to use a positive-only learning approach [7].

A 5-fold stratified cross-validation was performed on all datasets. The stratification of the cross validation was based on the length of the examples, ensuring that all training and test sets include a variety of examples of different lengths. The same Background Knowledge is used in each experiment.

2.2 Representation of Biological Grammars and Sequences

We are using the ILP tool Aleph to learn biological grammars. All input given to Aleph is using a Prolog related syntax. The same goes for the induced result, especially since we might want to use the induced grammar in further tests or experiments using Aleph or Prolog. The resulting BG, a context free grammar, is a set of rules that represent a given set of protein sequences. To represent such a grammar we use a Definite Clause Grammar (DCG) formalism [9] in the same way as in [8,1,3,2]. DCGs require sequences to be represented by a list, where each element in this list stands for a letter in the sequence. DCG rules take such a list as input and pass it on to the predicates that make up the rule. Each predicate, starting with the first, then matches one or more elements from the start of the list and returns the rest. This new, shorter list is then in turn given to the next predicate in the rule. If the last predicate returns an empty list, then the whole sequence is matched by the grammar rule and we consider the sequence to be covered by that rule. Aleph learns one rule at the time until all the examples are covered, and then it puts all the induced rules together to form the resulting grammar. (See Table 2 on page 183 for a summary of Aleph's search algorithm)

2.3 Suitable Performance Measure for an Induced Grammar

In Machine Learning and more specifically ILP, the most popular performance measure used to evaluate the final result of learning is the predictive accuracy:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP stands for true positives, FP for false positives, TN for true negatives and FN for false negatives However the accuracy of an induced grammar might not be a suitable performance measure when learning biological grammars. Our dataset (see Section 2.1) contains random examples instead of negative ones, so instead of TN and FN, we get TR and FR, referring to true and false randoms. These values can still be put in the above formula to compute the accuracy, but they don't quite mean the same thing. Our dataset also has a considerable unbalance in the ratio between positive and random examples: 76 positives compared to 2908 randoms. A consequence of this is that despite covering different numbers of the considerably rare positives, induced grammars have a high chance of being awarded a very high accuracy by excluding most of the abundant random examples [8]. To prevent cases where the accuracy could be inconclusive, we considered other, additional quality measures for the induced grammars.

From the domain of Information Retrieval (IR) we considered precision, recall and F-measure [14]. *Precision* in IR is the fraction of predicted positive examples that are indeed true positives and *Recall* is the fraction of true positives among all positives:

$$precision = \frac{TP}{TP+FP} \quad ; \quad recall = \frac{TP}{TP+FN}$$

In IR these two measures are often used in conjunction with the F-measure, which is the weighted harmonic mean of precision and recall:

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

These measures seem appropriate for our domain as well, so we decided to include them in this work (see Table 4 on page 184).

3 Clause Evaluation Functions

The focus of this work is the evaluation function which the ILP system uses to evaluate a clause. Such a clause evaluation function gives each accepted clause a score estimating its quality (Table 2 step 3). After the search is complete, the clause with the best score is added to the grammar (Table 2 step 4). This section describes the different clause evaluation functions used in this work. A summary can be found in Table 1.

Note here that a clause evaluation function and a grammar performance measure (introduced in Section 2.3) are two distinct things. While a clause evaluation function is used to grade each individual clause during learning, a grammar performance measure is only used after learning is completed successfully. The grammar performance measure aims to estimate the future performance of the entire learned grammar, which consists of a number of clauses.

3.1 Standard Positive-Only Evaluation Function

The standard positive-only learning evaluation score in ILP was devised by Muggelton [7]:

$$Score = log\,(P) - log(\frac{(R+1)}{(Rsize+2)}) - \frac{L}{P} \qquad (1)$$

Table 1. List of all evaluation functions used in this work

(1)	Score = $log(P) - log(\frac{R+1}{Rsize+2}) - \frac{L}{P}$
(2)	Score = $log(PosCoverage) - log(\frac{RanCoverage+1}{RLsize+2}) - \frac{L}{PosCoverage}$
(3)	Score = $log(PosCoverage) - log(\frac{RanCoverage+1}{Rsize+2}) - \frac{L}{PosCoverage}$
(4)	Score = $log(PosCoverage) - log(\frac{RanCoverage+1}{Rsize+2}) - L$

Where P is the number of positives covered; R is the number of randoms covered; $Rsize$ is the total number of randoms and L is the number of literals in the hypothesis. This evaluation function has been widely used by the community and therefore is our benchmark evaluation function. This means that our experiments aim to find a function that outperforms this one.

3.2 L-Modification of the Standard Positive-Only Evaluation Function

Function (1) does not consider the length of individual examples, so to put our hypothesis to the test we had to modify it. What we propose is to replace any variable in evaluation function (1) that would refer to numbers of examples (may that be covered, not covered or total) with a different variable that instead refers to a modified value which takes into account the length of examples. This means more precisely that instead of feeding P and R to to the evaluation function, we replaced all their occurrences in (1) with *PosCoverage* and *RanCoverage* respectively. PosCoverage is the sum of the lengths of all covered positive examples and RanCoverage is the sum of the lengths of all covered random examples. These changes combine the idea of existing pos-only evaluation functions with the idea of considering the length of examples. Henceforth we denote such a change of variables as *L-modification*. In addition to P and R, the variable *Rsize* in (1) also refers to a number of examples: the total number of random examples in the training data. As the name of the variable suggests, this number intends to represent the size of the set of randoms, so within the context of this work, it makes sense that we also apply our L-modification to this variable. This means that we replace *Rsize* with *RLsize*, which is the sum of the lengths of all random examples in the training data. Applying these modifications gives us:

$$Score = log(PosCoverage) - log(\frac{RanCoverage + 1}{RLsize + 2}) - \frac{L}{PosCoverage} \quad (2)$$

3.3 Further L-Modified Evaluation Functions Used in This Work

Although our experiments will focus on evaluation functions (1) and 2 we also ran our experiments on two other modified versions of evaluation function (1). The first of these only replaces P and R with *PosCoverage* and *RanCoverage* respectively with no further changes:

$$Score = log(PosCoverage) - log(\frac{RanCoverage + 1}{Rsize + 2}) - \frac{L}{PosCoverage} \quad (3)$$

The second one also takes into account that the value of PosCoverage is 10 to 60 times larger than P, therefore the last term of function 3, $(\frac{L}{PosCoverage})$, possibly leads to the clause length L greatly losing influence on the score. Therefore we tried giving L more weight by not dividing by $PosCoverage$:

$$Score \;=\; log(PosCoverage) - log(\frac{RanCoverage + 1}{Rsize + 2}) - L \qquad (4)$$

3.4 Implementing L-Modifications

Three of the four evaluation functions in this work use the L-modified coverage instead of the traditional coverage. Aleph, the ILP tool we are using does not have features that can access the length of individual examples in the training data. However it allows for user defined clause evaluation functions. Therefore, instead of having Aleph calculate the coverage of a clause by itself, we had it use customized predicates to compute the coverage during clause evaluation. These predicates parse examples through the clause and return the modified coverage. They still respect the search algorithm that Aleph applies (see Table 2), i.e. they only parse those examples that are still uncovered (not yet deemed redundant). In our experiments, when using the clause evaluation function (1) our predicates calculate the number of positive and random examples covered, just like Aleph's functions would, however for all subsequent experiments, using functions (3), (4) and (2) our predicates calculate the L-modified coverage, e.g. the sum of the lengths of all the covered positive or random examples. Some of the predicates that were used in this work to compute the L-modified coverage can be found in Appendix B on page 190.

Table 2. A simplification of the basic Aleph algorithm

1. Select a positive example to be generalised. If none exist, stop.
2. Build the bottom clause; the most specific clause entailing the example selected.
3. Search; Find a clause more general than the bottom clause.
   (Construct a search tree, each node containing a clause which consists of a subset of the literals in the bottom clause. Search for the clause with the best score)
4. Remove redundant. The clause with the best score is added to the grammar.
   All examples made redundant are removed. Return to Step 1.

4 Results

Table 3 shows the results obtained from running evaluation functions *(1)* and (2) on all datasets. From the results of the 5-fold cross validation we get the sum, average and standard deviation of TP (true positives), FP (false positives), TR (true randoms) and FR (false randoms).

Table 4 shows the evaluation of all the data collected in Table 3; the accuracy, precision, recall and F-measure of a theory. Definitions of these performance measures, can be found in Section 2.3.

Table 3. Summary of the results on all datasets (see Section 2.1) - The first leftmost column indicates which evaluation function was used (see Section 3 or Table 1), all the subsequent columns give the sum, average and standard deviation (std) of true positives (TP), false positives (FP), true randoms (TR) and false randoms (FR) observed during testing

Experiments conducted on NPP-middles												
evalfunc	TP			FP			TR			FR		
	sum	av.	std	sum	av.	std	sum	av.	std	sum	av.	std
(1)	50	10	2.12	652	130.4	72.44	2256	451.2	72.28	26	5.2	2.39
(2)	39	7.8	2.39	29	5.8	1.92	2879	575.8	1.79	37	7.4	2.61

Experiments conducted on NPP-middles-short												
(1)	20	4	1.00	62	12.4	9.13	846	169.2	8.93	4	0.8	1.10
(2)	15	3	0.71	5	1	1.00	903	180.6	1.34	9	1.8	1.10
Experiments conducted on NPP-middles-medium												
(1)	13	2.6	2.30	76	15.2	10.83	788	157.6	10.83	13	2.6	2.61
(2)	14	2.8	1.64	9	1.8	1.92	855	171	2.35	12	2.4	1.82
Experiments conducted on NPP-middles-long												
(1)	6	1.2	0.84	170	34	26.45	966	193.2	26.48	20	4	0.71
(2)	2	0.4	0.55	19	3.8	2.68	1117	223.4	2.88	24	4.8	0.8

Table 4. Evaluation of the results, derived from the values given in Table 3 - The first leftmost column indicates which dataset the values are referring to, the following column states which evaluation function (evalfunc) was used (see Section 3 or Table 1) and the next columns give the average of accuracy, precision, recall and F-measure observed during testing

Dataset	evalfunc	av. Accuracy	av. Precision	av. Recall	F-measure
NPP-middles	(1)	0.77	0.08	0.66	0.14
	(2)	0.98	0.57	0.52	0.54
NPP-middles-short	(1)	0.93	0.24	0.83	0.38
	(2)	0.98	0.75	0.62	0.68
NPP-middles-medium	(1)	0.90	0.15	0.50	0.23
	(2)	0.98	0.61	0.54	0.57
NPP-middles-long	(1)	0.84	0.03	0.23	0.06
	(2)	0.96	0.10	0.08	0.09

5 Discussion

5.1 Effects of the L-Modifications

Dataset NPP-middles

We use the experiment using clause evaluation function (1) as our benchmark and compare the results of the other experiments with this one. The main change that we can observe when looking at Table 3 is that the FP rate has been decreased drastically by the L-modified experiments. FP of 5.8 is an acceptable value, even within the positive-only learning NPP domain. It is mainly a consequence of this change in FP that the accuracy was increased from 0.77 to 0.98.

However, as we suggested in Section 2.3 this high accuracy could be misleading. The precision has been increased from 0.08 to 0.57 which is a considerable change. (1) has an extremely low precision as on average 130 random examples are accepted by the theory learned. This is over 25% of the total randoms provided in each fold.

The recall has been slightly decreased from 0.66 to 0.52. The reason for this is that the L-modified experiments produce theories with lower TP: 7.8 as opposed to 10 by our benchmark experiment.

Finally, the F-measure, the weighted harmonic mean of precision and recall, is increased in the L-modified experiments: from 0.14 to 0.54. We see this as a significant improvement.

Dataset NPP-middles-(short,medium,long)
Concerning all 3 subsets of this dataset, the same observations can be made as in the previous paragraph, using NPP-middles: the accuracy increases as a consequence of FP decreasing, the precision improves as well, the recall decreases, except using NPP-middles-medium where is increased by only 0.04 and the F-measure finally increases as well.

However a few additional observations can be made here. Looking at Figure 2 we can see that for each subset of the NPP-middles (short, medium and long) the performance of the L-modified evaluation functions is higher than that of our benchmark function (1). Also, generally, for each measure, the performance is better for shorter examples than for longer ones. The reason for this is that it is easier to learn rules covering shorter examples than longer ones, which makes sense.

Another observation that can be made when looking at Table 4 is that for datasets NPP-middle-short and NPP-middle-medium the accuracy and F-measure are higher than for dataset NPP-middles, even without L-modification. This is quite interesting as one would not expect this to be the case. Clearly, datasets NPP-middle-short and NPP-middle-medium, being subsets of NPP-middles, contain less examples than NPP-middles, so one would expect the performance to be lower. What sets the smaller datasets apart from the larger one is that the variation in the length of examples is different. This seems to indicate that the greater variations in the length of the examples contained in the NPP-middles dataset, compared to that in each of its subsets, make it harder for Aleph to generate hypotheses with similar performance.

5.2 Comparing Clause Evaluation Functions (3), (4) and (2)

In this work, we limited the reporting of results to evaluation functions (1), which served as our benchmark, and (2), which is the main contribution of this work. However as we stated in Section 3.3 we also ran all experiments reported using evaluation functions (3) and (4). The results of using these functions were not

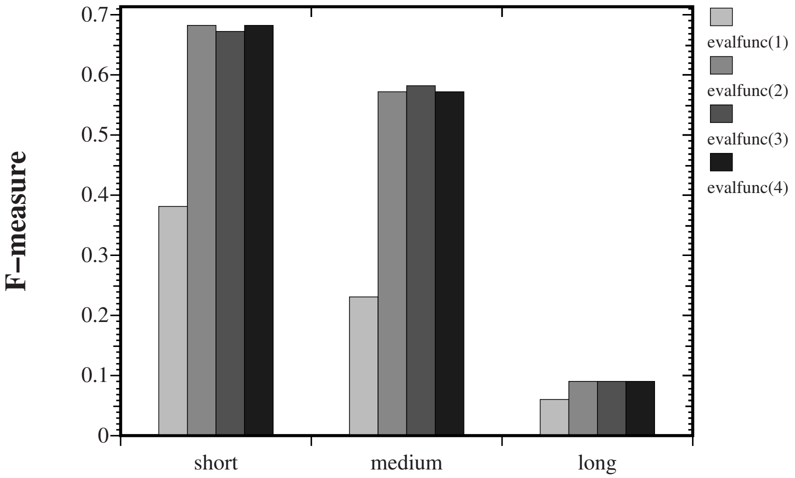


Fig. 2. For all three parts of dataset NPP-middles-(short, medium and long) the performance in terms of F-measure of each of the four evaluation functions is plotted

reported in Tables 3 and 4 because they were very similar, and in many cases identical to the outcomes of the experiments using function (2). Furthermore when we learn on the entire NPP-middle dataset, without any cross validation, then the grammars learned using these 3 evaluation functions are in fact identical and the search constructs the same number of nodes for these 3 experiments. There is a slight variation in time needed which can be accounted for by different computation times of the slightly different evaluation functions. We can conclude that although there is a significant difference in performance when introducing the length of examples into the clause evaluation function, there is no further advantage to be gained by considering the differences between (2), (3) and (4). The most likely explanation for this is that the first term of these L-modified equations, $log(PosCoverage)$, is the dominant term, especially in these experiments where we deal with large values of $PosCoverage$. As a consequence, the second and third terms in the L-modified equations have only a minor influence on the score.

5.3 Analysis of Induced Grammars

We have already noted that using our L-modified clause evaluation function increases the accuracy and F-measure of the learned grammars. However we made other observations that need mentioning. Table 5 contains an analysis of the grammars that were learned using the NPP-middle dataset, without any cross validation.

Applying L-modification to the evaluation function increases the number of rules in the learned grammars. This is the case because more rules cover only one positive example each (see Table 5 column 4). So even though the grammars are

Table 5. Details of the grammars induced using the NPP-middle dataset, without any cross validation - The first column indicates which evaluation function from Section 3 was used while inducing the grammar, the second column gives the average number of rules that make up the induced grammars when learned using 5-fold cross validation, the third column gives the number of rules that make up the induced grammar when learned on the whole dataset and the fourth column gives a count of how many rules were induced that only cover one positive example when learned on the whole dataset

evaluation	av. # of rules	# of rules in	# of rules
function	per fold	whole dataset	covering 1 ex.
(1)	15	14	3
(2)	30.6	35	22

better at describing the positive examples (cover a lot less random examples), they are more complex. This could be an indication that the grammars are more likely to be overfitted to the dataset. It is worth noting that those rules that cover only one example usually cover a very large example. Also, rules that cover only one example could provide useful information to the experts of the domain as they may indicate which part of a protein is of biological significance.

5.4 Total Time

In our experiments we also recorded the total time needed for the induction process. In general, experiments using functions (2), (4) and (3) took a lot longer than those using function (1). Even though computing our L-modified coverage requires a little more computational power, it is more likely that the large increase in running time is a consequence of more nodes being constructed by the search, which in turn results in the increase of performance of the resulting grammars.

In order to confirm this we ran an additional set of experiments using the NPP-middles dataset. This time we used a number of different parameters for `setNodes`, the Aleph setting that controls how many nodes are constructed during each search. We used the following values: 100000 (the value we used in all other experiments), 50000, 10000, 5000 and 1000. We then recorded the total number of nodes constructed for each learning task (the sum of all nodes constructed during each search). Figure 3 shows the graph plotting the performance of the evaluation functions against the total number of nodes constructed. We can observe that more nodes are constructed to learn grammars with better performance. In addition to that, the rate of increase is greater for the L-modified evaluation functions than for the benchmark function. This shows that indeed, for larger search spaces, the L-modified evaluation functions result in grammars with a higher F-measure. There are two factors that are responsible for the search space being larger when using L-modified evaluation functions:

1. If L-modified clause evaluation functions are used to calculate the score for a clause, the search is not as easily satisfied and more iterations of the search algorithm are called before a clause is finally added to the resulting grammar.

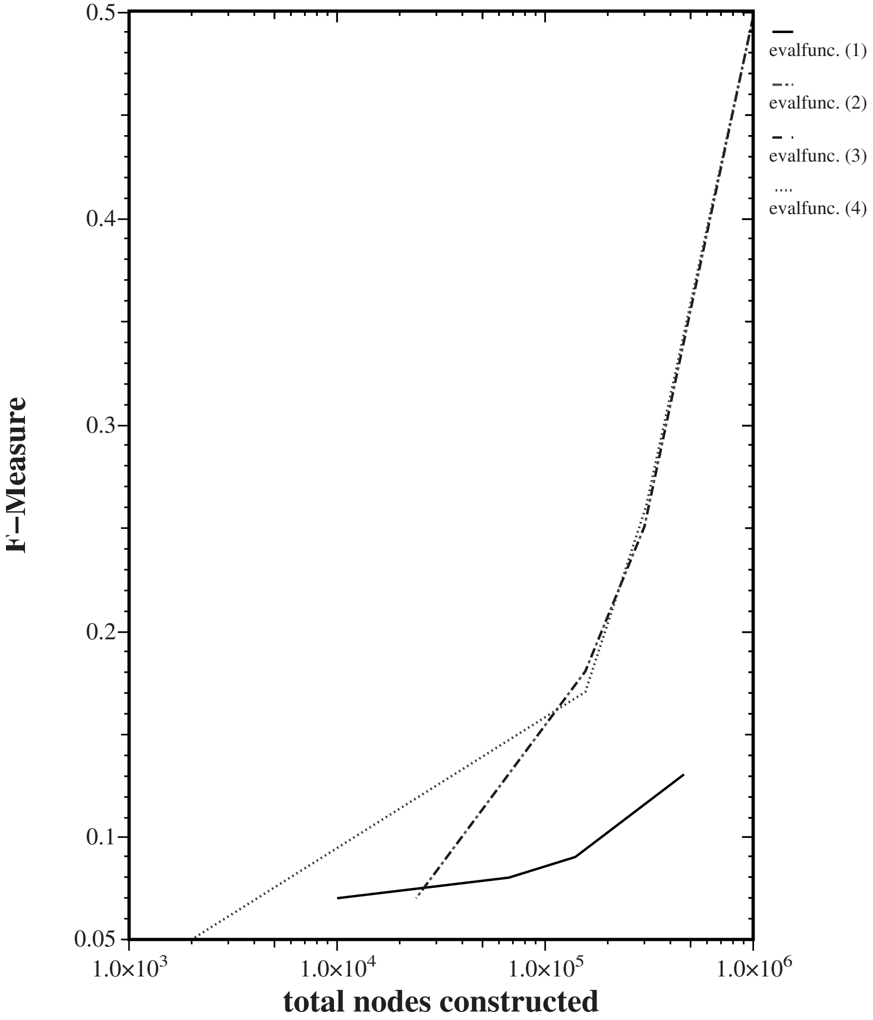


Fig. 3. For each evaluation function (evalfunc) the performance is plotted against the total number of nodes constructed at the end of induction

2. As we have shown in Section 5.3, grammars learned using the L-modified scores consist of more rules than their unmodified counterpart, which means that more examples are chosen to be generalized by step 1 of the search algorithm (see Table 2).

6 Conclusions

We have shown that when learning on the NPP-middle dataset, the L-modifications we propose do improve the performance of the induced grammars, both in terms of accuracy and F-measure. Splitting the NPP-middle dataset in 3 disjoint subsets and learning on those, we have shown that our L-modification improves performance of induced grammars for short, medium and long examples. Within this context, we have also shown that it is generally harder to learn rules covering longer examples than shorter ones. By observing the outcomes of learning on NPP-middle dataset and comparing them with those that we can observe when learning on each of its subsets (NPP-middle-short, -medium and -large) we conclude that it is harder for the ILP tool used in this work (Aleph) to learn from examples that have a high variation in their lengths. Finally, by running several experiments setting different limits for the amount of nodes allowed to be constructed during learning, we have shown that for larger search spaces, the L-modified evaluation functions result in grammars with a higher F-measure and that the rate of improvement is higher using L-modified functions.

Considering all the above, we can conclude that the L-modification proposed in this work does indeed improve the performance of evaluation function (1). Therefore we would expect the L-modification to improve other clause evaluation functions as well, so now there is a need to apply this approach to more than one standard clause evaluation function.

7 Future Work

Applying L-Modification to Different Evaluation Functions

To fully support our claims that the L-modification proposed in this work improves the grammars that are learned, there is a need to apply this approach to other standard clause evaluation functions. However, this is complicated by the fact that so far, we were dealing with positive-only learning. Evaluation functions meant for positive and negative learning are more common. We therefore propose to investigate how to apply this approach of L-modifying the coverage computation to clause evaluation functions which are tailored to positive and negative learning.

References

1. Bryant, C.H., Fredouille, D.: A parser for the efficient induction of biological grammars. In: Kramer, S., Pfahringer, B. (eds.) 15th International Conference on Inductive Logic Programming: late-breaking paper track, pp. 3–8. University of Bonn, Bonn (July 2005), http://wwwbib.informatik.tu-muenchen.de/infberichte/2005/TUM-I0510.idx
2. Bryant, C.H., Fredouille, D., Wilson, A., Jayawickreme, C.K., Jupe, S., Topp, S.: Pertinent background knowledge for learning protein grammars. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 54–65. Springer, Heidelberg (2006)
3. Fredouille, D., Bryant, C.H., Jayawickreme, C.K., Jupe, S., Topp, S.: An ILP refinement operator for biological grammar learning. In: Muggleton, S., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 214–228. Springer, Heidelberg (2007)

4. Muggleton, S., King, R.D., Sternberg, M.J.E.: Protein secondary structure prediction using logic-based machine learning. Protein Engineering Oxford 5(7), 647 (1992)
5. Muggleton, S., Srinivasan, A., Bain, M.: Compression, significance and accuracy. In: Sleeman, D., Edwards, P. (eds.) Proceedings of the Ninth International Machine Learning Conference, pp. 338–347. Morgan Kaufmann, San Francisco (1992)
6. Muggleton, S.H.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
7. Muggleton, S.H.: Learning from positive data. In: Muggleton, S.H. (ed.) ILP 1996. LNCS, vol. 1314, pp. 358–376. Springer, Heidelberg (1997)
8. Muggleton, S.H., Bryant, C.H., Srinivasan, A., Whittaker, A., Topp, S., Rawlings, C.: Are grammatical representations useful for learning from biological sequence data? - a case study. Journal of Computational Biology 8(5), 493–522 (2001)
9. Pereira, F., Warren, D.: Definite clause grammars for language analysis. Readings in natural language processing, pp. 101–124 (1986)
10. Rissanen, J.J.: Modeling by shortest data description. Automatica 14, 465–471 (1978)
11. Searls, D.B.: Linguistic approaches to biological sequences. Computer Applications in the Biosciences 13(4), 333–344 (1997)
12. Srinivasan, A.: A learning engine for proposing hypotheses (Aleph) (1993), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph
13. Srinivasan, A., Muggleton, S., Bain, M.: The justification of logical theories based on data compression. Machine Intelligence 13, 91–125 (1994)
14. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)

Appendix A: Materials

Most of the materials used in this work, dataset and input files for Aleph can be found online at www.comp.rgu.ac.uk/staff/tm/materials/ILP08/.

Appendix B: Code for L-Modified Coverage Computation

Predicate: Compute_Lmodcover/3

This predicate computes the L-modified coverage of a clause on all remaining examples of a certain type, where type stands for either positive or negative examples. In our experiments negative examples were substituted by random examples (see Section 2.1 page 179).

```
%used as: compute_Lmodcover(Type,Clause, L-modifiedCoverage)
compute_Lmodcover(Type,(Head:-Body), Cov) :- !,
     '$aleph_global'(atoms_left,atoms_left(Type,Left)),
     compute_Lmodcover(Type, (Head:-Body), Left, 0, Cov).
compute_Lmodcover(Type, ClauseWithoutBody, Cov) :-
     compute_Lmodcover(Type,(ClauseWithoutBody:-true),Cov).
compute_Lmodcover(_, _, [], Cov, Cov).
compute_Lmodcover(Type, Clause, [Inter|Rest], Cov, CovRes) :-
     compute_Lmodcover_interval(Type, Clause, Inter, Cov, Cov1),
     compute_Lmodcover(Type, Clause, Rest,Cov1,CovRes).

compute_Lmodcover_interval(_, _, Start-Finish, Cov, Cov) :-
     Start > Finish, !.
compute_Lmodcover_interval(Type,Clause,Start-Finish,Cov,CovRes) :-
     example(Start, Type, Atom),
%get the L-modified coverage SeqLength for this example:
     parse_exple_for_length(Clause, Atom, SeqLength),
%add L-modified coverage for this example to the total coverage:
     Cov1 is Cov+SeqLength,
     Start1 is Start+1,
     compute_Lmodcover_interval(Type, Clause, Start1-Finish,
                                          Cov1, CovRes).
```

Predicate: Parse_exple_for_length/3

This predicate parses an example and returns the length of the sequence if it was parsed sucessfully. If it cannot be parsed, 0 is returned.

```
%used as: parse_exple_for_length(Clause,Example,SequenceLength)
parse_exple_for_length((Head:-Body),Example,SeqLength) :-
    % the \+(\+()) are needed to ensure Head and Example
    % do not stay unified after the call
     \+((\+((Example=Head, call(Body))))),
     Example=middle(MidSeq,[]),
     length(MidSeq,SeqLength), !.
parse_exple_for_length(_,_,0).
```

Logical Hierarchical Hidden Markov Models for Modeling User Activities

Sriraam Natarajan[1], Hung H. Bui[2], Prasad Tadepalli[1], Kristian Kersting[3], and Weng-Keen Wong[1]

[1] School of EECS, Oregon State University, Corvallis USA*
[2] AI Center, SRI International, Menlo Park, CA, USA
[3] Fraunhofer IAIS, Schloss Birlinghoven, Germany**

Abstract. Hidden Markov Models (HMM) have been successfully used in applications such as speech recognition, activity recognition, bioinformatics etc. There have been previous attempts such as Hierarchical HMMs and Abstract HMMs to elegantly extend HMMs at multiple levels of temporal abstraction (for example to represent the user's activities). Similarly, there has been previous work such as Logical HMMs on extending HMMs to domains with relational structure. In this work we develop a representation that naturally combines the power of both relational and hierarchical models in the form of Logical Hierarchical Hidden Markov Models (LoHiHMMs). LoHiHMMs inherit the compactness of representation from Logical HMMs and the tractability of inference from Hierarchical HMMs. We outline two inference algorithms: one based on grounding the LoHiHMM to a propositional HMM and the other based on particle filtering adapted for this setting. We present the results of our experiments with the model in two simulated domains.

1 Introduction

Activity recognition is a problem that has long been the focus of researchers and has a wide range of applications from surveillance[8] to intelligent interfaces[11] to assisting the elderly[1]. Accordingly, several kinds of approaches ranging from domain-specific hand-coded solutions to the more general Hidden Markov Models(HMM) have been proposed for such activity recognition problems. Hidden Markov Models and their several extensions are among the most popular methods for activity recognition. The main advantage of HMMs and their extensions lies in the fact that they define a clear probabilistic semantics for the problem. Also, efficient inference algorithms have been proposed for HMMs making them very attractive for this problem. The different extensions of HMM like the Hierarchical HMM (HHMM) [5] and abstract HMMs (AHMM) [3] are being widely used in activity recognition for a variety of applications.

The main drawback of HMMs is that they do not take into account the complete structure of the problem. The objects in the domain may be related by

* Sriraam is currently at University of Wisconsin, Madison.

** Kristian was at CSAIL, MIT when the work was performed.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 192–209, 2008.

specific relationships and these relationships govern the action that the user performs in the current state. For example, a user is more likely to send a paper that he is writing to his co-authors and not to random email addresses. Also, since the HMMs are inherently propositional, they do not allow for generalization among the objects of the domain. For instance, the models cannot be shared among multiple users of the desktop, as a separate HMM needs to be constructed for every user in a propositional setting. Also in several cases, the user might decompose his goal of submitting a proposal or writing a paper by a similar methodology of running experiments, writing the paper, writing the abstract, sending it to one's co-authors etc.

The goal of this work is to extend the HMM with a logical model that allows for generalization of the objects in the domain and a hierarchical model that allows for richer structure of the user's tasks. The use of logical models will allow us to specify the models at an abstract level that can later be instantiated with specific instances and to perform inference on them. Also, it allows parameter sharing between the objects of the same type thus requiring a smaller number of examples for learning. The hierarchical structure will enable us to elegantly represent the user's goal-subgoal decomposition and allows for efficient inference. Kersting et al.[7] earlier introduced a logical extension to HMMs. In this paper, we extend their work in 2 ways: allowing conditional transitions and more importantly incorporating hierarchies in the logical models.

Our first contribution is to introduce the Logical Hierarchical HMMs (LoHiHMMs) and outline their syntax and semantics. In many real-world applications, the user chooses his action based on some conditions in the environment. For example, a user who goes shopping might prefer the nearest store to shop from. If the product that he wants to buy is not available in that store, he might go to another one. If it was available, he might buy the product and return home. The availability of the product can be observed by looking at the inventory. But, it is not possible to observe the mental state of the user. Hence, some parts of the state space are completely observed while some others are not. In our model, we consider the current state as having two components: an observable component (called the *world* state) and an unobservable component (called the *user state*). Note that the names of world and user are specific to our applications but they mean the *observable* and the *unobservable* parts of the state space. One of our critical assumptions is that the world state is completely observable. This would make it possible to perform tractable inference (as we show later in the paper) in large domains. In our model, we naturally model such conditional transitions that take place based on the observable part of the state space.

Logical models have generally been unrolled to their ground formulations and inference was performed using these ground models. We present an algorithm for unrolling the LoHiHMM to the corresponding ground HMM which will make it possible to perform exact inference in the model. But, in very large domains, the state space of the ground HMM could be prohibitively large and can make the inference hard. HMM inference is quadratic in the number of states and a very large state space can make the inference impractical if not infeasible. Hence, in this work, we avoid complete unrolling of the LoHiHMM. Our second major contribution is to adapt particle filtering to this logical setting. The filter avoids considering all the possible states by exploiting the conditions in the transitions and restricting the set of possible next states. We compare the performances of the exact inference on the ground HMM and the particle filter on 2 simulated domains: a grid world domain where the user has to navigate the grid to achieve his tasks and a kitchen domain where the user follows some recipes to cook his dishes. The results demonstrate that the particle filter performs comparably to that of the exact HMM with much less overhead.

The rest of the paper is organized as follows: the next section provides the background for the work. Section 3 first presents the LoHMM, its syntax and semantics and then extends them to the Logical Hierarchical HMMs. Section 4 outlines the inference algorithm based on particle filtering and the algorithm for unrolling it to a ground HMM. Section 5 presents the experimental results on the domains. The final section concludes the paper by reviewing the related work and outlining some areas of future research.

2 Background

In this section, we briefly review the background work namely, HMMs, Hierarchical HMMs, Particle Filtering and the previous work on Logical HMMs.

Hidden Markov models (HMMs) are used to model systems that follow a Markov process which is essentially a process with no memory. In a Markov process, the next state of the system is independent of the past states given its current state. In HMMs, there are 2 kinds of variables: a *state* variable that follows a Markov chain and an *observation* variable whose value is generated by the current state. As the name implies, in an HMM the states are assumed to be unobserved(hidden). Hence, one of the important tasks is to infer the distribution of the current state conditioned on the values of the observations (known as filtering). Formally, an HMM is defined using the 5-tuple $\langle S, \mathit{\Pi}, O, \mathit{\Omega}, I \rangle$, where S is a set of states, $\mathit{\Pi}(s'|s)$ is the next-state distribution, O is a set of observations, $\mathit{\Omega}(o|s)$ is the probability of observing o when in state s and I is the initial state distribution. The HMM starts in one of the states s chosen according to $I(s)$, at every step transitions to next states according to $\mathit{\Pi}$, and emits observation o with probability $\mathit{\Omega}(o|s)$.

It is clear that while considering the problem of activity recognition, the goal is to infer the distribution over the states given the current set of observations $p(s|o_{1:t})$. Hence the activity recognition problem is posed as performing inference in an HMM. Although a HMM is a very good choice for the problem, there are applications where the number of states can be arbitrarily large. It is also difficult to model situations where some states take more than one time-step to be executed. For instance, editing a particular document might take several time-steps and it is not possible to fix the number of time-steps in advance. In addition, the user's tasks could have a well-defined hierarchical structure that can be exploited if captured explicitly by the model. Though these can be captured by a HMM, the structure in the problem cannot be exploited efficiently.

Hierarchical HMMs[5] extend HMMs to include a hierarchical structure for the states. Each state in a HHMM is a self-contained probabilistic model. One way to understand a Hierarchical HMM is to think of each state of the HHMM being an HHMM in itself. So when the HHMM transitions to the current state, the sub-HHMM is activated which in turn activates a HHMM at the lower level. This would mean that at each state, the HHMM will emit a sequence of observation symbols rather than a single symbol. The key difference to a normal HMM is the notion of *end* states. Each sub-HMM has a set of end states that can terminate the HMM at the current level and return the control to the calling state of the parent HMM.

Any HHMM can be converted to a HMM[10]. The state of the HMMs correspond to the state stack of the HHMMs($S^{1:D}$), where D is the number of levels of the HHMM. If the HHMM structure is a tree, it would imply that there are no repetitive substructures. The corresponding HMM then contains one state for every combination of the parent and child states. If the HHMM has repeated substructures, they have to be duplicated in the HMM resulting in a large model. We refer to [2],[10] and [5] for more details on converting a HHMM to a HMM.

Since in large HHMMs (or even large HMMs), performing exact inference might be impractical, sampling methods became popular. The key innovation in these methods is to focus the set of samples on high probability regions of the state space i.e., generate more samples in high posterior regions. Among the sampling methods, *particle filtering*[4] methods are very popular. The general approach to particle filtering works as follows: Samples are generated according to the prior distribution and propagated to the next step according to the transition distribution. The samples are then re-weighed according to the observed evidence probabilities and new samples are generated. We present the generating distributions later in the paper when we discuss the particle filter for the logical setting.

Logical HMMs: Given that the HMMs are inherently propositional, Kersting et al. introduced Logical HMMs [7] to combine ideas from SRL[6] and dynamic models. In their framework, the states of the HMM are abstract states rather than propositional states. An abstract state consists of a predicate name and a set of parameters that can be instantiated with constant ground values. The transition distribution in the logical HMM now consists of two kinds of distributions: an abstract transition distribution that specifies the probability of the next abstract state given the current abstract state and a selection distribution that specifies the probability of the instantiation of the parameters for the current state. In their work, the observations are generated from state transitions and the selection distribution is specified using a Naive Bayes function.

3 Logical Hierarchical Hidden Markov Models

In this section, we present our formalism of LoHiHMMs. Since this formalism extends the HMM in two dimensions, we present the logical extension (LoHMMs) as a first-step and later extend it to include hierarchies. We refer to our logical extension as LoHMMs and that of Kersting et al. as Logical HMMs. We contrast our formalism to theirs after we introduce our model.

3.1 LoHMMs

Consider the HMM presented in Figure 1. In this example, a user receives a document to edit through an email, edits the document and sends it back to the sender. The semantics is that when the conditions in the arcs are satisfied the HMM transitions to the next state with the probability indicated by the number on the arcs. In this section, we first define the states and transitions of this HMM and then formally define LoHMM. In our model, the state in the model consists of two components, the state of the user and the state of the environment. The state of the user is assumed to be a single ground predicate that indicates the activity of the user. The user is restricted to be in only one state at any given instant of time. The state of the environment could be a conjunction of several predicates. An example of user state is $edit(D)$, where *edit* is the logical predicate and D is the parameter (variable) associated with this state. The environment state could include information about the current time, the deadlines, the set of projects that the user is involved with or as in Figure 1 whether a request for edit has been received etc.

State: A state consists of two components: the *user state* which is a predicate that represents the current activity of the user and the *environment state* which is a set of predicates describing the environment. We aim to capture transitions between user states conditioned on some states of the environment. This will enable us to model the user's activity conditioned on the context of the execution. For example, if the document that the user is currently editing is a long document, the user might prefer to print the document, while if it is a short one, the user might just read it off the monitor. Observing the length of the document will enable us to predict the user's next action. Though there are several ways

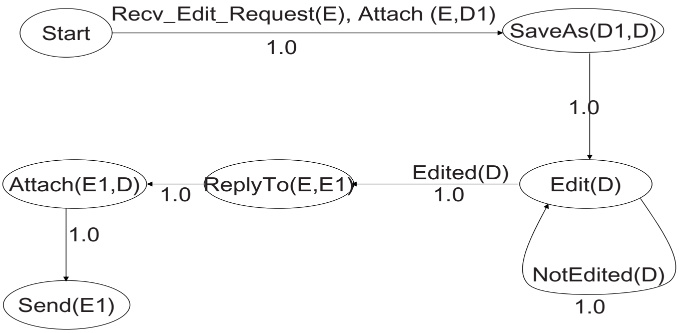


Fig. 1. A Logical Hidden Markov Model for editing a document. The logical conditions on the arcs indicate the conditions under which the transitions take place. The numbers on the arcs indicate the probability of the transition.

Table 1. Structure of an Logical transition in our model. The current user state is $A(X,Y)$ and transitions to one of the successor states based on the condition and logical transition probability.

$$A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(Y,Z)\text{... then} & \begin{cases} p_1^1: & B_1^1(X,Z) \\ p_2^1: & B_2^1(Y,Z) \end{cases} \\ \text{else if } C_1^2(X,Y) \wedge C_2^2(X,Y,Z)\text{... then} & \begin{cases} p_1^2: & B_1^2(X,Z) \\ p_2^2: & B_2^2(Y,Z) \end{cases} \\ \textit{else} & ... \end{cases}$$

of modeling the conditions in the transitions between the states, we use a model that is similar to decision lists or the case statements in programming languages.

Associated with each state predicate we can define a set of observation predicates and an associated observation distribution. The observations in our model are similar to the state predicates. The observation consists of a predicate name and a set of variables. The observation may or may not consist of the same variables that are present in the state predicate.

The transitions in our model are called *logical transitions* and are of the form presented in Table 1. The transition occurs between the current state of the user (called as *source* user state) and the next state of the user (called as *target* user state) conditioned on some of the predicates of the environment. As shown in the figure, there could be several logical test predicates that can be conjunctions of several predicates (shown as C in the figure) based on which transition could take place.

The most important constraint is that exactly one branch of the transition will be always taken for the given state. Once a particular branch, say k, evaluates to true and the corresponding branch is taken, the target atom is chosen according to the transition distribution (called ‘’logical transition distribution”) $\boldsymbol{p^k}$ for the current branch k. $\boldsymbol{p^k}$ is a well-defined distribution i.e, $\sum_{i \in S} p_i^k = 1$, where S is the set of target abstract states for the given logical transition. In the statement presented in Table 1 for instance, $p_1^1 + p_2^1 = 1$. An example transition is presented in Table 2. In this example, the user downloads a document P and if it is a long paper, the user prints the document with a probability 0.9 or reads it off the webpage with a probability 0.1. If not, he reads from the webpage.

The branches can be understood as similar to decision lists in programming languages. Given the instantiations of the current user state, the branch that first evaluates to true is taken. Once a branch is chosen, the logical transition distribution is used to choose the next user state predicate. The last branch has the key word *else* and has no conditions. It is a default one that will be taken if the other branch conditions fail to evaluate to true (for example see Table 2). This is to ensure that the conditions in at least one branch would evaluate to true and hence one branch will always be taken for the given user state.

Logical Transition: A logical transition consists of the current user state S as the source and a set of logical predicates of the environment. Corresponding to

Table 2. Example of a Logical transition. The user prints the document if it is a long document or reads it off the webpage.

$$Download(P) \mapsto \begin{cases} \text{if } LongPaper(P) \wedge URL(Z,P) \text{ then} & \begin{cases} 0.9: & Print(P) \\ 0.1: & Read(P,Z) \end{cases} \\ \text{else if } URL(Z,P) & \begin{cases} 1.0: & Read(P,Z) \end{cases} \\ \text{else} & \begin{cases} 1.0: & DoNothing \end{cases} \end{cases}$$

each predicate is a set of target user states and a corresponding set of probability distributions called the logical transition distributions. For each of the target user states, there is an associated instantiation distribution ($\boldsymbol{\mu}$) that specifies the probability of the values of the variables associated with the target state. We discuss the instantiation distribution later.

One potential issue is that there could exist several transitions that can be matched with the current instance. For example, ignoring the conditions there could be 2 different abstract transitions defined as follows:

$$A(X,Y) \longrightarrow B(X,Z)$$
$$A(x,Y) \longrightarrow C(Y,Z)$$

The second transition is more specific when compared to the first transition. If we use a substitution $\{\theta = X/x, Y/y\}$, we could match the two rules and there is a question of which transition to choose. In order to avoid multiple matching rules, we require that these 2 rules are ordered and combined to a single decision list as follows

$$A(X,Y) \begin{cases} \mathrm{I}f \quad (X = x) \quad 1.0 : C(Y,Z) \\ \mathrm{e}lse \quad 1.0 : B(X,Z) \end{cases}$$

The idea is to encode the more specific instances as well in the transition. This is natural in our formalism as we always assume that only one branch can be true for any ground instantiation. This means that no two different logical transitions can exist with the same predicate name as its source.

Now, we define a generalized LoHMM to consist of set of states S, a set of observations O and a set of logical transitions Δ. This definition of the LoHMM is too general for many problems. If we assume that the logical predicates are unobserved along with the user states, the problem of user activity recognition includes performing inference over the environment states as well. Hence, we consider a restricted model in which we assume that the environment states are completely observed and the user state is assumed to be unobserved. This assumption is not unreasonable in many domains such as a desktop domain, video surveillance etc where the effects of the user's actions in the environment are completely observable while the user's mental states are not.

Definition 1 (LoHMM). *A LoHMM consists of*

- *a set of states S which has two components: the user state S_u that is a single predicate and is unobserved and the environment state S_w that can possibly consist of many predicates and is completely observed*[1]
- *a set of observations O and the associated observation distributions*
- *a set of logical transitions Δ.*

Note that the assumption of the state being composed of the observable environment states and the unobserved user state predicate makes the inference process easier. This will enable us to ignore the changes to the environment predicates and just focus on the changes to the user state. The key thing to note here is that the logical conditions are the observable part of the state space. Hence given a current user state, it would be easy to evaluate the different logical predicates conditioned on the current user state. The LoHMM can be understood as defining a distribution over the set of ground predicates (interpretations). The ground interpretation consists of two parts: the user activity interpretation that is not observed and the environment interpretations that are observed and are fixed and do not change with time. The transitions can be understood as a stochastic mapping between two ground interpretations that depends on certain parts of the state.

Global Variables: There could be some variables in the LoHMM whose value should not change as the HMM evolves. For instance, consider the example in Figure 1. The user, after editing the document, has to reply to the email in which he received the request to edit. This means that the value of the variable E once set to the value of the email that has the request should not be changed. In our model, some of the variables are declared as global and their values do not change over time.

Substitution: Substitution in LoHMMs is defined exactly as in first-order logic. A substitution θ is defined as assignment of values to all the variables in the LoHMM. The set of variables is the union of all the variables in all the predicates of all the user states and the environment states.

Ground HMM: A ground HMM is a HMM constructed by substituting the values for all the variables of a LoHMM. We refer to the user state that has been substituted for its variables as an instantiated user state. The state-space of this ground HMM consists of two components: the instantiated user states and the global list. The transition distribution specifies a distribution over the next ground user states and the values of the global variables.

Theorem 1 (Existence of a ground HMM). *A ground HMM can be constructed for every LoHMM.*

[1] The names user state and environment state are chosen for the activity recognition purposes only. They mean the unobservable and observable part of the state space in general.

The proof is trivial and follows from the theorem in Kersting et al [7] and we omit the proof for brevity. The motivation behind stating the existence of a HMM is that, since a HMM defines a unique distribution over the trajectories, it follows that the LoHMM also defines a unique distribution over the trajectories. This ensures that there is a well-defined model-theoretic semantics for the LoHMM. In fact, HMMs are a special case of the LoHMMs with the states having an arity of zero, no conditional branches and no instantiation probability. Hence LoHMMs generalize HMMs.

We now present the relationship to the Logical HMMs of Kersting et.al [7]. In their model, there was an abstract transition distribution $\boldsymbol{p}$, which chooses the target predicate to transition to. In our model, first the logical conditions are evaluated, the corresponding branch is taken and then the target predicate is chosen according to the distribution $\boldsymbol{p^k}$. Their model does not allow for conditional branches. Thus their model can be considered as a special case of our LoHMMs where the default ($else$) branch is always taken. Recall that since they restrict their states to be predicates as well, the conditions cannot be captured in their Logical HMMs. Yet another difference is the presence of the global variables in our model. In their model, a variable has to be present in all the states for it to retain its value. Declaring a variable as global is useful in particular for the hierarchical setting as we explain in the next section.

3.2 LoHiHMMs

In many real-world situations, users solve difficult problems by decomposing them into a set of smaller ones. For example, proposal writing might involve writing the project description, preparing the budget, and then getting signatures from proper authorities. The tasks to be completed by the user have a natural hierarchical structure. To capture this kind of knowledge using LoHMMs, the state predicate should include the user's current task at all the levels of the hierarchy. This is not very elegant and it would be difficult for the domain expert to specify this kind of LoHMM; also, the inference process can be very difficult due to the explosive state space. A representation that could capture such a hierarchical structure cleanly can be used to perform efficient inference. In this section we introduce the Logical Hierarchical HMMs.

Recall that in a LoHMM, there are logical (horizontal) transitions between user states while in the Logical Hierarchical HMM (LoHiHMM) in addition to the horizontal transitions, there are vertical (task-subtask) transitions between parent and child tasks[2]. It should be mentioned that LoHiHMMs impose the hierarchical structure on the user's states and not on the environment state predicates as we are interested in modeling the user's activities and not how the environment evolves.

Instantiation Probability and Global Variables: The distribution that is used for grounding the state predicates to their ground values is called the *Instantiation Distribution* and denoted by μ. The constraints in the abstract transition

[2] We refer to the states of the LoHiHMM as tasks and subtasks.

Table 3. The syntax of the selection transition statements. The keywords *parent* and *child* denote the parent and child task respectively. The other parts of the statements are similar to LoHMM.

$$parent:\ A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(X,Y,Z)... \text{ then} & \begin{cases} sp_1^1 : child : & B(X,Z) \\ sp_2^1 : child : & C(Y,Z) \end{cases} \\ ... & \\ else & ... \end{cases}$$

serve as hard constraints for this distribution. In addition, in the hierarchical setting, the variable instantiation of the parent state should stay the same in all transitions made by its child states, i.e., the variables of the parent state should be"global". Their values cannot change once instantiated until that state finishes its execution. The global list thus includes variables that are declared global along with the values of the parent variables.

Selection Probability: The selection probability specifies the probability distribution over the child tasks given the parent tasks. We call the transition between a parent task and a child task a *selection transition* and the syntax is presented in Table 3.

The syntax is similar to the LoHMM syntax except that we now use special keywords namely, *parent* and *child* to identify the respective tasks. The rule that is presented in Table 3 is interpreted as: *"If the current user task is A(X,Y), then it chooses one of the possible next child tasks by evaluating the branch predicates (shown as C_i^j in the rule) and then chooses the child user task (example B or C) based on the selection distribution"*. This rule is similar to the transition rule in LoHMMs except that in this case, it defines a task-subtask transition of a HMM. The branch predicates are handled in a manner similar to a decision list such that the first satisfied branch is always taken. The selection probability is well defined, i.e., the sum of the probabilities of the different child abstract tasks for a given branch is 1 ($\sum_i sp_i^j = 1$). Note the presence of a default branch that is chosen if all the other branch conditions fail to evaluate to true.

Transition Probability: The transitions are similar to the LoHMMs with one difference: the transitions must include the task of the parent. Hence, the horizontal transition is conditioned on the instantiated value of the parent tasks. The syntax of the horizontal transition is presented in Table 4. The statement is similar to the ones presented for the LoHMMs except that these statements include the parent tasks (shown using the keyword *pa*). These statements can be interpreted as follows: *"If the current parent task is A(X,Y) and the current child task is B(X,Z), then it chooses one of the possible logical transitions by evaluating the branch predicates (shown as C_i^j in the rule) and chooses the abstract child task based on the transition distribution"*. For instance, the next child could be of the form $C(x,y,z,w)$ with probability $p_1^1.\mu(w)$, where p_1^1 is the abstract transition probability for $C(X,Y,Z,W)$ and $\mu(w)$ is the instantiation

Table 4. The syntax of the horizontal transition statements. The keywords *pa*, *ch* and *nch* denote the parent the current child and the next child abstract tasks respectively. The other parts of the statements are similar to LoHMM.

$$pa:\ A(X,Y);\ ch:\ B(X,Z) \mapsto \begin{cases} \text{if } C_1^1(X,Y,Z) \wedge C_2^1(X,Y,Z)... \text{ then} & \begin{cases} p_1^1 : nch : C(X,Y,Z,W) \\ p_2^1 : nch : D(X,Y,Z,W) \end{cases} \\ ... & \\ else & ... \end{cases}$$

probability for w. Note that the variables in the parent (X, Y) will be "global", since its binding will stay the same until the current parent terminates.

As with the selection transition of LoHiHMM, the set of branch predicates for a particular parent-child combination transition rule is part of a decision-list so that the first branch that evaluates to true is taken. The conditions on the transitions could also serve as hard constraints on the instantiation distributions. Similar to the LoHMMs, the transition distribution is well defined i.e., $\sum_i p_i^j = 1$.

Ending Probability: If the LoHiHMM at a lower level terminates at some task, the control would return to the parent task. For example, it is possible that the user might take a break from running the experiments and instead work on the abstract. It is possible that the current task might terminate in more than one task. Hence there is a necessity to define a distribution over the possible end tasks at a particular level. We specify the ending probability as a function β from the set of ground tasks (at any given level) to $[0, 1]$.

Execution semantics: The execution semantics of the LoHiHMM is as follows: When control reaches the current user state, it chooses a child state to transition to based on the selection distribution and the instantiation distribution. Once all the lower level HHMMs terminate, the control reaches back to the current state and it chooses a horizontal transition as with the case of LoHMMs. The main intuition is that at each level we have a LoHiHMM and the selection distribution chooses the LoHiHMM to execute. Each vertical transition (selection of a child) also has logical conditions that need to be evaluated to choose the child. LoHiHMMs are to LoHMMs what HHMMs are to HMMs. They can be viewed as either incorporating hierarchies to LoHMMs or adding logical models to Hierarchical HMMs. Similar to the LoHMMs, it is possible to state and prove the existence of a ground Hierarchical HMM for every LoHiHMM.

It must be mentioned that since a HHMM exists for every LoHiHMM and since a HMM exists for every HHMM, there exists a HMM for every LoHiHMM but with a prohibitively large state space. LoHiHMM provides 2 further significant advantages: first, it is easier for the domain expert to elucidate the knowledge as it is difficult to encode all the relationships and the task hierarchy into the CPTs of the HMM. Second, learning can be easier since the parameters are shared in the LoHiHMM.

4 Inference

In this section we present two methods for inference in these models. The first is to unroll the LoHiHMM to a ground HMM and use a sparse matrix package. Our second method avoids explicit unrolling and adapts particle filtering for the logical setting. Although we present the algorithm for unrolling to a ground HMM, we do not do that automatically in our experiments and instead use an equivalent hand-crafted HMM for comparison.

4.1 Particle Filter with Logical Querying

Particle Filter[4] samples the next state based on the evidence and computes the weight of the samples. The state at time t is denoted by x_t and the observation at time step t is denoted by y_t. Samples are drawn according to the optimal proposal distribution $P(x_t|x^i_{t-1}, y_t)$ given that i is the current level of the goal stack where,

$$P(x_t|x^i_{t-1}, y_t) = \frac{P(y_t|x_t)P(x_t|x_{t-1})}{\sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})} \tag{1}$$

and the weight w_t is given by,

$$w_t \propto P(y_t|x^i_{t-1}) = \sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})$$

The states are then re-sampled based on the weights of the samples. In this section, we modify the algorithm for the LoHiHMMs. The main bottleneck is computing the summation over all the next states (x_t)[3] in the equation 1. In a logical model, the number of possible ground states can be very large. Thus it might not even be possible to enumerate all the states for the summation to be computed. For instance, in a desktop there could be a very large number of files and emails. It is impractical to enumerate all possible combinations of files that can be attached to an email. Hence we propose to use a method that avoids enumeration of the complete state space. In our model, given the value of the previous sample and the values of the elements in the global list, a query answering procedure would return the set of possible next states. The intuition is that the conditions in the abstract transitions would greatly reduce the set of possible next states.

Given a particular ground state and the values of the global list, the selection procedure would first determine the set of logical transitions that have the current user state predicates as the source and obtain the set of the possible next user state predicates (task-subtask combinations) based on the logical conditions. For each of the next predicates, possible instantiations are considered using the instantiation distribution. The probability of the next ground state is the product of the selection probability and the instantiation probability. In some cases, if we are only interested in the current user state predicate (for instance,

[3] In the case of LoHiHMMs, x_t corresponds to the user's goal stack.

whether the user is composing an email or editing a document), we ignore the instantiation distribution and sample only from the selection distribution.

For instance, consider the example presented in Figure 1. Let us assume that the user just completed editing the document say $d1$. So when the particle filter queries for possible next states, the condition $edited(d1)$ is satisfied. Since the email E is in the global list, the selection procedure returns the next state as replying to email E. This greatly reduces the computation that is otherwise needed to sum over all the states.

Our inference procedure can be understood as a lazy evaluation procedure where the HMM is constructed on the fly based on the values that have been assigned to the variables and the conditions that are satisfied. The main goal of this procedure is to avoid the explicit construction of a ground HMM. This is because as we have pointed out earlier, the number of objects in the real-world is very large which in turn leads to huge transition and observation matrices. But considering the objects lazily based on the current state and the values of the variables, we are able to perform effective inference in real time.

4.2 Constructing a Ground HHMM

It is clear that the LoHiHMMs extend the LoHMMs similar to the way in which Hierarchical HMMs (HHMMs) extend HMMs. Hence it is conceivable that a LoHiHMM can be unrolled into a ground HHMM. The basic idea of grounding to a HHMM is as follows: once the current abstract state at level d has been instantiated with the corresponding values subject to the constraints, it chooses a child abstract state at level $d + 1$ based on the logical conditions and the selection distribution. The child state then chooses its instantiation based on the instantiation distribution and the global list of variables. It then chooses its child state at depth $d+2$ recursively. If the current state at d is a primitive state, it chooses the next state at the subsequent time-step based on the conditions and transition distribution. If the child state terminates, the control returns to the parent state, which makes an abstract transition to the next state at the same level based on the conditions and the transition distribution.

The key thing to note is that the process is very similar to the execution of a HHMM. Since at any time a particular condition can only be true for any particular transition, we obtain a HHMM at the ground level. We now proceed to define this ground HHMM and the process of unrolling. The states of the ground HHMM at level d correspond to the possible ground instantiations of the atoms of all the possible predicates at level d. Let us consider the selection of a ground child state given the parent state. Assume that the current abstract state is $A(X, Y)$ and the selection transition is defined by the rule in Table 3. Then, for all substitutions of X,Y,Z, i.e, $\forall\theta = X/x, Y/y, Z/z$

$$If\ BP_1(\theta) = True, \begin{cases} \mathrm{P}(B\theta \mid A\theta) = sp_1^1 \cdot \mu(\theta) \\ \mathrm{P}(C\theta \mid A\theta) = sp_2^1 \cdot \mu(\theta) \end{cases}$$

$$else...$$

Hence the idea is to consider all the substitutions of the current parent states and evaluate the branch conditions and choose the next child ground state based on the selection and instantiation distributions. The above step takes place if the state $A(X, Y)$ is instantiated at the previous time-step and it has to choose the ground child state in the current time-step.

If the current state is $B(X, Y)$ and the child state finishes executing, it will make an abstract transition to the next state at the same level. For example, consider the transition in Table 4 and assume that the current state at level d is $B(x, z)$ and then child tasks at level $d' > d$ are completed. $B(x, z)$ then makes an abstract transition. For all substitutions of X,Y,Z,W $i.e, \forall\theta = X/x, Y/y, Z/zW/w$

$$If\ BP_1(\theta) = True, \begin{cases} \mathrm{P}(C\theta \mid B\theta, A\theta) = p_1^1 \cdot \mu(\theta) \\ \mathrm{P}(D\theta \mid B\theta, A\theta) = p_2^1 \cdot \mu(\theta) \end{cases}$$

$$else...$$

Here the current state considers all the substitutions, evaluates the branch condition and chooses the next state based on the abstract transition and instantiation distributions. Since it has been shown that a HMM exists for every HHMM [5], we can use any standard HMM inference with a sparse matrix representation for performing inference.

5 Experiments

In this section, we present our experiments with the particle filter and unrolled (hand-crafted) HMM in 2 simulated domains: a grid world and a cooking domain.

5.1 Doorman Domain

In this domain, the user is in a gridworld where each grid cell has 4 doors that the user has to open to navigate to the adjacent cell. The LoHiHMM for the figure is presented in Figure 2.a. The highest level goals of the user are to *Gather* a resource or to *Attack* an enemy. To *gather* a resource, the user has to *collect* the resource and *deposit* it at the corresponding location. Similarly, to *destroy* an enemy, the user has to kill the *dragon* and *destroy* the castle. There are different kinds of resources, namely *food* and *gold*. Each resource can be stored only in a storage of its own type (i.e, *food* is stored in *granary* and *gold* is stored in *bank*). These serve as constraints in the LoHiHMM and are shown as conditions on the arcs in the Figure 2.a. There are 2 locations for each of the resources and its storage. Similarly there are 2 kinds of enemy *red* and *blue*. The user has to kill the *dragon* of a particular kind and *destroy* the castle of the same kind. The actions that the user can perform are to move in 4 directions, open the 4 doors, pick up, put down and attack.

The states of the LoHiHMM are the goal-subgoal combinations of the user. The world state (observable) consists of the current square that the user is

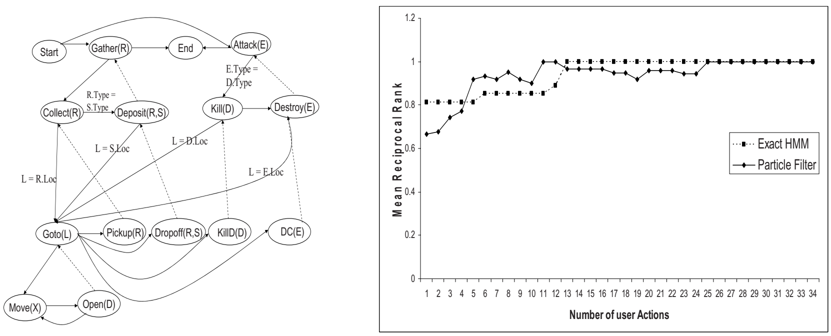


Fig. 2. (a)LoHiHMM for the gridworld domain. (b)Results. The y-axis presents the Mean Reciprocal Ratio of the correct user state.

in and the current door that is open. The observations are the user actions. The observation distribution is generated using a Boltzmann distribution on the number of actions to the goal. The results are presented in Figure 2.b. The particle filtering method is compared against the ground HMM on 50 different runs where the user chooses the top-level goal and starts acting towards achieving it. The mean reciprocal rank (mean of the inverse of the rank of the current state) was measured and presented. A MRR of 1 indicates that the correct state has always been ranked as 1 by the algorithm.

As can be seen, the grounded HMM (being an exact inference) method has a slightly better performance than the particle filter. More precisely, when a subgoal has been achieved (or at the start of the run), particle filter can sample a few of the incorrect subgoals. Once a few observations are obtained, the particle filter performs as well as the ground HMM (in fact even better before the first sub-goal has been achieved). On seeing a few observations, it converges to the right state as evident from the slight dip in performance. On the other hand, while the performance of the exact HMM inference is marginally better, it has a large number of states. In this domain, the HMM had about 144 ground states and about 2200 observations. Thus the transition matrix is of the size 144 × 144 while the observation matrix is of the size 144 × 2200. It took about 3ms on an average to perform inference using the ground HMM and a sparse matrix representation while it was nearly 0ms using the filter. In larger domains, such as a real-time desktop assistant or video survelliance, it is likely that this state space can grow quickly and it could become infeasible for performing inference.

5.2 Kitchen Domain

The other domain is a kitchen domain where the user has to cook some dishes. The user has 2 kinds of higher-level goals: one in which he could prepare a recipe which contains a main dish and a side dish and the second in which he could use some instant food to prepare a main dish and a side dish. There are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from the recipe.

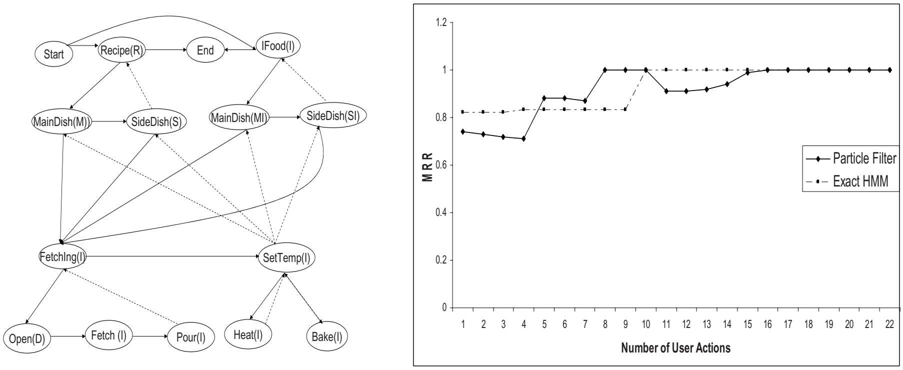


Fig. 3. (a)The kitchen LoHiHMM. (b)Results of the particle filter and the ground HMM inference.

Similarly, there are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from instant food. The LoHiHMM is presented in Figure 3.*a*.

There are 2 shelves with 3 ingredients each. The shelves have doors that must be opened before fetching ingredients and only one door can be open at a time. The observable part of the state consists of the contents of the bowl, the ingredient on the table, the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf.

As with the previous domain, we present the Mean Reciprocal Ratio of the correct state when using the particle filter and the exact inference using ground HMM in this domain. The 2 algorithms were executed for 50 runs where the user chooses the high-level goal randomly. In this domain, the performance of the particle filter is better than the grid world domain. As can be observed, the particle filter samples more correct states in the beginning and hence has a better performance compared to the exact inference method. But like the previous domain, once a sub-goal has been achieved, the filter samples some of the incorrect states thus having a small dip in the MRR but then recovers by using the observations to converge quickly on the true state. The ground HMM had a larger number of states(64) and observations (> 700) respectively leading to large transition and observation matrices and took about 2.5ms per inference query when compared to nearly 0ms for the particle filter.

6 Conclusion and Future Work

In this work, we motivated the need for the combination of the logical and hierarchical versions of HMMs for performing activity recognition. To this effect, we have developed and presented Logical Hierarchical HMMs. We outlined the syntax and semantics of the conditional branching for the vertical and the horizontal transitions that depend on certain attributes of the environment. We also outlined a particle filter algorithm to perform efficient inference on these models and presented a procedure for unrolling the LoHiHMMs to HMMs. We then evaluated our algorithms on 2 simulated domains and showed that the particle filter can perform efficiently while sacrificing a small amount of accuracy.

Zettlemoyer et al. [13] consider the problem of filtering in relational HMMs and propose the idea of logical particle filter for this case. In their model, the states are conjunctions of propositions and/or functions. The observations and the actions are propositions (i.e., unparameterized). The basic idea is to construct a set of partitions (hypotheses) where each of the partitions represents a set of states such that every state in a hypothesis has the same transition probability. The hypotheses at the current time step are split into a set of mutually exclusive ones, the transition probability is then applied and the observations are used to specialize the hypothesis. The expectation over the states in a hypothesis is computed analytically while the hypotheses themselves are sampled. The main bottleneck lies in the construction of these hypothesis. It is not clear how to construct these mutually exclusive hypothesis in the presence of function symbols for the state space. We sacrifice expressiveness for efficiency in our work and consider only predicate symbols for the state space, hence making the particle filter simpler. Natarajan et.al [12] propose the use of relational hierarchies for specifying prior knowledge to an assistant. We consider a representation that is not specific to the assistant setting but propose a general method for extending the HMMs in the relational and hierarchical settings.

One of the important directions for future research is to improve the accuracy of the particle filter. One possible solution that we are currently exploring is the use of Rao-Blackwellization (similar to that of Zettlemoyer et al. but in a simpler setting using predicates for the states) to analytically marginalize out part of the state space and then sample the rest. It is possible to sample the predicates at the next time-step while inferring the values of the variables exactly. This would lead to improved inference while not increasing the size of the state space drastically. Recently Milch and Russell [9] presented an Metropolis-Hastings based MCMC algorithm for relational structures. The algorithm considers states as partial description of the world and uses context specific independences among the state variables to factor out the acceptance probabilities. It remains an interesting future work to explore the use of MCMC for the inference in LoHiHMMs. Such a direction would provide more insights into the relative merits of particle filtering and MCMC in relational domains. We are currently working on automating the grounding of the LoHiHMMs to a ground HMM.

Yet another important direction is to evaluate on real-world domains. Currently the logical version of the model (LoHMM) is being deployed and evaluated on a real-world assistant (name withheld for blind review) for activity recognition and initial results are promising. Our future goal is to use the LoHiHMMs for the same purpose. In many real-world domains, the variables can possibly take infinite values (such as filenames of the documents). There is a need for the logical representations to handle these infinite variables. We are exploring methods that can make our particle filter to handle these infinite variables in a principled way. The principle advantage of Logical models lie in the fact that they can exploit parameter tying while learning and our future goal is to design efficient learning algorithms for LoHiHMMs.

Acknowledgements. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), under Contract Nos. FA8750-07-D-0185/0004, FA8650-06-C-7606, and NBCHD030010. Kristian Kersting is partially supported by a Fraunhofer ATTRACT fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, the Air Force Research Laboratory (AFRL), or the Department of Interior-National Business Center (DOI-NBC).

References

1. Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., Mihailidis, A.: A decision-theoretic approach to task assistance for persons with dementia. In: IJCAI (2005)
2. Bui, H., Phung, D., Venkatesh, S.: Hierarchical hidden markov models with general state hierarchy. In: Proceedings of AAAI 2004 (2004)
3. Bui, H., Venkatesh, S., West, G.: Policy recognition in the abstract hidden markov models. JAIR 17 (2002)
4. Doucet, A., De Freitas, N., Gordon, N. (eds.): Sequential Monte Carlo methods in practice (2001)
5. Fine, S., Singer, Y., Tishby, N.: The hierarchical hidden markov model: Analysis and applications. Machine Learning 32, 41–62 (1998)
6. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
7. Kersting, K., De Raedt, L., Raiko, T.: Logical hidden markov models. JAIR 25, 425–456 (2006)
8. Leo, M., D'Orazio, T., Spagnolo, P.: Human activity recognition for automatic visual surveillance of wide areas. In: VSSN 2004: Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks (2004)
9. Milch, B., Russell, S.: General-purpose mcmc inference over relational structures. In: Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI 2006) (2006)
10. Murphy, K., Paskin, M.: Linear time inference in hierarchical HMMs. In: Proceedings of Neural Information Proceesing Systems (2001)
11. Myers, K., Berry, P., Blythe, J., Conleyn, K., Gervasio, M., McGuinness, D., Morley, D., Pfeffer, A., Pollack, M., Tambe, M.: An intelligent personal assistant for task and time management. AI Magazine (2007)
12. Natarajan, S., Tadepalli, P., Fern, A.: A relational hierarchical model for decision-theoretic assistance. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 175–190. Springer, Heidelberg (2008)
13. Zettlemoyer, L.S., Pasula, H.M., Kaelbling, L.P.: Logical particle filtering. In: Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning (2007)

Learning with Kernels in Description Logics

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito

LACAM – Dipartimento di Informatica, Università degli studi di Bari
Campus Universitario, Via Orabona 4, 70125 Bari, Italy
{fanizzi,claudia.damato,esposito}@di.uniba.it

Abstract. We tackle the problem of statistical learning in the standard knowledge base representations for the Semantic Web which are ultimately expressed in description Logics. Specifically, in our method a kernel functions for the $\mathcal{ALCN}$ logic integrates with a support vector machine which enables the usage of statistical learning with reference representations. Experiments where performed in which kernel classification is applied to the tasks of resource retrieval and query answering on OWL ontologies.

1 Learning from Ontologies

The Semantic Web (SW) represents an emerging applicative domain where knowledge intensive automated manipulations on complex relational descriptions are foreseen. Although machine learning techniques may have a great potential in this field, so far research has focused mainly on methods for knowledge acquisition from text (*ontology learning*) [4]. Yet machine learning methods can be transposed from ILP to be applied to ontologies described with formal concept representations employed to model knowledge bases in the SW.

Description Logics (DLs) [1] is a family of languages that has been adopted as the core technology for representing ontologies. Such languages are endowed with an open-world semantics which makes them particularly fit for the SW applications where, differently from classical DB contexts, knowledge bases are considered inherently incomplete, since new resources may continuously made available across the Web. Thus, few methods have been proposed for learning these representations (e.g. see [5, 13, 8, 14]).

While classic ILP techniques have been adapted to work with DLs representations, purely logic approaches often fall short in terms of efficiency and noise-tolerance. Learning with logic-based methods is inherently intractable in multi-relational settings. Moreover, for the sake of tractability, only very simple DL languages have been considered so far [6]. Recently, it has been shown that kernel methods may be effectively applied to structured representations [10] and also to ontology languages [9, 2].

In this work, a family of kernel functions is defined for DLs representations. Specifically, the $\mathcal{ALCN}$ logic [1] is adopted as a tradeoff between efficiency and expressiveness. The kernel functions are defined encoding a notion of similarity between objects expressed in this representation, which is based on structural

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 210–225, 2008.

and semantic aspects of the reference language, namely a normal form for the concept descriptions and the extension of concepts approximated through the objects that are (directly or provably) known to belong to them.

By coupling the kernel functions with support vector machines (SVMs) many tasks can be tackled. Particularly, we demonstrate how to perform important inference services based on inductive classification, namely concept retrieval and approximate query answering [1], that may turn out to be hard for logic methods, especially with knowledge bases built from heterogeneous sources. These tasks are generally grounded on merely deductive procedures which easily fail in case of (partially) inconsistent or incomplete knowledge. We show how inductive methods perform comparably well w.r.t. a standard deductive reasoner, allowing the suggestion of new knowledge that is not logically derivable similarly to *abductive* conclusions.

An experimentation on both artificial and real ontologies available in standard repositories proves the effectiveness of inductive classification using the kernel function integrated with a SVM.

The paper is organized as follows. After recalling the basics of the DLs representation (Sect. 2), we introduce relational kernels for the $\mathcal{ALCN}$ logic in Sect. 3. The application of kernel-based classification for inductive resource retrieval is presented in Sect. 4 and an experimental evaluation of the method is reported in Sect. 5. Finally, Sect. 6 concludes and outlines further applications and extensions of this work.

2 Reference Representation

The basics of the $\mathcal{ALCN}$ logic will be recalled (see [1] for a thorough reference). Such a logic is endowed with the basic constructors employed by the standard ontology languages adopted in the SW (such as OWL).

2.1 Knowledge Bases in Description Logics

Concept descriptions are inductively defined starting with a set $N_C = \{C, D, \ldots\}$ of *primitive concept* names, a set $N_R = \{R, Q, \ldots\}$ of *primitive roles* and a set of names for the *individuals* (objects, resources) $N_I = \{a, b, \ldots\}$. Complex descriptions are built using primitive concepts and roles and the language constructors. The set-theoretic semantics of these descriptions is defined by an *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set, the *domain* of the interpretation, and $\cdot^\mathcal{I}$ is the *interpretation function* that maps each $A \in N_C$ to a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and each $R \in N_R$ to $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$.

Complex descriptions can be built in $\mathcal{ALCN}$ using the language constructors listed in Table 1, along with their semantics derived from the interpretation of atomic concepts and roles [1]. Note that the *open-world assumption* (OWA) is made.

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. $\mathcal{T}$ is the set of definitions[1] $C \equiv D$, meaning $C^\mathcal{I} = D^\mathcal{I}$, where C is the concept name

[1] More general definitions of concepts by means of inclusion axioms ($C \sqsubseteq D$) may also be considered.

Table 1. Syntax and semantics of concepts in the $\mathcal{ALCN}$ logic

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
full negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
c. conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
c. disjunction	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
existential r.	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$
universal r.	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}((x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$
at least r.	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\rvert\} \geq n\}$
at most r.	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\rvert\} \leq n\}$

and D is its description. $\mathcal{A}$ contains assertions on the world state concerning the individuals, e.g. $C(a)$ and $R(a,b)$, meaning that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Example 2.1 (Royal Family). This example shows a knowledge base modeling concepts and roles related to the British royal family[2]:

$$\begin{aligned}\mathcal{T} = \{\ & \mathsf{Male} \equiv \neg\mathsf{Female},\\ & \mathsf{Woman} \equiv \mathsf{Human} \sqcap \mathsf{Female},\\ & \mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male},\\ & \mathsf{Mother} \equiv \mathsf{Woman} \sqcap \exists\mathsf{hasChild}.\neg\mathsf{Human},\\ & \mathsf{Father} \equiv \mathsf{Man} \sqcap \exists\mathsf{hasChild}.\neg\mathsf{Human},\\ & \mathsf{Parent} \equiv \mathsf{Father} \sqcup \mathsf{Mother},\\ & \mathsf{Grandmother} \equiv \mathsf{Woman} \sqcap \exists\mathsf{hasChild}.\neg\mathsf{Parent},\\ & \mathsf{Mother\text{-}w/o\text{-}daughter} \equiv \mathsf{Mother} \sqcap \forall\mathsf{hasChild}.\neg\mathsf{Female},\\ & \mathsf{Super\text{-}mother} \equiv \mathsf{Mother} \sqcap \geq 3.\mathsf{hasChild}\ \}\end{aligned}$$

$$\begin{aligned}\mathcal{A} = \{\ & \mathsf{Woman(elisabeth)},\ \mathsf{Woman(diana)},\ \mathsf{Man(charles)},\ \mathsf{Man(edward)},\\ & \mathsf{Man(andrew)},\ \mathsf{Mother\text{-}w/o\text{-}daughter(diana)},\\ & \mathsf{hasChild(elisabeth,\ charles)},\ \mathsf{hasChild(elisabeth{,}edward)},\\ & \mathsf{hasChild(elisabeth,\ andrew)},\ \mathsf{hasChild(diana,\ william)},\\ & \mathsf{hasChild(charles,\ william)}\ \}\end{aligned}$$

2.2 Inference Services

Many inference services are supported by a growing number of DL resoners. The principal inference service amounts to assessing whether a concept *subsumes* another concept according to their semantics:

Definition 2.1 (subsumption). *Given two descriptions C and D, C subsumes D, denoted by $C \sqsupseteq D$, iff for every interpretation $\mathcal{I}$ it holds that $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. When $C \sqsupseteq D$ and $D \sqsupseteq C$ then they are equivalent, denoted with $C \equiv D$.*

[2] From Franconi's DLs course: http://www.inf.unibz.it/~franconi/dl/course

Normally subsumption is computed w.r.t. the interpretations satisfying the knowledge base. More expressive languages allo for the construction or role-hierarchies based on subsumption.

Another important inference, since we aim at inductive methods that manipulate single resources, is *instance checking*, that amounts to deciding whether an individual belongs to the extension of a given concept [1]. Another related inference is *retrieval* which consists in finding the extension of a given concept:

Definition 2.2 (retrieval). *Given an knowledge base $\mathcal{K}$ and a concept C, find all individuals a such that $\mathcal{K} \models C(a)$.*

Conversely, it may be necessary to find the concepts which an individual belongs to (*realization problem*), especially the most specific one:

Definition 2.3 (most specific concept). *Given an ABox $\mathcal{A}$ and an individual a, the* most specific concept *of a w.r.t. $\mathcal{A}$ is the concept C, denoted $\mathsf{MSC}_{\mathcal{A}}(a)$, such that $\mathcal{A} \models C(a)$ and for any other concept D such that $\mathcal{A} \models D(a)$, it holds that $C \sqsubseteq D$.*

For some languages, the MSC may not be expressed by a finite description [1], yet it may be approximated by a more general concept. Generally approximations up to a certain depth k of nested levels are considered, denoted MSC^k. A maximal depth approximation will be generically indicated with MSC^*.

2.3 Normal Form

Many semantically equivalent (yet syntactically different) descriptions can be given for the same concept. Equivalent concepts can be reduced to a normal form by means of rewriting rules that preserve their equivalence [1]. We will adopt a normal form derived from [3].

Some notation is necessary for naming the various parts of a description:

- $\mathsf{prim}(C)$ is the set of all the primitive concepts (or their negations) at the top-level of C;
- $\mathsf{val}_R(C) = C_1 \sqcap \cdots \sqcap C_n$ if there exists a value restriction $\forall R.(C_1 \sqcap \cdots \sqcap C_n)$ at the top-level of C, otherwise $\mathsf{val}_R(C) = \top$;
- $\mathsf{ex}_R(C)$ is the set of the descriptions C' appearing in existential restrictions $\exists R.C'$ at the top-level conjunction of C.
- $\mathsf{min}_R(C) = \max\{n \in \mathbb{N} \mid C \sqsubseteq (\geq n.R)\}$ (always a finite number);
- $\mathsf{max}_R(C) = \min\{n \in \mathbb{N} \mid C \sqsubseteq (\leq n.R)\}$ (if unlimited then $\mathsf{max}_R(C) = \infty$).

A normal form may be recursively defined as follows:

Definition 2.4 ($\mathcal{ALCN}$ normal form). *A concept description C is in $\mathcal{ALCN}$* normal form *iff $C = \bot$ or $C = \top$ or if $C = C_1 \sqcup \cdots \sqcup C_n$ with*

$$C_i = \bigsqcap_{P \in \mathsf{prim}(C_i)} P \sqcap \bigsqcap_{R \in N_R} \left[\forall R.\mathsf{val}_R(C_i) \sqcap \bigsqcap_{E \in \mathsf{ex}_R(C_i)} \exists R.E \sqcap \geq m_i^R.R \sqcap \leq M_i^R.R \right]$$

where, for all $i = 1, \ldots, n$, $m_i^R = \mathsf{min}_R(C_i)$, $M_i^R = \mathsf{max}_R(C_i)$, $C_i \not\equiv \bot$ *and, for all* $R \in N_R$, $\mathsf{val}_R(C_i)$ *and every sub-description in* $\mathsf{ex}_R(C_i)$ *are, in their turn, in* $\mathcal{ALCN}$ *normal form.*

This normal form can be obtained by means of a repeated application of equivalence preserving operations, namely replacing defined concepts with their definition as in the TBox and pushing the negation in the nested level (*negation normal form*).

Example 2.2 (normal form). The concept description
$C \equiv (\neg A_1 \sqcap A_2) \sqcup (\exists R_1.B_1 \sqcap \forall R_2.(\exists R_3.(\neg A_3 \sqcap B_2)))$
is in normal form, whereas the following is not:
$D \equiv A_1 \sqcup B_2 \sqcap \neg(A_3 \sqcap \exists R_3.B_2) \sqcup \forall R_2.B_3 \sqcap \forall R_2.(A_1 \sqcap B_3)$
where A_i's and B_j's are primitive concept names and the R_k's are role names.

3 Defining Kernels for $\mathcal{ALCN}$

A family of valid kernels for the space $\mathcal{X}$ of $\mathcal{ALCN}$ descriptions can be proposed, based on the family defined for $\mathcal{ALC}$ [9]. The definition is based on the AND-OR tree structure of the descriptions in normal form, like for the standard tree kernels [10] where similarity between trees depends on the number of similar subtrees (or paths unraveled from such trees). Yet this would end in a merely syntactic measure which does not fully capture the semantic nature of expressive DLs languages such as $\mathcal{ALCN}$.

Normal form descriptions can be decomposed level-wise into sub-descriptions. There are three possibilities for each level: the upper level is dominated by the disjunction of concepts that, in turn, are made up of a conjunction of complex or primitive concepts. In the following the definition of the $\mathcal{ALCN}$ kernels (parametrized on the decaying factor λ) is reported.

Definition 3.1 ($\mathcal{ALCN}$ kernels). *Given an interpretation* $\mathcal{I}$ *of* $\mathcal{K}$*, the* $\mathcal{ALCN}$ kernel *based on* $\mathcal{I}$ *is the function* $k_\mathcal{I} : \mathcal{X} \times \mathcal{X} \mapsto \mathbf{R}$ *structurally defined as follows: given two disjunctive descriptions* $D_1 = \bigsqcup_{i=1}^n C_i^1$ *and* $D_2 = \bigsqcup_{j=1}^m C_j^2$ *in* $\mathcal{ALCN}$ *normal form:*
disjunctive descriptions:

$$k_\mathcal{I}(D_1, D_2) = \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} k_\mathcal{I}(C_i^1, C_j^2)$$

with $\lambda \in]0, 1]$
conjunctive descriptions:

$$k_\mathcal{I}(C^1, C^2) = \prod_{\substack{P_1 \in \mathsf{prim}(C^1) \\ P_2 \in \mathsf{prim}(C^2)}} k_\mathcal{I}(P_1, P_2) \cdot \prod_{R \in N_R} k_\mathcal{I}((m_{C^1}^R, M_{C^1}^R), (m_{C^2}^R, M_{C^2}^R)) \cdot$$
$$\prod_{R \in N_R} k_\mathcal{I}(\mathsf{val}_R(C^1), \mathsf{val}_R(C^2)) \cdot \prod_{R \in N_R} \sum_{\substack{C_i^1 \in \mathsf{ex}_R(C^1) \\ C_j^2 \in \mathsf{ex}_R(C^2)}} k_\mathcal{I}(C_i^1, C_j^2)$$

where $m^R_{C^i} = \mathsf{min}_R(C^i)$ *and* $M^R_{C^i} = \mathsf{max}_R(C^i)$, $i = 1, 2$.

numeric restrictions:

$$k_{\mathcal{I}}((m_C, M_C), (m_D, M_D)) = \frac{\min(M_C, M_D) - \max(m_C, m_D) + 1}{\max(M_C, M_D) - \min(m_C, m_D) + 1}$$

if $\min(M_C, M_D) > \max(m_C, m_D)$ *and* $k_{\mathcal{I}}((m_C, M_C), (m_D, M_D)) = 0$ *otherwise.*

primitive concepts:

$$k_{\mathcal{I}}(P_1, P_2) = k_{\text{set}}(P_1^{\mathcal{I}}, P_2^{\mathcal{I}}) = |P_1^{\mathcal{I}} \cap P_2^{\mathcal{I}}|$$

where k_{set} *is the kernel for set structures defined in [10]. This case includes also the negation of primitive concepts using:* $(\neg P)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus P^{\mathcal{I}}$

This kernel computes the similarity between disjunctive as the sum of the cross-similarities between any couple of disjuncts from either description. The rationale for this kernel is that similarity between disjunctive descriptions is treated by taking the sum of the cross-similarities between any couple of disjuncts from either description. The term λ is employed to downweight the similarity of the sub-descriptions on the grounds of the level where they occur.

Following the normal form, the conjunctive kernel computes the similarity between two input descriptions distinguishing for factors corresponding to primitive concepts, universal, existential and numeric restrictions, respectively. These values are multiplied reflecting the fact that all the restrictions have to be satisfied at a conjunctive level. Not that the values computed for value and existential restrictions involve recursive calls to the kernel functions on less complex structures.

The similarity of the numeric restrictions is simply computed as a measure of the overlap between the two intervals. Namely it is the ratio of the amounts of individuals in the overlapping interval and those the larger one, whose extremes are minimum and maximum. Note that some intervals may be unlimited above: $\max = \infty$. In this case we may approximate with an upper limit N greater than $|\Delta^{\mathcal{I}}| + 1$.

Finally, the similarity between primitive concepts is measured in terms of the intersection of their extension. Making the *unique names assumption* on the names of the individual occurring in the ABox $\mathcal{A}$, one can consider the *canonical interpretation* [1] $\mathcal{I}$, using $\mathsf{Ind}(\mathcal{A})$ as its domain ($\Delta^{\mathcal{I}} := \mathsf{Ind}(\mathcal{A})$). Therefore, the kernel can be specialized as follows: since the kernel for primitive concepts is essentially a set kernel we can set the constant λ_p to $1/\Delta^{\mathcal{I}}$ so that the cardinality of he intersection is weighted by the number of individuals occurring in the overall ABox. Alternatively, another choice could be $\lambda_P = 1/|P_1^{\mathcal{I}} \cup P_2^{\mathcal{I}}|$ which would weight the rate of similarity (the extension intersection) measured by the kernel with the size of the concepts measured in terms of the individuals belonging to their extensions.

Being partially based on the concept structure and only ultimately on the extensions of the concepts at the leaves, it may be objected that this kernel function can roughly grasp the concept similarity based on their semantics. This may be well revealed by the case of input concepts that are semantically almost equivalent yet structurally different. However, it must be pointed out that the process of rewriting for putting the concepts in normal form tends to eliminate these differences. More importantly, the ultimate goal for defining a kernel will be comparing individuals rather than concepts. This will be performed recurring to the most specific concepts of the individuals w.r.t. the same ABox. Hence, it was observed that semantically similar individuals will tend to share the same structures as elicited from the same source.

3.1 Discussion

The validity of a kernel depends on the fact that the function is *positive definite*. Positive definiteness can be also proved exploiting some closure properties of the class of positive definite kernel functions [11]. Namely, multiplying a kernel by a constant, adding or multiplying two kernels yields another valid kernel. We can demonstrate that the function introduced above is indeed a valid kernel for our space of hypotheses. Observe that the core function is the one on primitive concept extensions. It is essentially a set kernel [10]. The versions for top-level conjunctive and disjunctive descriptions are also positive definite being essentially based on the primitive kernel. Descending through the levels there is an interleaving of the employment of these function up the the basic case of the function for primitive descriptions.

Exploiting these closure properties it could be pr oven[3] that:

Proposition 3.1. *Given an interpretation $\mathcal{I}$, the function $k_{\mathcal{I}}$ is a valid kernel for the space $\mathcal{X}$ of $\mathcal{ALCN}$ descriptions in normal form.*

As regards efficiency, it is possible to show that the kernel function can be computed in time $O(|N_1||N_2|)$ where $|N_i|$, $i = 1, 2$, is the number of nodes of the concept AND-OR trees. It can computed by means of dynamic programming. Knowledge Base Management Systems, especially those dedicated to storing instances, generally maintain information regarding concepts and instances which may further speed-up the computation.

The kernel can be extended to the case of individuals $a, b \in \mathsf{Ind}(\mathcal{A})$ simply by taking into account the approximations of their MSCs:

$$k_{\mathcal{I}}(a, b) = k_{\mathcal{I}}(\mathsf{MSC}^*(a), \mathsf{MSC}^*(b))$$

In this way, we move from a graph representation like the ABox portion containing an individual to an intensional tree-structured representation.

Observe that the kernel function could be specialized to take into account the similarity between different relationships. This would amount to considering each

[3] Proof omitted for brevity.

couple of existential and value restriction with one element from each description (or equivalently from each related AND-OR tree) and the computing the convolution of the sub-descriptions in the restriction. As previous suggested for λ, this should be weighted by a measure of similarity between the roles measured on the grounds of the available semantics. We propose therefore the following weight: given two roles $R, S \in N_R$: $\lambda_{RS} = |R^{\mathcal{I}} \cap S^{\mathcal{I}}|/|\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}|$.

As suggested before, the intersection could be measured on the grounds of the relative role extensions with respect to the whole domain of individuals, as follows: $\lambda_{RS} = |R^{\mathcal{I}} \cap S^{\mathcal{I}}|/|R^{\mathcal{I}} \cup S^{\mathcal{I}}|$. It is also worthwhile to recall that some DLs knowledge bases support also the so called *R-box* [1] with assertions concerning the roles, thus we might know beforehand that for instance $R \sqsubseteq S$ and compute heir similarity consequently.

The extension of the kernel function to more expressive DL is not trivial. DLs allowing normal form concept definitions can only be considered. Moreover, for each constructor not included in the $\mathcal{ALCN}$ logic, a kernel definition has to be provided.

Related distance measures can also be derived from kernel functions which essentially encode a notion of similarity between concepts and between individuals. This can enable the definition of various distance-based methods for these complex representations spanning from clustering to instance-based methods.

4 Inductive Classification and Retrieval

In this paper, a kernel method is used to solve the following classification problem:

Definition 4.1 (classification problem). *Given a knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$*, the set of individuals* Ind *and a set of concepts* $DC = \{C_1, \ldots, C_s\}$ *defined on the grounds of those in* $\mathcal{K}$*, the primal problem to solve is: considered an individual* $a \in \mathsf{Ind}$ *determine the subset of concepts* $\{C_1, \ldots, C_t\} \subseteq DC$ *to which* a *belongs to.*

This classification problem can be also be regarded as a retrieval problem with the following dual definition:

Definition 4.2 (retrieval problem). *Given a knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$*, a query concept* Q *defined on the grounds of those in* $\mathcal{K}$ *and the set of individuals in the ABox* $\mathsf{Ind}(\mathcal{A})$*, the dual problem to solve is: find all* $b \in \mathsf{Ind}(\mathcal{A})$ *such that* $\mathcal{K} \models Q(b)$*.*

In the general learning setting, the target classes are disjoint. This is not generally verified in the SW context, where an individual can be instance of more than one concept in the hierarchy. To solve this problem, a new answering procedure is proposed. It is based on the decomposition of the multi-class problem into smaller binary classification problems (one per class). Therefore, a simple binary value set ($V = \{-1, +1\}$) can be employed, where ($+1$) indicates that an example x_i occurs in the ABox w.r.t. the considered concept C_j (namely $C_j(x_i) \in \mathcal{A}$);

(-1) indicates the absence of the assertion in the ABox. As an alternative, it can be considered $+1$ when $C_j(x_i)$ can be inferred from the knowledge base, and -1 otherwise.

Another issue has to be considered. In the general classification setting an implicit assumption of *Closed World* is made. On the contrary, in the SW context the *Open World Assumption* is generally made. To deal with the OWA, the absence of information on whether a certain instance x_i belongs to the extension of concept C_j should not be interpreted negatively, as seen before, rather, it should count as neutral information. Thus, another value set has to be considered, namely $V = \{+1, -1, 0\}$, where the three values denote, respectively, assertion occurrence ($C_j(x_i) \in \mathcal{A}$), occurrence of the opposite assertion ($\neg C_j(x) \in \mathcal{A}$) and assertion absence in $\mathcal{A}$.

Occurrences can be easily computed with a lookup in the ABox. Moreover, as in the previous case, a more complex procedure may be devised by substituting the notion of occurrence (absence) of assertions in (from) the ABox with the one of derivability from the whole KB, i.e. $\mathcal{K} \vdash C_j(x_i)$, $\mathcal{K} \not\vdash C_j(x_i)$, $\mathcal{K} \not\vdash C_j(x_i)$ and $\mathcal{K} \not\vdash \neg C_j(x_i)$, respectively.

Although this may improve the precision of inductive reasoning, it is also more computationally expensive, since the simple lookup in the ABox must be replaced with instance checking. Hence, considered the query instance x_q, for every concept $C_j \in C$ the classifier will return $+1$ if x_q is an instance of C_j, -1 if x_q is an instance of $\neg C_j$, and 0 otherwise. The classification is performed on the ground of a set of training examples from which such information can be derived.

These results can be used to improve concept retrieval service. By classifying the individuals in the Abox w.r.t. all concepts, concept retrieval is performed exploiting an inductive approach. As will be experimentally shown in the following, the classifier, besides of having a comparable behavior w.r.t. a standard reasoner, is also able to induce new knowledge that is not logically derivable. Moreover it can be employed for the query answering task by determining, as illustrated above, the extension of a new query concept built from concepts and roles in the considered ontology.

Note that instance-checking, as performed by a reasoner is P-SPACE complete for the reference DL language [1]. Conversely, the inductive classification procedure is very efficient once the SVM has been trained. Most of the training time is actually devoted to the construction of the kernel matrix which gains a lot of speed-up exploiting statistics on concept extensions normally maintained by knowledge base management systems [12]. Moreover ad hoc data structures can be employed to make this process even more efficient.

5 Experimental Evaluation

The $\mathcal{ALCN}$ kernel function has implemented in Java and integrated with a support vector machine from the LIBSVM library[4].

[4] http://www.csie.ntu.edu.tw/~cjlin/libsvm

Table 2. Ontologies employed in the experiments

Ontology	DL lang.	#concepts	#object prop.	#datatype prop.
People	$\mathcal{ALCHIN}(D)$	60	14	1
University	$\mathcal{ALC}$	13	4	0
FSM	$\mathcal{SOF}(D)$	20	10	7
Family	$\mathcal{ALCF}$	14	5	0
Newspaper	$\mathcal{ALCF}(D)$	29	28	25
Wines	$\mathcal{ALCIO}(D)$	112	9	10
Science	$\mathcal{ALCIF}(D)$	74	70	40
S.-W.-M.	$\mathcal{ALCOF}(D)$	19	9	1
NTN	$\mathcal{SHIF}(D)$	47	27	8

In order to assess the value of the kernel integrated in a SVM at solving the retrieval problem, experiments have been carried out on a number of simple and more complex ontologies.

It is difficult to find in the literature similar methods for a comparison of the outcomes. A recent approach using simple kernels with SVM^{light} has been qualitatively evaluated [2]; unfortunately the data (drawn from two ontologies) are not publicly available. Namely the authors artificially populated a knowledge base and then assess the quality of the induced models for a selected number of concepts.

We preferred to carry out more extensive experiments on available knowledge bases with no random population involving all concepts and individuals of the ontology. As such experiments are more easily repeatable. The details of experimental settings and the outcomes are reported in the following.

5.1 Experimental Setup

The experiments have been performed on nine different ontologies (w.r.t. the domain and size) represented in OWL.

Namely, the family and university ontologies were developed in our lab[5] and populated with real data; the FSM, Surface-Water-Model, NewTestamentNames, Science, People, Newspaper and Wines ontologies were selected from the Protégé library[6]. Details about such ontologies are reported in Table 2. The number of individuals spans from 50 to 1000. However ontologies are normally measured in terms of triples; in the experiment we have ontologies whose size goes from hundreds up to ten thousands of triples.

These ontologies are represented in languages that are generally more complex than $\mathcal{ALCN}$. Hence, in order to apply the kernel function, constructs that are not allowed by $\mathcal{ALCN}$ were discarded during the computation of the MSCs of the individuals.

[5] http://lacam.di.uniba.it:8000/~nico/research/ontologymining.html

[6] http://protege.stanford.edu

The inductive classification method was applied to each ontology; namely, the individuals were checked to assess if they were instances of the concepts in the ontology using the classifier induced by the SVM adopting the $\mathcal{ALCN}$ kernel function (initially λ was set to 1). The performance of the classifier was evaluated comparing its responses to those returned by a standard reasoner[7] used as baseline.

Specifically, for each individual in the ontology the MSC was computed and enlisted in the set of training or test examples (individuals). For each concept, a model was built training the SVM with the kernel on the proper set of examples. Each test example was then classified applying the induced classifier. The experiment has been repeated for each concept adopting a ten-fold cross-validation procedure. Actually there were two sessions: in the first we tested the system on primitive and defined concepts in the ontology while in the second more complex random concepts were randomly built on the grounds of these concepts for testing the system.

For the evaluation, initially the standard measures of precision, recall, F-measure were considered. Yet in a setting complying with an open-world semantics cases when the resulting answer was unknown had to be considered more carefully, since they still might possibly imply a classification of an instance as relevant. Then we decided to measure a sort of alignment between the response given by the deductive reasoner and the one returned by our inductive classifier.

The running time (on a Core2Duo 2Ghz Linux box with 2GB RAM) goes from minutes to 1.2 hours for a run of 10-fold cross-validation procedure on the individuals belonging to a single ontology w.r.t. each test concept. That includes the time elapsed for getting the correct classification from the reasoner for the comparison.

Hence, for each concept in the ontology, the following parameters have been measured for the evaluation [7]:

- *match rate*: cases of individuals that got the same classification by both classifiers;
- *induction rate*: individuals that the classifier found to belong to the target concept or its negation, while this was not logically deducible;
- *omission error rate*: cases of individuals for which the inductive classifier omitted to determine whether they were instances (or not) of the target concept while this could be logically ascertained by the reasoner;
- *commission error rate*: amount of individuals labeled as instances of a given concept, while they (logically) do not belong to that concept or vice-versa.

Further experimental sessions, reported in the following section, were performed by varying the parameter λ. Besides, another experiment has regarded testing the performance of the classifier on random query concepts generated by composition of (primitive and defined) concepts in the knowledge base.

[7] Pellet ver. 1.5.1: http://pellet.owldl.com

Table 3. Outcomes of the concept classification experiments ($\lambda = 1$): average percentages and standard deviations

ONTOLOGY	match rate	induction rate	om. error rate	com. error rate
PEOPLE	86.61 (± 10.37)	5.40 (± 12.44)	7.99 (± 6.44)	0.0 (± 0.0)
UNIVERSITY	78.94 (± 9.78)	11.40 (± 8.65)	1.76 (± 6.09)	7.90 (± 7.26)
FSM	91.72 (± 9.32)	0.72 (± 2.79)	0.0 (± 0.0)	7.56 (± 9.53)
FAMILY	61.95 (± 20.36)	3.15 (± 11.37)	34.89 (± 22.88)	0.0 (± 0.0)
NEWSPAPER	90.33 (± 8.29)	0.0 (± 0.0)	9.67 (± 8.29)	0.0 (± 0.0)
WINES	95.58 (± 7.85)	0.43 (± 3.44)	3.99 (± 7.30)	0.0 (± 0.0)
SCIENCE	94.20 (± 7.91)	0.72 (± 1.61)	5.08 (± 8.31)	0.0 (± 0.0)
S.-W.-M.	87.09 (± 15.83)	6.73 (± 15.96)	6.18 (± 9.14)	0.0 (± 0.0)
N.T.N.	92.52 (± 24.71)	2.58 (± 8.44)	0.15 (± 3.90)	4.75 (± 11.28)

5.2 Outcomes

Classification with Concepts in the Knowledge Base. Table 3 reports the outcomes of this first experiment. The average rates obtained over all the concepts in each ontology are reported, jointly with their range.

It is important to note that, for every ontology, the commission error was quite low. This means that the classifier did not make critical mistakes, i.e. cases when an individual is deemed as an instance of a concept while it really is an instance of another disjoint concept. Particularly, the commission error rate is not null in case of UNIVERSITY and FSM ontologies and consequently also the match rate is the lowest. It is worthwhile to note that these ontologies have also a limited number of individuals. Specifically, the number of concepts is almost similar to the number of individuals, which represents a difficult situation in which there is not enough information for separating the feature space and then produce correct classifications. However, also in these conditions, the commission error was quite low, the matching rate is considerably high and the classifier was even able to induce new knowledge.

Interestingly, it was noticed that the match rate increased with the increase of the number of individuals in the considered ontology with a consequent strong decrease of the commission error rate that tends to 0 in for the most populated ontologies. For almost all ontologies the SVM classifier is also able to induce new knowledge, i.e. to assign an individual to a concept when this could not be decided by the deductive reasoner due to the open-world semantics.

For some ontologies some rates exhibit high standard deviations. In a careful insight of such cases, we found that this was due to cases of concepts with very few positive (negative) examples. This problem is made harder by the particular DL setting that allows many individuals to have an unknown classification w.r.t. some concepts because of the OWA. This is particularly true for the ontologies FAMILY and UNIVERSITY that were intentionally built as *sparse* knowledge bases (lots of class-membership assertions for the various individuals cannot be logically derived from the knowledge base).

Table 4. Outcomes of the concept classification experiments ($\lambda = .5$): average percentages and standard deviations

ONTOLOGY	match rate	induction rate	om. error rate	com. error rate
PEOPLE	86.61 (± 10.37)	5.4 (± 12.44)	7.99 (± 6.44)	0.0 (± 0.0)
UNIVERSITY	71.06 (± 13.36)	11.40 (± 8.65)	4.38 (± 15.18)	13.16 (± 8.56)
FSM	91.72 (± 9.32)	0.72 (± 2.79)	0.0 (± 0.0)	7.56 (± 9.53)
FAMILY	61.55 (± 20.47)	3.55 (± 12.81)	34.89 (± 22.88)	0.0 (± 0.0)
NEWSPAPER	90.38 (± 8.15)	0.0 (± 0.0)	9.62 (± 8.15)	0.0 (± 0.0)
WINES	95.15 (± 8.81)	0.65 (± 5.19)	4.21 (± 7.50)	0.0 (± 0.0)
SCIENCE	87.68 (± 12.71)	6.52 (± 12.61)	5.80 (± 9.85)	0.0 (± 0.0)
S.-W.-M.	86.18 (± 17.86)	8.01 (± 16.36)	5.81 (± 7.74)	0.0 (± 0.0)
NTN	90.52 (± 25.10)	4.27 (± 10.03)	4.90 (± 11.73)	0.31 (± 5.22)

Besides a stable behavior was also observed as regards the omission error rate which is very often non-null, yet very limited. This is due to a high number of training examples classified as unknown w.r.t. a certain class. To decrease the tendency to a conservative behavior of the classifier, a threshold could be introduced for the consideration of the training examples labeled with 0 (”unknown” classification).

The experiment has been repeated by setting the parameter λ of the kernel function to smaller values. For example, the average results when $\lambda = 0.5$ are reported in Table 4 (we omit the other results for sake of brevity). From this table, where the average rates w.r.t. the various ontologies are reported, we can note that the classification results are comparable with those of the previous experiment. Again it is possible to note that halving of the λ value does not generally influence the classification results.

Particularly, for ontologies with the lowest numbers of individuals (e.g. UNIVERSITY, FSM) the match rate sometimes also decreases w.r.t. the classification performed using $\lambda = 1$.

Random Query Concepts. Another experiment has been carried out, to test the kernel classifier as a means for performing inductive concept retrieval w.r.t. new query concepts built from the considered ontology. The method has been applied to solve a number of retrieval problems using $\lambda = 1$ for the kernel function. To this purpose, 15 queries were randomly generated by means of conjunctions / disjunctions of (3 thru 8) primitive and/or defined concepts of each ontology.

As for the previous experiment, a ten fold cross-validation setting was applied with the same nine ontologies. The individuals have been assigned to each of the three classes and the classifier induced by the SVM has been used to decide on the membership to the query class of the test individuals. The outcomes are reported in Table 5, from which it is possible to observe that the behavior of the classifier generally remains unvaried w.r.t. the previous experiment whose outcomes are reported in Table 3. As in the other experiments, they were repeated for different

Table 5. Outcomes of the experiments with random query concepts ($\lambda = 1$): average percentages and standard deviations

Ontology	match rate	induction rate	om. error rate	com. error rate
People	88.56 (± 9.30)	4.05 (± 10.50)	7.4 (± 6.26)	0.0 (± 0.0)
University	71.99 (± 12.15)	15.98 (± 8.18)	0.94 (± 4.97)	11.10 (± 11.22)
FSM	87.80 (± 10.83)	0.86 (± 2.39)	0.0 (± 0.0)	11.34 (± 10.80)
Family	66.33 (± 16.79)	4.53 (± 10.93)	29.14 (± 20.87)	0.0 (± 0.0)
Newspaper	77.91 (± 10.06)	0.0 (± 0.0)	22.09 (± 10.06)	0.0 (± 0.0)
Wines	94.33 (± 9.12)	0.0 (± 0.0)	5.67 (± 9.12)	0.0 (± 0.0)
Science	97.86 (± 1.61)	0.51 (± 1.36)	1.63 (± 1.64)	0.0 (± 0.0)
S.-W.-M.	80.39 (± 16.26)	13.40 (± 18.93)	6.21 (± 7.28)	0.0 (± 0.0)
NTN	90.58 (± 25.23)	2.18 (± 11.0)	7.19 (± 14.36)	0.40 (± 7.39)

Table 6. Outcomes of the experiments with random query concepts ($\lambda = .5$): average percentages and standard deviations

Ontology	match rate	induction rate	om. error rate	com. error rate
People	86.71 (± 8.97)	3.97 (± 10.75)	9.33 (± 6.49)	0.0 (± 0.0)
University	63.95 (± 18.56)	17.16 (± 10.08)	2.14 (± 11.14)	16.76 (± 13.06)
FSM	84.25 (± 12.40)	0.90 (± 3.05)	0.0 (± 0.0)	14.85 (± 12.84)
Family	66.38 (± 11.87)	7.86 (± 16.71)	28.77 (± 18.12)	0.0 (± 0.0)
Newspaper	81.81 (± 14.11)	0.92 (± 4.58)	17.28 (± 12.39)	0.0 (± 0.0)
Wines	89.46 (± 15.41)	5.20 (± 14.21)	1.95 (± 1.48)	0.0 (± 0.0)
Science	97.15 (± 0.87)	0.90 (± 1.80)	1.63 (± 1.64)	0.0 (± 0.0)
S.-W.-M.	84.86 (± 15.94)	8.6 (± 15.42)	6.88 (± 6.42)	0.0 (± 0.0)
NTN	89.11 (± 25.91)	5.17 (± 12.59)	5.35 (± 14.19)	0.37 (± 9.48)

values of λ. Table 6, reports the outcomes of such experiments for the case when $\lambda = .5$.

Summarizing, the $\mathcal{ALCN}$ kernel function can be effectively used, jointly with a SVM, to perform inductive concept retrieval, guaranteeing almost null commission error and, interestingly, the ability to induce new knowledge. The performance of the classifier increases with the increase of the number of individuals populating the considered ontology and the homogeneity of their spread across the concepts in the ontology.

These results are comparable to those obtained on an overlapping pool of datasets with a nearest neighbor classification method based on a semantic distance [7].

6 Conclusions and Future Work

We investigated multi-relational learning techniques in the SW peculiar context. Specifically, a kernel function has been defined for $\mathcal{ALCN}$ descriptions which was integrated with a SVM for inducing a statistical classifier working with this complex representation. The resulting classifier was tested on inductive retrieval and classification problems. Experimentally, it was shown that its performance is not only comparable to a standard deductive reasoner, but it is also able to induce new knowledge, which is not logically derivable. Particularly, an increase in prediction accuracy was observed when the instances are homogeneously spread.

The induced classifier can be exploited for predicting or suggesting missing information about individuals, thus completing large ontologies. Specifically, it can be used to semi-automatize the population of an ABox. Indeed, the new assertions can be suggested to the knowledge engineer that has only to validate their inclusion. This constitutes a new approach in the SW context, since the efficiency of the statistical and numerical approaches and the effectiveness of a symbolic representation have been combined.

The main weakness of the approach is on its scalability towards more complex DLs. While computing MSC approximations might be feasible, it may be more difficult focusing on a normal form when comparing descriptions. Indeed, as long as the expressiveness increases, the gap between syntactic structure semantics of the descriptions becomes more evident. As a next step, we can foresee the investigation of defining kernels for more expressive languages w.r.t. $\mathcal{ALCN}$, e.g. languages enriched with (qualified) number restrictions and inverse roles [1].

The derivation of distance measures from the kernel function may enable a series of further distance-based data mining techniques such as clustering and instance-based classification. Conversely, new kernel functions can be defined transforming newly proposed distance functions for these representations, which are not language dependent and allow the related data mining methods to better scale w.r.t. the number of individuals in the ABox.

References

[1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)

[2] Bloehdorn, S., Sure, Y.: Kernel methods for mining instance data in ontologies. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 58–71. Springer, Heidelberg (2007)

[3] Brandt, S., Küsters, R., Turhan, A.-Y.: Approximation and difference in description logics. In: Fensel, D., Giunchiglia, F., McGuinness, D., Williams, M.-A. (eds.) Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning, KR 2002, pp. 203–214. Morgan Kaufmann, San Francisco (2002)

[4] Buitelaar, P., Cimiano, P., Magnini, B. (eds.): Ontology Learning from Text: Methods, Evaluation And Applications. IOS Press, Amsterdam (2005)

[5] Cohen, W.W., Hirsh, H.: Learning the CLASSIC description logic. In: Torasso, P., Doyle, J., Sandewall, E. (eds.) Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning, pp. 121–133. Morgan Kaufmann, San Francisco (1994)

[6] Cumby, C.M., Roth, D.: On kernel methods for relational learning. In: Fawcett, T., Mishra, N. (eds.) Proceedings of the 20th International Conference on Machine Learning, ICML 2003, pp. 107–114. AAAI Press, Menlo Park (2003)

[7] d'Amato, C., Fanizzi, N., Esposito, F.: Query answering and ontology population: An inductive approach. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 288–302. Springer, Heidelberg (2008)

[8] Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-intensive induction of terminologies from metadata. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004)

[9] Fanizzi, N., d'Amato, C.: A declarative kernel for $\mathcal{ALC}$ concept descriptions. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 322–331. Springer, Heidelberg (2006)

[10] Gärtner, T., Lloyd, J.W., Flach, P.A.: Kernels and distances for structured data. Machine Learning 57(3), 205–232 (2004)

[11] Haussler, D.: Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California – Santa Cruz (1999)

[12] Horrocks, I.R., Li, L., Turi, D., Bechhofer, S.K.: The instance store: DL reasoning with large numbers of individuals. In: Haarslev, V., Möller, R. (eds.) Proceedings of the 2004 Description Logic Workshop, DL 2004. CEUR Workshop Proceedings, vol. 104, pp. 31–40. CEUR (2004)

[13] Kietz, J.-U., Morik, K.: A polynomial approach to the constructive induction of structural knowledge. Machine Learning 14(2), 193–218 (1994)

[14] Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894. Springer, Heidelberg (2008)

Querying and Merging Heterogeneous Data by Approximate Joins on Higher-Order Terms

Simon Price and Peter Flach

Department of Computer Science, University of Bristol, Bristol BS8 1UB, United Kingdom
{simon.price,peter.flach}@bristol.ac.uk

Abstract. Integrating heterogeneous data from sources as diverse as web pages, digital libraries, knowledge bases, the Semantic Web and databases is an open problem. The ultimate aim of our work is to be able to query such heterogeneous data sources as if their data were conveniently held in a single relational database. Pursuant to this aim, we propose a generalisation of joins from the relational database model to enable joins on arbitrarily complex structured data in a higher-order representation. By incorporating kernels and distances for structured data, we further extend this model to support approximate joins of heterogeneous data. We demonstrate the flexibility of our approach in the publications domain by evaluating example approximate queries on the CORA data sets, joining on types ranging from sets of co-authors through to entire publications.

1 Introduction

An increasingly important problem is the integration of data from sources as diverse as web pages, digital libraries, knowledge bases, the Semantic Web and databases that, collectively, are referred to as heterogeneous data. Integration allows an application to query the data using a single query language, just as if the data were a single homogeneous data source.

In this paper we combine two contrasting knowledge representational approaches into a single coherent formalism that is well suited to the integration of heterogeneous data. The first of these representational approaches, the relational model, is widely used as the basis for relational databases and is accompanied by a well-defined algebra for manipulating relational data. However, relational representations of complex structured data can be difficult to design and even more difficult for a human to read. The second representational approach, terms in a higher-order logic, offers a more human-readable representation of structured data than the relational model but has no well-defined analogue of the relational algebra for the querying of its terms. The formalism we introduce here is a subset of relational algebra upgraded for terms in a higher-order logic, bringing the well known and widely used join operator of relational algebra to the knowledge representational formalism of higher-order terms. This new algebra incorporates a generalisation of the relational model to higher-order terms and we show that a join operator from the relational model may be viewed as a special case of the higher-order join.

Data integration typically transforms heterogeneous data formats into a single homogeneous data format, usually into the format which has the most convenient algebra for data integration rather than the format with the most natural representation of the data.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 226–243, 2008.

In an ideal data integration scenario, where no uncertainty exists in the correspondences between individuals from different data sources, the homogeneous data format chosen is merely a technical implementation detail and places no restrictions on what may be reliably integrated. Unfortunately, the more diverse the sources of data being integrated, the more likely it will be that the integration involves a degree of uncertainty – for example, in identifying correspondences between individuals from different data sources. In order to automate such integration tasks, approximate matching techniques from statistics, machine learning and data mining may be employed. However, the transformation of data to the most widely used homogeneous format, relational data, obscures the data's natural type and structure, unnecessarily complicating the application of approximate matching techniques. In this paper, we show that approximate matching can take place without this obfuscating transformation to a relational representation. Our approach to approximate matching for data integration uses kernels and distances applied directly to representations of indivuals as (closed) terms in a higher-order logic.

The outline of the paper is as follows: Section 2 reformulates the traditional relational join from the relational model. Section 3 introduces the knowledge representational formalism and describes a family of kernels and distances on that formalism. Section 4 upgrades the relational join to handle structured data. Section 5 investigates the application of joins for structured data. The remaining sections review related work and future directions before concluding.

2 Relational Joins

We first define the relational join in its exact form and then adapt this to define the approximate relational join. Our definitions differ from those of the traditional relational model in that we have not embedded attribute names, which are in fact database-specific schema metadata, into the data representation. We assume throughout that there is an associated schema for each relation. By keeping the metadata separate from the data we achieve a more elegant upgrade from the relational model to the structured data model.

2.1 Exact Relational Joins

An n-tuple $(x_1, \ldots, x_n)$ is an element in $D_1 \times \cdots \times D_n$ where $x_1, \ldots, x_n$ are values drawn from domains $D_1, \ldots, D_n$ respectively. We refer to n-tuples as *tuples* unless the value of n is significant. Item $i \in \{1, \ldots, n\}$ of a tuple $t = (x_1, \ldots, x_n)$ is the value x_i and is written $t|_i$. A *relation* R of degree n is a finite set of n-tuples such that $R \subseteq D_1 \times \cdots \times D_n$ where $D_1, \ldots, D_n$ are domains, which need not necessarily be distinct. The *relation index* I_R of a relation R of degree n is the set $\{1, \ldots, n\}$.

Definition 1 (θ-Restriction). *Let θ be a predicate $\theta : D \times D \to \mathbb{B}$ for some domain D. If A and B are relations with tuple items $a|_i \in D$ and $b|_j \in D$ respectively for some $(i, j) \in I_A \times I_B$, then the* θ-restriction *$\sigma_{i\theta j}$ is defined on $T \subseteq A \times B$ as follows: $\sigma_{i\,\theta j}(T) = \{(a, b) \mid\ a|_i\, \theta\, b|_j\ \wedge\ (a, b) \in T\}$.*

The infix θ in the subscript of σ follows the historical convention from the relation database literature and so $i\,\theta j$, or equivalently $\theta(i, j)$, does not mean that θ applies to i and j; instead $i\,\theta j$ is shorthand notation for the membership test $a|_i\,\theta\; b|_j$ for all $(a, b) \in T$. The operator θ is typically drawn from the set $\{=, \neq, <, \leq, >, \geq\}$ but does not necessarily have to come from this set. θ-restriction is often just referred to as *restriction* and in such cases θ is assumed to be the equality operator. The name *selection* is often used instead of *restriction*, but the latter avoids confusion with the similarly named and better known `select` operator from SQL which has a somewhat different meaning. Restriction is also sometimes defined as *generalised restriction*: Let φ be a proposition that consists of atoms as allowed in θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$, then if A and B are relations, the generalised restriction σ_φ is defined on $T \subseteq A \times B$ as $\sigma_\varphi(T) = \{\ t \mid \varphi(t)\ \wedge\ t \in T\}$. Standard results show that generalised restrictions can always be expressed as combinations of θ-restrictions. θ-restriction is thus the basis of the following fundamental relational join operator.

Definition 2 (θ-Join). *Let θ be a predicate $\theta : D \times D \rightarrow \mathbb{B}$ for some domain D. If A and B are relations with tuple items $a|_i \in D$ and $b|_j \in D$ respectively for some $(i, j) \in I_A \times I_B$, then the θ-*join* $\bowtie_{i\,\theta j}$ of A and B is defined as $A \bowtie_{i\,\theta j} B = \sigma_{i\,\theta j}(A \times B)$.*

When θ is equality the θ-join is called the *equi-join*. By replacing the θ-restriction operator in the θ-join by the generalised restriction operator we arrive at the definition of the *generalised join*: Let φ be a proposition that consists of atoms as allowed in θ-restriction and the logical operators $\wedge$, $\vee$ and $\neg$. If A and B are relations then the *relational join* $\bowtie_\varphi$ is defined as $A \bowtie_\varphi B = \sigma_\varphi(A \times B)$. Other non-fundamental joins include the *natural join*, *semijoin*, *antijoin*, *outer joins* and *inner joins* [1,2]. For the purposes of upgrading relational joins to handle structured data, it is sufficient to consider just the θ-join and optionally, as a useful syntactic convenience, the generalised join.

2.2 Approximate Relational Joins

In order to turn an exact relational join into an approximate one it is necessary to replace the exact θ operator in θ-restriction with a suitable approximate version. For example, substituting exact equality $=$ with an approximate equality $\approx$ enables joining on tuple items that are either the same or in some way sufficiently similar.

One method of implementing approximate equality $\approx$ is to use a distance metric or pseudo-metric *dist*, defined on the domain of a pair of relational tuple items, together with a threshold δ to define a *proximity relation*.

Definition 3 (Proximity). *If the function $dist : D \times D \rightarrow \mathbb{R}$ is a distance on pairs of values from some domain D and $\delta \in \mathbb{R}$ $(\delta \geq 0)$ is a threshold then* proximity *is a predicate $\approx : D \times D \rightarrow \mathbb{B}$ defined by*

$$\forall x, y \in D\ \ (x \approx y) \iff \{(x, y) \mid dist(x, y) \leq \delta\ \wedge\ x, y \in D\}.$$

By the definition of distance, the co-domain of *dist* is not constrained to have an upper bound. Some normalising function φ may be used to apply an upper bound to a distance. The function $\varphi : \mathbb{R} \rightarrow \mathbb{R}$ must be a non-decreasing function from the positive reals into some closed interval, typically $[0, 1]$, such that $\varphi(0) = 0$, $\varphi(v) > 0$ if $v > 0$, and $\varphi(v + u) \leq \varphi(v) + \varphi(u)$, for each v and y. Example choices of φ from [3] are $\varphi(v) = min(v, 1)$ or $\varphi(v) = \frac{v}{v+1}$. Alternatively, the normalisation may be performed in the feature space of $x, y \in D$ so that $dist(x, y)$ is inherently normalised. For example, if the distance is derived from a kernel then a normalising kernel may be used [4].

Definition 4 (Proximity-Join). *Let* $\approx : D \times D \to \mathbb{B}$ *be a proximity for some domain* D*. If* A *and* B *are relations with tuple items* $a|_i \in D$ *and* $b|_j \in D$ *respectively for some* $(i, j) \in I_A \times I_B$*, the* proximity-join $\tilde{\bowtie}_{i \approx j}$ *of* A *and* B *is* $A \tilde{\bowtie}_{i \approx j} B = \sigma_{i \approx j}(A \times B)$.

The same historical notational convention is followed here for the subscripted $\approx$ as for the subscripted θ described earlier for the exact θ-join. The proximity-join as defined here is an approximate analogue of the exact relational equi-join. By choosing other proximity relations that are approximate analogues of exact relations, for example where $\theta \in \{=, \neq, <, \leq, >, \geq\}$, an approximate version of the relational θ-join might be defined. In this paper we restrict our attention to the proximity-join.

3 Representing Structured Data

The relational model is now the de facto standard for database-driven applications in data mining and computing in general, but it is not ideally suited for representing semi-structured data such as Web pages and XML, nor structured data such as the Semantic Web and RDF. However, the representation of such structures in relational databases is commonplace using a multitude of (often tortuous) representations and querying patterns. By contrast, the *individuals-as-terms* model offers straight forward representations of both structured and semi-structured data while at the same time having the representational capacity to represent relations from the relational model. The individuals-as-terms representation is a generalisation of the relational model's attribute-value representation and collects all information about an individual in a single term.

We are not advocating the individuals-as-terms repesentation as a replacement for the general purpose relational representation. But in the context of querying and merging heterogeneous data, the individuals-as-terms representation more transparently models the structure of the data in a way that is both human-readable and that explicitly exposes that structure to machine learning and data mining algorithms.

The knowledge representational formalism we use as our individuals-as-terms representation is *basic terms*, a family of typed terms in higher-order logic, which is based on Church's simple theory of types [5] with several extensions [3]. This formalism has been chosen over the possible alternative of first-order logic because terms in the higher-order logic natively support a variety of data types that are important for representing individuals, including sets, multisets and graphs. Being a strongly typed logic, helps to reduce search spaces and the type of terms provides useful metadata. The theory behind the logic and the individuals-as-terms formalism is set out in [3] and we give only a brief overview here.

We assume an *alphabet* consisting of: $\mathfrak{T}$ the set of type constructors of various arities, $\mathfrak{P}$ the set of parameters, $\mathfrak{C}$ the set of constants, and $\mathfrak{V}$ the set of variables. Included in $\mathfrak{T}$ is the constructor Ω of arity 0 with a corresponding domain of $\{True, False\}$, the booleans. Types are constructed from type constructors in $\mathfrak{T}$ and type variables in $\mathfrak{P}$ using the symbols $\to$ for function types and $\times$ for product types. A *type* is defined inductively as follows: (1) Each parameter in $\mathfrak{P}$ is a type. (2) If T is a type constructor in $\mathfrak{T}$ of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T\ \alpha_1 \ldots \alpha_k$ is a type. (For $k = 0$, this reduces to a type constructor of arity 0 being a type). (3) If α and β are types, then $\alpha \rightarrow \beta$ is a type. (4) If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type. (For $n = 0$, this reduces to 1 being a type). A type is *closed* if it contains no parameters. $\mathfrak{S}^C$ denotes the set of all closed types obtained from an alphabet.

The set of constants $\mathfrak{C}$ includes $\top$ (true) and $\bot$ (false). A *signature* is the declared type for a constant. A constant C with signature α is often denoted $C\ :\ \alpha$. Let $[]$ be the empty list constructor with signature *List* a where a is a parameter and *List* is a type constructor. Let $\#$ be the list constructor with signature $a \rightarrow$ *List* $a \rightarrow$ *List* a.

The *terms* of the logic are the terms of typed λ-calculus and are formed in the usual way by abstraction, tupling and application from constants in $\mathfrak{C}$ and a set of variables. The set of all terms obtained from a particular alphabet is denoted $\mathfrak{L}$. A basic term is the canonical representative of an equivalence class of terms [4,3]. The set of *basic terms*, $\mathfrak{B}$, is defined inductively as follows: (1) *Basic structures* – If C is a data constructor having signature $\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow (T\ a_i \ldots a_k)$, $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$, and t is $C\ t_1 \ldots t_n \in \mathfrak{L}$, then $t \in \mathfrak{B}$. (2) *Basic abstractions* – If $t_1, \ldots, t_n \in \mathfrak{B}$, $s_1, \ldots, s_n \in \mathfrak{B}$ $(n \geq 0)$, $s_0 \in \mathfrak{D}$ and t is λx if $x = t_1$ then s_1 else $\ldots$ if $x = t_n$ then s_n else $s_0 \in \mathfrak{L}$, then $t \in \mathfrak{B}$. (3) *Basic tuples* – If $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$ and t is $(t_1, \ldots, t_n) \in \mathfrak{L}$, then $t \in \mathfrak{B}$. See section 4 for examples and diagrams of basic terms.

3.1 Kernels and Distances for Basic Terms

Kernel functions [6] are an effective way of inducing distances on a wide variety of data structures. One promising recent kernel function for structured data is the default kernel for basic terms introduced in [4].

Definition 5 (Default Kernel for Basic Terms [4]). *The function $k : \mathfrak{B} \times \mathfrak{B} \rightarrow \mathbb{R}$ is defined inductively on the structure of terms in $\mathfrak{B}$ as follows.*

1. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = T \alpha_1 \ldots \alpha_k$, for some $T, \alpha_1, \ldots, \alpha_k$, then*
$$k(s,t) = \begin{cases} \kappa_T(C,D) & \textit{if } C \neq D \\ \kappa_T(C,C) + \sum_{i=1}^{n} k(s_i,t_i) & \textit{otherwise} \end{cases}$$
where s is $C\ s_1 \ldots s_n$ and t is $D\ t_1 \ldots t_m$.

2. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \beta \rightarrow \gamma$, for some β, γ, then*
$$k(s,t) = \sum_{\substack{u \in supp(s) \\ v \in supp(t)}} k(V(s\ u), V(t\ v)) \cdot k(u,v).$$

3. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \alpha_1 \times \cdots \times \alpha_n$, for some $\alpha_1, \ldots, \alpha_n$, then*
$$k(s,t) = \sum_{i=1}^{n} k(s_i,t_i),$$
where s is $(s_1, \ldots, s_n)$ and t is $(t_1, \ldots, t_n)$.

4. *If there does not exist $\alpha \in \mathfrak{S}^c$ such that $s, t \in \mathfrak{B}_\alpha$, then $k(s,t) = 0$.*

The definition, assumes that for each type constructor $T \in \mathfrak{T}$, $\kappa_T : \mathcal{X}_T \times \mathcal{X}_T \to \mathbb{R}$ is a kernel on the set of data constructors $\mathcal{X}_T$ associated with T. Below we give an example of the calculation of the default kernel for an example data structure: sets of strings.

Example 1 (Default Kernel on Sets of Strings). Let S be a nullary type constructor for strings and $A, B, C, D : S$. Choose κ_S and κ_Ω to be the matching kernel. Let s be the set $\{A, B, C\} \in \mathfrak{B}_{S \to \Omega}$, $t = \{A, D\}$, and $u = \{B, C\}$. Then

$$\begin{aligned}
k(s,t) &= k(A,A)k(\top,\top) + k(A,D)k(\top,\top) + k(B,A)k(\top,\top) \\
&\qquad + k(B,D)k(\top,\top) + k(C,A)k(\top,\top) + k(C,D)k(\top,\top) \\
&= \kappa_S(A,A)\kappa_\Omega(\top,\top) + \kappa_S(A,D)\kappa_\Omega(\top,\top) + \kappa_S(B,A)\kappa_\Omega(\top,\top) \\
&\qquad + \kappa_S(B,D)\kappa_\Omega(\top,\top) + \kappa_S(C,A)\kappa_\Omega(\top,\top) + \kappa_S(C,D)\kappa_\Omega(\top,\top) \\
&= \kappa_S(A,A) + \kappa_S(A,D) + \kappa_S(B,A) + \kappa_S(B,D) \\
&\qquad + \kappa_S(C,A) + \kappa_S(C,D) \\
&= 1 + 0 + 0 + 0 + 0 + 0 \\
&= 1.
\end{aligned}$$

Similarly, $k(s,u) = 2$ and $k(t,u) = 0$.

Noting that valid positive semi-definite kernels induce pseudo-metrics [4], this allows the derivation of a distance from any such kernel, including the kernel for basic terms, as follows. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel on $\mathcal{X}$. The distance measure induced by k is defined as $d_k(s,t) = \sqrt{k(s,s) - 2k(s,t) + k(t,t)}$. If k is a valid kernel the d_k is well behaved in that it satisfies the conditions of a pseudo-metric. Continuing the earlier sets of strings example, the following example illustrates the calculation of a distance from the default kernel for basic terms.

Example 2 (Default Distance on Sets of Strings). Let $s = \{A, B, C\}$, $t = \{A, D\}$, and $u = \{B, C\}$ where $s, t, u \in \mathfrak{B}_{S \to \Omega}$. We have $k(s,s) = 3$, $k(t,t) = 2$ and $k(u,u) = 2$. Then, $d_k(s,t) = \sqrt{3 - 3 + 2} = 1.73$, $d_k(s,u) = \sqrt{3 - 4 + 2} = 1$, and $d_k(t,u) = \sqrt{2 - 0 + 2} = 2$.

However, one of the strengths of the default kernel is that it allows any other valid kernel to be associated with a specific type. For example, the following *p*-spectrum kernel, defined on strings, is used in our experiments later in the paper.

Definition 6 (*p*-Spectrum Kernel [6]). *The feature space F associated with the* p*-spectrum kernel is indexed by $I = \Sigma^p$, with the explicit embedding from the space of all finite sequences over and alphabet Σ to a vector space F and is given by $\phi^p_u(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, u \in \Sigma^p$. The associated kernel is defined as $\kappa_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi^p_u(s)\phi^p_u(t)$.*

4 Relational Joins for Structured Data

We now upgrade both exact and approximate relational joins for structured data. The way we achieve this is to first upgrade the knowledge representation of the relation to be a set of basic terms rather than the traditional set of tuples. We then upgrade the relation index so that it indexes parts of a basic term rather than the traditional parts of a tuple. Once these two steps are completed, upgrading the exact relational join follows almost automatically with only modest changes to the definitions of the θ-restriction and joins. The final step then brings together the default kernel for basic terms and the approximate join to arrive at the main result of an approximate relational join for structured data. So to begin, we first upgrade the relation from section 2.1 to become the *basic term relation*, which is a basic term of type $\alpha \rightarrow \Omega$.

Definition 7 (Basic Term Relation). *A* basic term relation $R \subseteq \mathfrak{B}_\alpha$ *is a finite set of basic terms of the same type.*

In order to upgrade the relation index from section 2.1 so that it is applicable to the basic term relation, a suitable method of indexing sub-parts of a basic term is required. Recall that well-formed basic terms can consist of basic structures (e.g. lists, trees), basic abstractions (e.g. sets, multisets), basic tuples or arbitrary combinations of these three.

4.1 Indexing Basic Terms

In the logic, sub-parts of a term are referred to as subterms and so we are concerned with indexing the subterms of a basic term. The standard method for indexing subterms in the logic enumerates a decomposition of a given term such that every subterm is labelled with a unique string [3].

However, we introduce an alternative approach to indexing that, instead of enumerating all subterms of a term, defines a *type tree index set* over all subtypes of the type of a basic term. To do this we first adopt the definition of a type tree from [7] and then define a different annotation of the tree such that every member of the type tree index set identifies a set of terms rather than a single term. This ensures any index defined on a type is meaningful across all terms of that type. Furthermore, the set of subterms identified is guaranteed to consist entirely of well-formed basic terms.

To achieve this we follow the same interpretation of subtypes as [3] and restrict our attention to basic terms whose basic structures are in canonical form as defined below.

Definition 8 (Basic Structures in Canonical Form). *A type $\tau = T\ \alpha_1 \ldots \alpha_k$ is a basic structure in canonical form when, for all data constructors $C_i : \tau_{i1} \rightarrow \cdots \rightarrow \tau_{in} \rightarrow \tau$ that are associated with T, all the types of τ_{ij} are subtypes of τ.*

We begin our definition of the type tree index set with some preparatory notation. Let $\mathbb{Z}^+$ denote the set of positive integers and $(\mathbb{Z}^+)^*$ the set of all strings over the alphabet of positive integers, with ε denoting the empty string. io denotes the string concatenation of i with o where $i \in \mathbb{Z}^+$ and $o \in (\mathbb{Z}^+)^*$.

Definition 9 (Type Tree Index Set). *The* type tree index set *of a canonical type τ, denoted $\mathcal{O}(\tau)$, is the set of strings in $(\mathbb{Z}^+)^*$ defined inductively on the structure of τ.*

1. *If τ is an atomic type, then $\mathcal{O}(\tau) = \{\varepsilon\}$.*
2. *If τ is a basic structure type $\tau = T\ \alpha_1 \ldots \alpha_n$ in canonical form, with data constructors $C_i : \tau_{i_1} \to \cdots \to \tau_{i_m} \to \tau$ for all $i \in \{1, \ldots, l\}$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \bigcup_{v=1}^{p} \{ v o_v \mid o_v \in \mathcal{O}(\xi_v)\}$, where $\xi_1, \ldots, \xi_p$ are the types from α_k where $\alpha_k = \tau_{i_j}$ and $\tau_{i_j} \neq \tau$, and assuming that for every $\tau_{i_j} \neq \tau$ there exists an α_k such that $\alpha_k = \tau_{i_j}$.*
3. *If τ is a basic abstraction type $\beta \to \gamma$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \{ 1o \mid o \in \mathcal{O}(\beta)\} \cup \{ 2o \mid o \in \mathcal{O}(\gamma)\}$.*
4. *If τ is a basic tuple type $\tau = \tau_1 \times \cdots \times \tau_n$, then $\mathcal{O}(\tau) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{ i o_i \mid o_i \in \mathcal{O}(\tau_i)\}$.*

Part 1, the base case, states that types for which all the associated data structures have arity zero, such as Ω (the type of the booleans), *Int* (the type of the integers), and *Char* (the type of characters), have a singleton index set containing the empty string. Part 2 states that each subtype that occurs in the signatures of the associated data constructors, and that is not itself of type τ of the basic structure, is labelled with a unique string. Part 3 labels the β and γ types of basic abstractions with a pair of unique strings. Similarly, part 4 labels each tuple item in a basic tuple with a unique string.

The significance of defining indexing on the type tree of basic terms rather than on the terms themselves is that each member of a type tree index set $o \in \mathcal{O}(\tau)$ is not uniquely tied to any individual term of type τ. This increases the generality of the indexing such that each member of the type tree index set for type τ identifies, for any basic term $t : \tau$, an equivalence class of subterms rather than a single term. Thus $\mathcal{O}(t)$ induces a set of equivalence classes on the subterms of t. We refer to the set of subterms identified with a given member (index) of the type tree index set as the *basic subterm set* at that index.

Definition 10 (Basic Subterm Set). *If t is a basic term of type τ and $o \in \mathcal{O}(\tau)$ then the* basic subterm set of t at type tree index *o, denoted $t|_o$, is defined inductively on the length of o as follows.*

1. *If $o = \varepsilon$, then $t|_o = \{t\}$.*
2. *If $o = jo'$, for some o', and t has the form $C\ t_1 \ldots t_m$, with associated type $T\ \alpha_1 \ldots \alpha_n$, then $t|_o = s_j|_{o'}$ where $s_j = t_i : \tau_i$ such that $\tau_i \neq \tau$ and $\tau_i = \alpha_j$.*
3. *If $o = 1o'$, for some o', and t has the form* if_then_else(u, v, s)*, then $t|_o = u|_{o'} \cup s|_o$.*
4. *If $o = 2o'$, for some o', and t has the form* if_then_else(u, v, s)*, then $t|_o = v|_{o'} \cup s|_o$.*
5. *If $o = io'$, for some o', and t has the form $(t_1, \ldots, t_n)$, then $t|_o = t_i|_{o'}$, for $i = 1, \ldots, n$.*

A basic subterm set *is a set of basic subterms of a basic term at some type tree index. A basic subterm is* proper *if it is not at type tree index ε.*

Basic subterms indexed in part 1, the base case, are singleton sets containing an atomic term. Basic subterms indexed in part 2 are basic structures. Basic subterms indexed in parts 3 and 4 are the support and value of basic abstractions, i.e. respective instances of α and β, from $\alpha \to \beta$. Basic subterms indexed in part 5 are basic tuples.

Below we give examples of a type tree index set and basic subterm sets for each of basic tuples, basic structures, and basic abstractions. Starting with basic tuples in

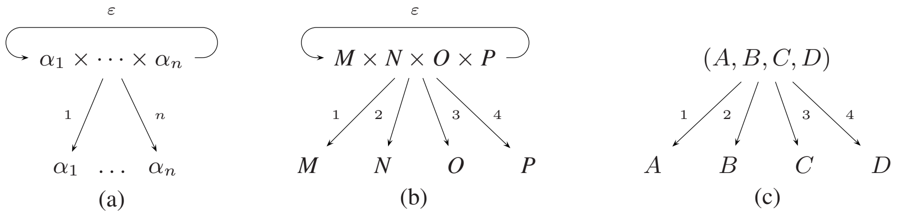


Fig. 1. Type-based indexing for basic tuples. (a) Type tree index for n-tuples of type $\alpha_1 \times \cdots \times \alpha_n$. (b) Type tree index for 4-tuples of type $M \times N \times O \times P$. (c) Basic subterm tree for term (A, B, C, D) where $A : M,\ B : N,\ C : O,\ D : P$.

Example 3 where it can be seen that type-based indexing identifies all the tuple items, but as singleton sets, and in addition it identifies the reflexive term at $t|_\varepsilon$.

Example 3. If basic tuple $t \in \mathfrak{B}_{M\times N\times O\times P}$ is the term $t = (A, B, C, D)$, where $A : M$, $B : N$, $C : O$, $D : P$, then the type tree index set of t is $\mathcal{O}(t) = \{\varepsilon, 1, 2, 3, 4\}$, the derivation of which can be seen from Fig. 1. The basic subterm sets of t are $t|_\varepsilon = \{(A, B, C, D)\}$, $t|_1 = \{A\}$, $t|_2 = \{B\}$, $t|_3 = \{C\}$ and $t|_4 = \{D\}$.

Representing basic structures, the usual right branching representation of lists is given in Example 4, where the basic subterm set at $t|_1$ captures one meaning of a list as a set of values and $t|_\varepsilon$ captures the meaning of a list as a set of sequences.

Example 4. If τ is a type of lists such that $\tau = \mathit{List}\ M$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor, with associated data constructors $\#$ and $[]$, having signatures $[] : \mathit{List}\ M$, and $\# : M \to \mathit{List}\ M \to \mathit{List}\ M$, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1\}$. If basic terms $s, t \in \mathfrak{B}_{\mathit{List}\ M}$ are the lists $s = [A, B, C]$ and $t = [A, D]$, then as can be seen from Fig. 2, the basic subterm sets of s and t are $s|_\varepsilon = \{[A, B, C], [B, C], [C], []\}$, $s|_1 = \{A, B, C\}$, and $t|_\varepsilon = \{[A, D], [D], []\}$, $t|_1 = \{A, D\}$.

For basic abstractions, a set is given in Example 5 and a multiset in Example 6. For both sets and multisets, $t|_1$ captures the meaning as a set of values whereas $t|_2$ will always be $\{\top\}$ for sets and a set of multiplicities for multisets. A corollary of Definition 9 is that the type tree index set of a basic abstraction type is always $\{\varepsilon, 1, 2\}$.

Example 5. If τ is a basic abstraction type representing sets such that $\tau = M \to \Omega$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1, 2\}$. If basic term $t = \{A, B, C\}$, where $A, B, C : M$, then the basic subterm sets are $t|_\varepsilon = \{\{A, B, C\}\}$, $t|_1 = \{A, B, C\}$ and $t|_2 = \{\top\}$.

Example 6. If τ is a basic abstraction type representing multisets such that $\tau = M \to \mathit{Nat}$, where $M \subseteq \mathfrak{B}$ is a nullary type constructor and *Nat*, then the type tree index set of τ is $\mathcal{O}(\tau) = \{\varepsilon, 1, 2\}$. If basic term $t = \{A, A, A, B, C, C\}$, where $A, B, C : M$, then the basic subterm sets are $t|_\varepsilon = \{\{A, A, A, B, C, C\}\}$, $t|_1 = \{A, B, C\}$ and $t|_2 = \{1, 2, 3\}$.

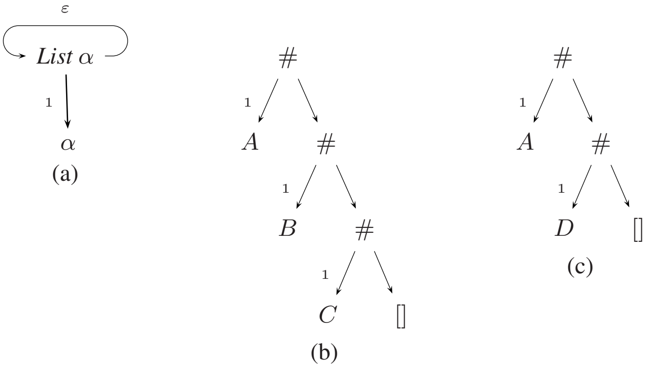


Fig. 2. Type-based indexing for basic structures. (a) Type tree index for *List* α. (b) and (c) Basic subterm trees for terms $[A, B, C]$ and $[A, D]$ of type *List M* where A, B, C, D : *M*.

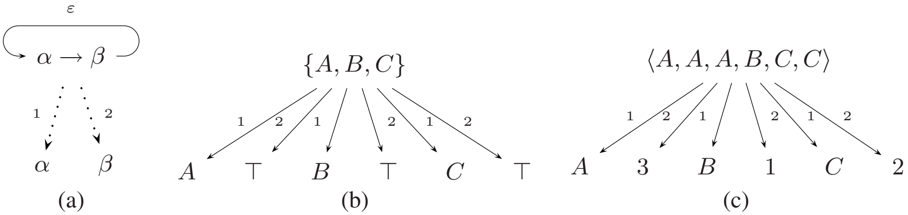


Fig. 3. Type-based indexing for basic abstractions. (a) Type tree index for type $\alpha \to \beta$. (b) Basic subterm tree for set $\{A, B, C\}$, type $M \to \Omega$, where A, B, C : *M* and $\top : \Omega$. (c) Basic subterm tree for multiset $\langle A, A, A, B, C, C \rangle$, type $M \to$ Nat, where A, B, C : *M* and $1, 2, 3$: Nat.

A useful and straight forward reformulation of type-based indexing is *type name-based indexing* that, instead of enumerating the edges of the type tree, directly labels the vertices of the type tree. The simplest approach being to assign a unique type name to every vertex in the type tree. If the names assigned have no understandable meaning to humans then this method offers no advantages over type-based indexing. However, if the knowledge representational formalism used to define types and data instances uses human-understandable names then type name-based indexing provides a useful notation for referring to basic subterm sets, as illustrated in Example 7.

Example 7. Let *Author* be the type of authors from the publications domain, which define declaratively in the Haskell style syntax from [4] as follows.

```
type Author = (Name,Publications);
type Name = String;
type Publications = List Publication;
type Publication = (Mode,Coauthors,Title,Venue,Year);
data Mode = Journal | Proceedings | ... | Book;
type Coauthors = Coauthor -> Bool;
type Coauthor = String;
type Title = String;
```

```
    type Venue = String;
    type Year = Int;
```

This states that *Author* is a pair of *Name* and *Publications*, where *Name* is an alias for *String* the type of strings, and *Publications* is a list of publications, which in turn is a 5-tuple of *Mode, Coauthors, ..., Year*, where *Mode* has the nullary data constructors *Journal*, *Proceedings*, ..., *Book*, and so on through to *Year* which is an alias for the type *Int*, the type of the integers. *Coauthors* is a basic abstraction from *Coauthor* to *Bool*, where *Bool* is the type Ω, i.e. *Coauthors* is a set of *coauthors*. To ensure the required uniqueness of type names *Coauthor*, *Title*, *Venue* and *Name* are aliases for the type *String*. The type tree index set is thus {*Author, Author.Name, Author.Publications.Publication.Mode, ..., Author.Publications.Publication.Year*}.

A type tree index set generated using this method is isomorphic with that produced by Definition 9, as illustrated informally in Fig. 4. The constraint that all basic subtypes must be uniquely named permits the following simpler definition of a basic subterm set.

Definition 11 (Basic Subterm Set (with named types)). *If t is a closed basic term of type τ and $\alpha \subseteq \tau$ then the* basic subterm set of t at type α*, denoted $t|_\alpha$, is $t|_\alpha = \{s \mid s$ occurs in t with type $\alpha\}$. A* basic subterm set *is a set of basic subterms of a basic term at some type tree $\alpha \subseteq \mathfrak{B}$. A basic subterm is* proper *if $\alpha \neq \tau$.*

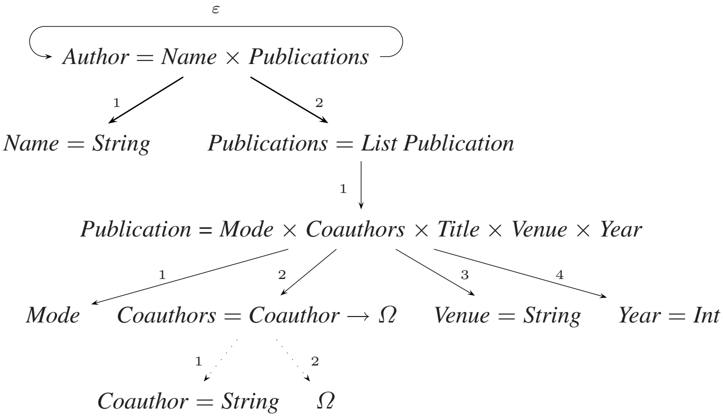

ε
Author = Name × Publications
1
2
Name = String
Publications = List Publication
1
Publication = Mode × Coauthors × Title × Venue × Year
1
2
3
4
Mode
Coauthors = Coauthor → Ω
Venue = String
Year = Int
1
2
Coauthor = String
Ω

Fig. 4. Type name-based and type-based indexing for type *Author*

4.2 Indexing Basic Term Relations

Having upgraded our representation of a relation $R : \tau$ to handle structured data represented as basic terms, and having chosen a suitable indexing method for the basic subterm set $\mathcal{O}(\tau)$, we are now able to conveniently define the *basic term relation index* as the structured data counterpart of the relation index.

Definition 12 (Basic Term Relation Index). *The* basic term relation index I_R *of a basic term relation R of type τ is $I_R = \mathcal{O}(\tau)$.*

4.3 Exact Relational Join for Structured Data

The upgraded definitions of an exact relation join for structured data closely follow the earlier relational definitions but now using basic term relations and indexing.

Definition 13 (Basic Term Projection). *If* $t \in \tau$*, where* $\tau \subseteq \mathfrak{B}$*, then the* basic term projection π *of* t *on* $i \in I_t$ *is* $\pi_i(t) = \{s \mid s \textit{ is the basic subterm at type tree index } i\}$.

A basic term projection $\pi_i(t)$ may also be written as $t|_i$.

Definition 14 (Basic Term Generalised Projection). *If* $t \in \mathfrak{B}$ *then the* basic term generalised projection π *of* t *on* $\rho \subseteq I_t$ *is the set* $\pi_\rho(t) = \{\, t|_i \mid i \in \rho \,\}$.

Definition 15 (Basic Term θ-Restriction). *Let* θ *be a predicate* $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ *for some* $\alpha \in \mathfrak{B}_\alpha$*. If* A *and* B *are basic term relations with basic terms* $a|_i \subseteq \mathfrak{B}_\alpha$ *and* $b|_j \subseteq \mathfrak{B}_\alpha$ *respectively for some* $(i, j) \in I_A \times I_B$*, then* basic term θ-restriction $\sigma_{i\theta j}$ *is defined on* $T \subseteq A \times B$ *as* $\sigma_{i\theta j}(T) = \{(a, b) \mid a|_i \,\theta\, b|_j \,\wedge\, (a, b) \in T\}$.

The predicate $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ is defined on sets of basic terms. In other words, θ is a predicate on basic term relations.

Definition 16 (Basic Term Generalised Restriction). *Let* φ *be a proposition that consists of atoms as allowed in basic term* θ*-restriction and the logical operators* $\wedge$, $\vee$ *and* $\neg$*. If* A *and* B *are basic term relations then the* basic term generalised restriction σ_φ *is defined on* $T \subseteq A \times B$ *as* $\sigma_\varphi(T) = \{\, t \mid \varphi(t) \,\wedge\, t \in T\}$.

Definition 17 (Basic Term θ-Join). *Let* $\theta : (\mathfrak{B}_\alpha \to \Omega) \to (\mathfrak{B}_\alpha \to \Omega) \to \Omega$ *be a predicate for some type* $\alpha \in \mathfrak{B}$*. If* A *and* B *are basic term relations with basic terms* $a|_i \subseteq \mathfrak{B}_\alpha$ *and* $b|_j \subseteq \mathfrak{B}_\alpha$ *respectively for some* $(i, j) \in I_A \times I_B$ *then the* basic term θ-join $\bowtie_{i\,\theta j}$ *of* A *and* B *is defined as* $A \bowtie_{i\,\theta j} B = \sigma_{i\,\theta j}(A \times B)$.

Definition 18 (Basic Term Generalised Join). *Let* φ *be a proposition that consists of atoms as allowed in basic term* θ*-restriction and the logical operators* $\wedge$, $\vee$ *and* $\neg$*. If* A *and* B *are basic term relations then the* basic term join $A \bowtie_\varphi B = \sigma_\varphi(A \times B)$.

4.4 Approximate Relational Join for Structured Data

We assume some distance for basic terms and note that positive semi-definite kernels induce pseudo-metric distances. One suitable kernel is the kernel for basic terms from [4] and described earlier, but other kernels and distances may also be suitable.

Definition 19 (Basic Term Proximity-Join). *Let* $\approx : \mathfrak{B}_\alpha \times \mathfrak{B}_\alpha \to \Omega$ *be a proximity for some* $\mathfrak{B}_\alpha$ *of type* α*. If* A *and* B *are basic term relations with subterms* $a|_i \in \mathfrak{B}_\alpha$ *and* $b|_j \in \mathfrak{B}_\alpha$ *respectively for some* $(i, j) \in I_A \times I_B$*, then the* proximity-join $\widetilde{\bowtie}_{i\approx j}$ *of* A *and* B *is defined as* $A \,\widetilde{\bowtie}_{i\approx j}\, B = \sigma_{i\approx j}(A \times B)$.

This definition closely parallels that of the approximate relational join on account of the following: the basic term relation is a set which allows the same set theoretic operators from the relational case to apply; the basic term relation index fulfills the same role as the relation index from the relational case; and, finally, the kernel for basic terms' own inductive definition implicitly handles the often recursive nature of structured data. The closeness in form of the definition of the basic term join to that of the relational join facilitates the following result.

Proposition 1. *Relational joins are a special case of basic term relational joins.*

Proof. Assume relation $R \subseteq D_1 \times \cdots \times D_n$ for some domains $D_1, \ldots, D_n$. Assume appropriate type constructors and data constructors such that $D_1, \ldots, D_n \subseteq \mathfrak{B}$. Let basic term relation $S \subseteq D_1 \times \cdots \times D_n$. Let I_R be the relation index of R and I_S be the basic term relation index of S. Clearly there is a surjection from I_R into I_S and thus from the set of tuple items in each tuple in R to the set of subterms in each corresponding basic term tuple in S. Assume the θ operators are available for basic terms and the result follows. □

5 Applications

We have implemented the higher-order relational projection, restriction and join operators and a range of supporting kernels, including the kernel for basic terms, in Prolog. Although Prolog does not natively support the higher-order logic necessary to represent data as basic terms, emulation of typed data, basic tuples, basic structures and basic abstractions (including sets and multisets) has proven to be unproblematic in practice. We are currently working to characterise and evaluate this framework using the application domain of bibligraphic publications. Heterogenous data sets within this domain include CORA, DBLP, Citeseer and Google Scholar. Interesting higher-order approximate joins between pairs (A, B) of these datasets might, for instance, include the following.

- $A \tilde{\bowtie}_{\textit{Author.name}} B$, authors in A and B that have similarly names
- $A \tilde{\bowtie}_{\textit{Author.affiliation}} B$, authors in A and B affiliated to the same institution
- $A \tilde{\bowtie}_{\textit{Author}} B$, authors in A and B similar across all their properties
- $A \tilde{\bowtie}_{\textit{Publication.venue}} B$, publications in A and B from the same venue
- $A \tilde{\bowtie}_{\textit{Publication.coauthors}} B$, publications in A and B with similar coauthors

For the sake of evaluation we require the ground truth V for each join to be evaluated, where $V \subseteq A \tilde{\bowtie} B$ and, for the case where the individuals as terms represent publications, $V = \{(a, b) \mid a \in A \land b \in B \land \textit{a and b are variants of the same publication}\}$. The goal is to reconstruct V as $V' = A \tilde{\bowtie}_s B$ by choosing an appropriate s from the intersection of the basic subterm sets of A and B. In reality, V is not usually available for pairs of different data sets. For this reason we narrow down our initial evaluation to consider self-joins, $A \tilde{\bowtie}_s A$, on a single data set $A = \textit{CORA}$, for which ground truths are available [8]. *CORA* consists of bibliographic citations, hand-labelled with unique identifiers so that variant citations of the same paper share the same identifier[1].

[1] The specific CORA data set used is an aggregation of all three CORA-REFS citation matching data sets (*fahl-labeled*, *kibl-labeled*, and *utgo-labeled*). The raw CORA-REF files have numerous XML mark-up errors which we have manually corrected to enabled parsing.

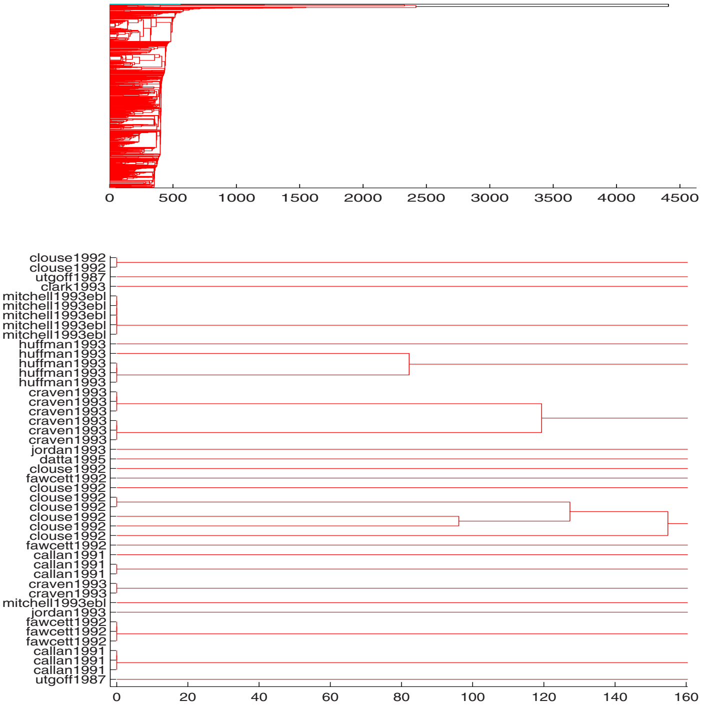


Fig. 5. Dendrogram (above), showing clusterings at successive thresholds for a proximity-joins on the CORA publication type, and (below) a close-up showing labelled ground truths

Given this supervised learning setting, a number of distance-based methods could be used to implement the approximate join, including k-means, k-NN, and agglomerative hierarchical clustering. We chose the latter for this initial investigation on the basis that it produces a dendrogram that is useful in visualising and charactering the join. Although this is a clustering method more normally associated with unsupervised learning, here we are able to make use of the ground truth labelling to achieve a supervised setting. A dendrogram represents a progressive series of joins (or clusterings), with instances in the same cluster being leaves of the same sub-tree. The distance value at each non-terminal node represents a potential threshold δ at which to 'cut' the tree and arrive at set of clusters. δ is the threshold from the proximity predicate from Definition 19. To evaluate the quality of the clustering at a given δ we consider whether each pair of instance data is correctly classified as being in the same cluster or in different clusters; in other words we evaluate a binary classification on all pairs of instances to determine if the two instances in a pair refer to the same publication or to different publications. A confusion matrix is then calculated in order to determine precision and recall for this specific value of δ. To characterise a proximity-join across a range of thresholds we vary δ across the length of the tree and plot a precision-recall chart.

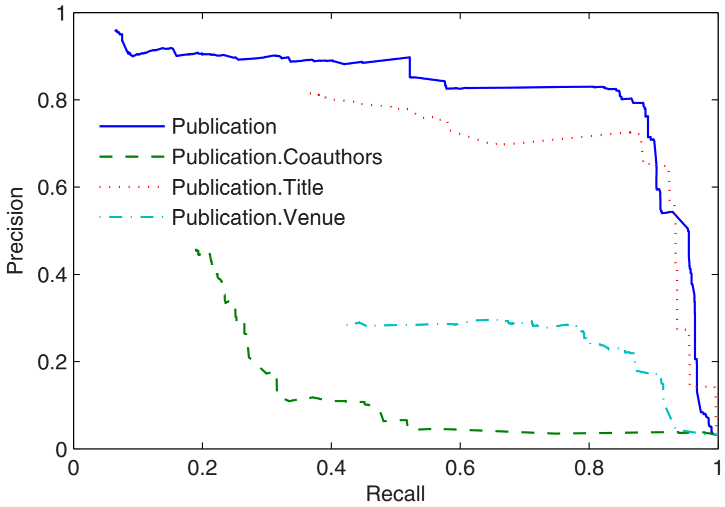


Fig. 6. Precision and recall for various decompositions of CORA publication type

To represent the publications we choose the following type structure[2].

```
type Publications = Publication -> Bool;
type Publication = (Coauthors,Title,Venue,Year);
type Coauthors = Coauthor -> Bool;
type Coauthor = String;
type Title = String;
type Venue = String;
type Year = String;
```

Hence by representing CORA as a basic terms relation of type *Publications*, where $\mathfrak{B}_{Publications} \in \mathfrak{B}$, we are able to execute the following basic term proximity-joins:

- $CORA \ \widetilde{\bowtie}_{Publication.Title} \ CORA$, a self join on only the publication's *Title* subterm;
- $CORA \ \widetilde{\bowtie}_{Publication.Venue} \ CORA$, a self join on only the publication's *Venue* subterm;
- $CORA \ \widetilde{\bowtie}_{Publication.Coauthors} \ CORA$, a self join on only the publication's *Coauthors* subterm, which is in turn a set of *Coauthor* subterms;
- $CORA \ \widetilde{\bowtie}_{Publication} \ CORA$, a self join on the entire *Publication* term.

For each join, to keep results comparable, we choose the p-spectrum(2) kernel for strings and accept the default kernels for all other types. We do not optimise the default kernel for basic terms by choosing weighting modifiers that, for example, might be used to encode the intuition that a year of publication is less discriminating than the title of a publication when aggregated into an overall kernel on publications.

For each of these four joins we constructed a dendrogram such as Fig.5 and calculated the corresponding precision-recall chart in Fig.6. Note that the trivial reflexive

[2] *Year* is string rather than a numeric type due to non-numeric characters in the data. Also, *Venue* is constructed as a concatenation of venue-related fields; CORA has no venue field.

pairs, i.e. cluster sizes of 1, are ignored in the plots as they convey no useful information here and so lines are not interpolated to the top left of the chart (recall=0, precision=1). As would intuitively be expected, joins on *Publication.Title* is generally a better discriminator of publications than *Publication.Coauthors* and *Publication.Venue*. However, the default kernel for basic terms clearly effectively aggregates the information contained the subterms of *Publication* to outperform any single one of the three subterms taken in isolation. The only exception being *Publication.Title*, which sometimes outperforms its parent *Publication* above recall values greater than 0.9.

6 Related Work

Our work, to the best of our knowledge, is a unique combination of the relational model with a higher-order representation and distance-based methods. Thus we now describe our work with respect to related work in three related fields: relational learning, knowledge representation, and distances for structured data.

In database literature and more recently, particularly since the advent of the Web, within the relational learning literature, there has been considerable interest in the data integration aspects of database deduplication and record linkage [9,10]. However, in addition to dealing with heterogeneous data structures, our work adopts an *individuals as terms* representation so that both type and structure of data is not obfuscated by traditional relational representations. Therefore our approach has the advantage of simplifying data modelling and the application of approximate matching techniques. Despite this, it should be noted that we propose the higher-order relational representation solely as an approach for data integration tasks, not as a replacement for general purpose relational databases. Our present implementation certainly has none of the optimisations of a modern relational database. Ultimately though, a higher-order view could be layered on top of a traditional relational database system, efficiently combining the two approaches, so that higher-order queries are automatically translated into and executed as equivalent relational queries.

Our goal of integrating and querying heterogeneous data is also a goal shared by the Semantic Web community [11]. The fundamental data model of the Semantic Web is the directed labelled graph, represented as RDF triples, which may be queried using the SPARQL query language [12]. Data structures such as lists, sets, multisets, trees and graphs are readily supported through RDF Schema and the OWL ontology language [13] and as such have similar representational advantages to basic terms as compared to the relational model. SPARQL queries can be used to retrieve a subgraph describing an individual that is analogous to a representation of that individual as a basic term. Conversely, it is straight forward to transform the same subgraph into a basic term in order to apply our own approach to RDF data. For RDF data integration, or *smushing* as it is informally known, the emphasis in the Semantic Web languages to-date has been on exact matching, using inverse-functional properties such as email addresses, homepage URLs or entity URIs. This is an obvious shortcoming in the presence of noisy data or representational variations between data from different sources. To address the consequent data integration problem, work has been done in the area of ontology matching, including work on measuring proximity between ontologies [14]. Our approximate matching work differs from this explicit semantic integration approach in that we rely primarily on the implicit semantics of the type structure and data instances themselves. This is an advantage in cases where detailed ontological information is not available but potentially a disadvantage in other cases because background knowledge encoded in an ontology is not exploited in our approximate joins. The incorporation of background knowledge into our approximate joins is an area for future work.

Turning now to related work on distances, we first note that kernels and distances used in this paper are not of themselves a contribution of our work. Also, the choice of the default kernel for basic terms is not a specific requirement for the approximate relational join; any distance for basic terms would be suitable. Prior work on distances for logical terms includes distances between Herbrand interpretations [15] and between first-order terms (including structures and lists) [16,17,18]. None of these directly apply to basic terms and while it may be possible to apply distances on first-order terms to our first-order representation of basic terms, the semantics of basic abstractions would be lost as a result. Most closely related to our work, are various similarity-based methods that have been upgraded to handle structured data [4,19,20]. Contrasting approaches apply probabilistic models to take account of dependencies between resolution decisions [21,22]. Most recently, a family of pseudo-distances over the set of objects in a knowledge base has been introduced although not specifically for basic terms [23].

7 Conclusion

In this paper we have combined two contrasting knowledge representational approaches, the relational model and basic terms, into a single coherent formalism that is well suited to the integration of heterogeneous data. This, in conjuction with the default kernel for basic terms has been shown to have potential for data integration and to be worthy of further investigation.

References

1. Codd, E.F.: The Relational Model for Database Management, Version 2. Addison-Wesley, Reading (1990)
2. Date, C.J.: An Introduction to Database Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (1991)
3. Lloyd, J.W.: Logic and Learning. Springer, New York (2003)
4. Gaertner, T., Lloyd, J.W., Flach, P.A.: Kernels and distances for structured data. Mach. Learn. 57(3), 205–232 (2004)
5. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5(2), 56–68 (1940)
6. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
7. Gyftodimos, E., Flach, P.A.: Combining bayesian networks with higher-order data representations. In: Famili, A.F., Kok, J.N., Peña, J.M., Siebes, A., Feelders, A. (eds.) IDA 2005. LNCS, vol. 3646, pp. 145–156. Springer, Heidelberg (2005)
8. Culotta, A., McCallum, A.: Joint deduplication of multiple record types in relational data. In: CIKM 2005: Proceedings of the 14th ACM international conference on Information and knowledge management, pp. 257–258. ACM, New York (2005)

9. Lawrence, S., Bollacker, K., Giles, C.L.: Autonomous citation matching. In: Proceedings of the 3rd International Conference on Autonomous Agents, pp. 392–393. ACM Press, New York (May 1999)
10. Newman, M.E.J.: The structure of scientific collaboration networks. Proc. Natl. Acad. Sci. USA 98, 404–409 (2001)
11. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
12. Prud'hommeaux, E., Seabourne, A.: SPARQL Query Language for RDF. W3C, W3C Working Draft April 19, 2005 edn. (April 2005)
13. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview (2004)
14. Maedche, A., Staab, S.: Measuring similarity between ontologies. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 251–263. Springer, Heidelberg (2002)
15. Nienhuys-Cheng, S.H.: Distance between herbrand interpretations: A measure for approximations to a target concept. In: [24], pp. 213–226
16. Sebag, M.: Distance induction in first order logic. In: [24], pp. 264–272
17. Bohnebeck, U., Horváth, T., Wrobel, S.: Term comparisons in first-order similarity measures. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 65–79. Springer, Heidelberg (1998)
18. Kirsten, M., Wrobel, S.: Extending k-means clustering to first-order representations. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 112–129. Springer, Heidelberg (2000)
19. Bhattacharya, I., Getoor, L.: Relational clustering for multi-type entity resolution. In: MRDM 2005: Proceedings of the 4th international workshop on Multi-relational mining, pp. 3–12. ACM Press, New York (2005)
20. Woznica, A., Kalousis, A., Kalousis, M.H.A., Hilario, M.: Kernels over relational algebra structures. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 588–598. Springer, Heidelberg (2005)
21. Domingos, P., Domingos, P.: Multi-relational record linkage. In: Dzeroski, S., Blockeel, H. (eds.) Proceedings of the 2004 ACM SIGKDD Workshop on Multi-Relational Data Mining, pp. 31–48 (August 2004)
22. Bhattacharya, I., Getoor, L.: A latent Dirichlet model for unsupervised entity resolution. In: 6th SIAM Conference on Data Mining (SDM 2006), Bethesda, MD (2006)
23. d'Amato, C., Fanizzi, N., Esposito, F.: Induction of optimal semantic semi-distances for clausal knowledge bases. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 29–38. Springer, Heidelberg (2008)
24. Lavrac, N., Dzeroski, S.(eds.): ILP 1997. LNCS, vol. 1297. Springer, Heidelberg (1997)

A Comparison between Two Statistical Relational Models

Lorenza Saitta[1] and Christel Vrain[2]

[1] Dip. di Informatica, Università del Piemonte Orientale
Via Bellini 25/G, 15100 Alessandria, Italy
saitta@mfn.unipmn.it
[2] LIFO, University of Orléans , BP 6759, 45067 Orléans cedex 02, France
christel.vrain@univ-orleans.fr

Abstract. Statistical Relational Learning has received much attention this last decade. In the ILP community, several models have emerged for modelling and learning uncertain knowledge, expressed in subset of first order logics. Nevertheless, no deep comparisons have been made among them and, given an application, determining which model must be chosen is difficult. In this paper, we compare two of them, namely Markov Logic Networks and Bayesian Programs, especially with respect to their representation ability and inference methods. The comparison shows that the two models are substantially different, from the point of view of the user, so that choosing one really means choosing a different philosophy to look at the problem. In order to make the comparison more concrete, we have used a running example, which shows most of the interesting points of the approaches, yet remaining exactly tractable.

1 Introduction

In the last years there has been an increasing interest in graphical models for reasoning and learning. In particular, several approaches aiming at representing structured domains, described in some subset of First Order Logic, have been proposed, such as, for instance, Markov Logic Networks [RD06], Bayesian Logic Programs [KdR07], Probabilistic Relational Models [FGKP99], Relational Bayesian Networks [Jae97, Jae02]. All these approaches rely on the possible-world semantics of first-order probabilistic logic [Hal96], which assigns a probability distribution over possible worlds[1]. Stochastic Logic Programs [Mug00] are slightly different in the sense that they extend stochastic grammars and allow a distribution to be computed on the Herbrand base; in [Cus01], they have been proven useful for representing undirected Bayes nets.

In this paper we review two statistical relational models, namely Markov Logic Networks and Bayesian Logic Programs, and we attempt a quantitative comparison between them. The reader is assumed to be familiar with both syntax and semantics of First Order Logics, especially with clausal form representation.

Throughout this paper we will use a running example to perform the comparison. Even though it is a very simple one, it shows most of the interesting features of a

[1] Let us notice that Probabilistic Relational Models have been defined both on possible-world semantics and on domain frequency semantics [Get06].

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 244–260, 2008.

stochastic predicate logic knowledge base, yet remaining tractable, so that exact inferences can be done.

Example 1. Let $\mathcal{K}$ be a knowledge base containing four rules:

$$F_1: \ R(X,Y):-P(X),Q(Y) \qquad F_2: \ S(X,Y):-P(X),Q(Y)$$
$$F_3: \ V(X,Y):-Q(X),Q(Y) \qquad F_4: \ R(X,Y):-V(X,Y)$$

In $\mathcal{K}$ the two rules F_1 and F_2 share the antecedent but have different consequents, whereas F_1 and F_4 share the consequent, but have different antecedents. Moreover, if $\mathbf{C} = \{a,b\}$ is the set of constants in the universe, the Herbrand base $HB_{\mathcal{K}}$, relative to the knowledge base $\mathcal{K}$, contains the following ground atoms:

$$HB_{\mathcal{K}} = \{P(a),P(b),Q(a),Q(b),R(a,a),R(a,b),R(b,a),R(b,b),S(a,a),S(a,b),$$
$$S(b,a),S(b,b),V(a,a),V(a,b),V(b,a),V(b,b)\}$$

The paper is organized as follows. Section 2 is devoted to Markov Logic Networks, whereas we present Bayesian Logic Programs in Section 3. In Section 4 we compare the two frameworks, and in Section 5 some conclusions are drawn.

2 First Order Logic Background

Classically, a *first order language* $\mathcal{L}$ is defined by a countable set $\mathbf{V}$ of variables, a set $\mathbf{F}$ of function symbols (including functions with arity 0, also called constants) and a set $\mathbf{P}$ of predicate symbols.

In this paper, we consider only *function-free first-order languages* that contain no function symbols other that constants, and in the following, $\mathbf{C}$ denotes the set of constants. With this restriction, a *term* is either a variable or a constant. A *ground term* is a constant. The *Herbrand Universe* of $\mathcal{L}$, denoted by $HU_{\mathcal{L}}$, is the set $\mathbf{C}$ of constants. A substitution σ is a *variable assignment* that maps each variable X to a variable or to a constant. An *atom* is an expression $p(t_1,\dots,t_n)$, where p is a predicate symbol and t_1, ..., t_n are either constants or variables. A *ground atom* is an atom that contains no variables. The *Herbrand Base* of $\mathcal{L}$, denoted by $HB_{\mathcal{L}}$, is the set of ground atoms.

A *(positive) rule* r is an expression $A \leftarrow B_1,\dots,B_k$, where B_1, $\dots B_k$ are atoms. The atom A is called the *head* of r and is denoted by $head(r)$. The right-hand side $B_1,\dots,B_k$ is called the *body* of r and is denoted by $body(r)$.

A *(positive) program* $\mathcal{P}$ is a set of positive rules. In the following, if $\mathcal{P}$ is a program, $Inst(\mathcal{P})$ is the set of all ground rules obtained by substituting variables in $\mathcal{P}$ by ground terms ($c \in Inst(\mathcal{P})$ when c is ground and there exists a clause c' of $\mathcal{P}$ and a ground substitution σ of variables of c' s.t. $c'\sigma = c$)

In the following, $HB_{\mathcal{P}}$ denote the Herbrand base of the first-order language, underlying the definition of program $\mathcal{P}$.

An *interpretation* I is defined by giving a domain D, $D \neq \emptyset$, and by associating to each constant c, $c \in \mathbf{C}$, an element of D and to each predicate symbol p, $p \in \mathbf{P}$, with arity n, a function $I_p : D^n \rightarrow \{0,1\}$. A *Herbrand interpretation I* is an interpretation on the Herbrand Universe that interprets each constant by itself.

The semantics of a positive logic program can be defined as in the following:

Definition 1. *Let $\mathcal{P}$ be a positive logic program. The semantics $\mathcal{M}_{\mathcal{P}}$ of $\mathcal{P}$ is defined as the least fixpoint of the operator $T_{\mathcal{P}}$ defined for a set a ground atoms I by:*

$$T_{\mathcal{P}}(I) = \{A \in HB_{\mathcal{P}} | \exists C \in Inst(\mathcal{P}),\ head(C) = A \text{ and } body(C) \subseteq I\}.$$

$\mathcal{M}_{\mathcal{P}}$ is the least Herbrand model of $\mathcal{P}$, *i.e.*, it is the set of ground atoms that are logical consequences of $\mathcal{P}$.

3 Markov Logic Networks

Domingos and co-workers have used the concepts of *Markov Networks* (MN) and of *Markov Logic Networks* (MLN) to address the problem of making probabilistic inferences in first order logic [RD06, KD05, DR04].

3.1 Markov Network Representation

A *Markov Network* (**MN**) is a model of the joint probability distribution over a set $\mathbf{X} = (X_1, ..., X_N)$ of variables. More precisely, a Markov network is an undirected graph G, with N nodes (each corresponding to one of the stochastic variables) and M edges, and a set of potential functions Φ_k $(1 \leqslant k \leqslant K)$. Each Φ_k is associated to a different *clique* in the graph. Let $\mathbf{x} = (x_1, ..., x_N) \in \mathcal{X}$ be a set of values that the variables can take on; then, a Markov network is associated to the following probability distribution:

$$Pr(\mathbf{X} = \mathbf{x}) = \frac{1}{Z}\, e^{\sum_{k=1}^{K} w_k f_k(\mathbf{x}^{(k)})} \tag{1}$$

In (1) each function $f_k(\mathbf{x}^{(k)})$ is a *feature* of the k-th clique, and w_k is its *weight*. Moreover, Z is a normalization factor. Even though a feature can be any function, Domingos's approach focuses on binary features, *i.e.*, $f_j(\mathbf{x}^{(\mathbf{k})}) \in \{0, 1\}$. Moreover, $w_k = ln\, \Phi_k(\mathbf{x}^{(k)})$.

A Markov Network is derived from a *Markov Logic Network* (**MLN**) and a set $\mathbf{C} = \{a_1, ..., a_C\}$ of constants. A Markov Logic Network $\mathbf{MLN} = \{(F_i, w_i) | 1 \leqslant i \leqslant M\}$ is a set of pairs (F_i, w_i), consisting of a logical formula F_i, belonging to a knowledge base $\mathcal{K}$, and an associated real number w_i. The logical language is function-free.

The association between $\mathbf{MLN}, \mathbf{C}$ and a Markov network **MN** is performed as follows: let $HB_{\mathcal{K}}$ be the Herbrand base of $\mathcal{K}$. Each element of $HB_{\mathcal{K}}$ is associated to a node in the graph **MN**, which then represents a binary variable. Now we can ground all formulas F_i $(1 \leqslant i \leqslant M)$ in all possible ways. In **MN** two nodes are connected if they appear together in at least one of the groundings of a formula F_i, for some i.

Example 2. By considering the knowledge base $\mathcal{K}$ of Example 1, we can define an **MLN** as follows:

$$\mathbf{MLN} = \{(F_1, w_1), (F_2, w_2), (F_3, w_3), (F_4, w_4)\},$$

where the F_i's are the rules in $\mathcal{K}$ and $\mathbf{w} = (w_1, w_2, w_3, w_4)$ is a vector of weights.

Using the **MLN** of Example 1 and its Herbrand base $HB_{\mathcal{K}}$, we obtain the Markov network **MN** reported in Figure 1. The network **MN** has $N = 16$ nodes and $M = 31$ edges.

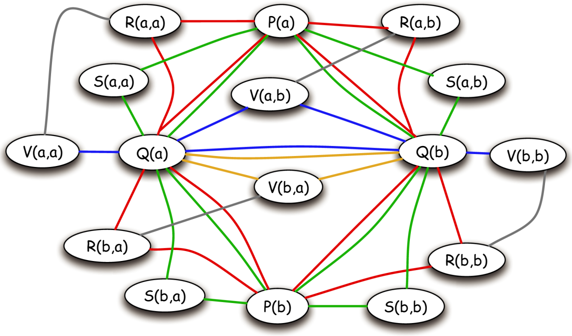


Fig. 1. Markov network corresponding to Example 3

Clearly, groundings of the same formula constitute cliques in **MN**. Each grounding of each formula is a feature f_j. To each grounding of the same formula F_i the same weight w_i, specified in **MLN**, is attached. We may notice, however, that in **MN** there may appear spurious cliques, *i.e.*, cliques that do not correspond to a grounding of any formula. In the Markov network **MN** of Figure 1, there are 31 cliques of length 2 (equal to the number of edges in the graph), among which only 6 are maximal, and 16 cliques of length 3 (all maximal ones). In fact, in this case, the length-2 and length-3 cliques derive from different formulas and have different weights associated to them. In particular, the length-2 cliques to be considered are the groundings of formula F_4, and some of formula F_3. Of the sixteen length-3 cliques, only ten are true groundings, and they are reported in Figure 2 (each one with its associated weight). A spurious clique is, for instance, (Q(b), R(a,b), V(a,b)).

Let us now consider the set $\mathcal{X}$ of possible worlds. Given a world $\mathbf{x} \in \mathcal{X}$, all variables associated to the nodes of **MN** have a value in $\{\mathit{true, false}\}$. Correspondingly, each feature f_j, corresponding to the grounding of a formula, has the value *1*, if the corresponding grounding is *true*, 0 otherwise. By reconsidering formula (3) and recalling that $w_k = \mathit{ln}\, \Phi_k(\mathbf{x}^{(k)})$, we obtain the following final form for the probability distribution:

$$Pr(\mathbf{X} = \mathbf{x}) = \frac{1}{Z}\, e^{\sum_{i=1}^{M} w_i\ n_i(\mathbf{x})} = \frac{1}{Z} \prod_{i=1}^{M} e^{w_i\ n_i(\mathbf{x})} \qquad (2)$$

In (2) $n_i(\mathbf{x})$ is the number of true groundings of formula F_i in $\mathbf{x}$, or, in other words, the number of features, corresponding to F_i, that are equal to 1 in $\mathbf{x}$. Formula (2) will be the one used in the computation of probabilities. However, it has to be noted that computing the partition function Z is intractable, because it requires a number of steps $O(2^{|\mathbf{MN}|})$. From (2) and Z, it is possible to compute the probability of any possible world $\mathbf{x}$.

For the sake of exemplification, let $\mathbf{w} = (w_1, w_2, w_3, w_4) = (2, 1.5, 0.8, 1.7)$ be the weight vector associated to $\mathcal{K}$. Using formula (3) with the weights introduced before, it is possible to compute, in this simple case, the exact value of $Z = 3.0225... \cdot 10^{14} \simeq 3.023 \cdot 10^{14}$.

From Figure 1 we see that the state space corresponding to the Markov network is the space $\{0, 1\}^{16}$, consisting of all the possible truth value assignments to the binary

F_i	Formula	Features	Weight w_i
F_1	R(X,Y) :- P(X), Q(Y)	(P(a), Q(a), R(a,a))	w_1
		(P(a), Q(b), R(a,b))	w_1
		(P(b), Q(a), R(b,a))	w_1
		(P(b), Q(b), R(b,b))	w_1
F_2	S(X,Y) :- P(X), Q(Y)	(P(a), Q(a), S(a,a))	w_2
		(P(a), Q(b), S(a,b))	w_2
		(P(b), Q(a), S(b,a))	w_2
		(P(b), Q(b), S(b,b))	w_2
F_3	V(X,Y) :- Q(X), Q(Y)	(Q(a), V(a,a))	w_3
		(Q(a), Q(b), V(a,b))	w_3
		(Q(b), Q(a), V(b,a))	w_3
		(Q(b), V(b,b))	w_3
F_4	R(X,Y) :- V(X,Y)	(R(a,a), V(a,a))	w_4
		(R(a,b), V(a,b))	w_4
		(R(b,a), V(b,a))	w_4
		(R(b,b), V(b,b))	w_4

Fig. 2. List of features associated to the formulas in the **MLN** of Example 2

variables associated to the nodes. The probability distribution over the possible worlds can be written as follows:

$$Pr(\mathbf{X}=\mathbf{x}) = \frac{1}{Z}\prod_{k=1}^{4} e^{w_k n_k(\mathbf{x})} = \frac{1}{Z}\, e^{2\, n_1(\mathbf{x})} \cdot e^{1.5\, n_2(\mathbf{x})} \cdot e^{0.8\, n_3(\mathbf{x})} \cdot e^{1.7\, n_4(\mathbf{x})} \qquad (3)$$

In equation (3), $n_k(\mathbf{x})$ is the number of features associated to formula F_k that are true in the world $\mathbf{x}$. In the extreme case when all the variables are true (let $\mathbf{x_1}$ be the corresponding world), we have $n_1(\mathbf{x_1}) = n_2(\mathbf{x_1}) = n_3(\mathbf{x_1}) = n_4(\mathbf{x_1}) = 4$, and then:

$$Pr(\mathbf{X}=\mathbf{x_1}) = \frac{1}{Z}\, e^{4\ (2+1.5+0.8+1.7)} = \frac{e^{24}}{3.023 \cdot 10^{14}} = 8.76 \cdot 10^{-5}$$

On the opposite, when all the variables are false (let $\mathbf{x_0}$ be the corresponding world), we still have $n_1(\mathbf{x_0}) = n_2(\mathbf{x_0}) = n_3(\mathbf{x_0}) = n_4(\mathbf{x_0}) = 4$, because all the rules' premises are false and hence all the groundings of the rules are true; then:

$$Pr(\mathbf{X}=\mathbf{x_1}) = Pr(\mathbf{X}=\mathbf{x_0})$$

The following result could be easily shown:

Proposition. When the knowledge base $\mathcal{K}$ consists only of positive rules, then the probability of the world where all the variables are false is the same as the probability of the world where all all the variables are true, and this probability is maximal.

As a further example, let us consider the world $\mathbf{x_3}$ in which P(a) = 1, P(b) = 1, Q(a) = 0, Q(b) = 1, R(a,b) = 0, S(a,b) = 1, R(b,b) = 0, S(b,b) = 0, R(b,a) = 1, S(b,a) = 0, R(a,a) = 0, S(a,a) = 0, V(a,a) = 0, V(b,b) = 0, V(a,b) = 1, V(b,a) = 0. In this world, we have $n_1(\mathbf{x_2}) = 2, n_2(\mathbf{x_2}) = 3, n_3(\mathbf{x_2}) = 3, n_4(\mathbf{x_2})$ = 3, and, hence:

$$Pr(\mathbf{X} = \mathbf{x_3}) = \frac{1}{Z}\, e^{2\cdot 2+1.5\cdot 3+0.8\cdot 3+1.7\cdot 3} = 2.94\ 10^{-8}$$

We may notice that, from the point of view of computing probability distribution (2), the translation of the Markov logic network **MLN** into the Markov network **MN** is not useful. As spurious cliques may appear in the transformation, it is not possible to write formula (2) from **MN** only. In fact, even though, in the general case, all cliques contribute to the probability distribution with a potential function, in this case only those that correspond to groundings of formulas in **MLN** must be considered, and these are only a subset of all the cliques occurring in the network; then, a further check is needed to discover which ones they are.

3.2 Inference

Given a Markov Logic Network **MLN** and a set **C** of constants, the central inference problem can be formulated as follows:

”What is the probability that a ground formula φ_1 is true, knowing that another ground formula φ_2 is true?”[2].

In general, to find an answer to this question is intractable. Then, approximate algorithms are to be used. Richardson and Domingos provide such an algorithm for the special case where φ_1 and φ_2 are conjunctions of ground literals. The algorithm proceeds in two phases; the first one returns the minimal subset of the ground Markov network required to compute $Pr(\varphi_1|\varphi_2, \mathbf{MLN}, \mathbf{C})$. The authors observe that the size of the network returned may be further reduced by ignoring any ground formula which is made true by the evidence (*i.e.*, by φ_2), and by removing the corresponding arcs removed from the network.

The second phase of Richardson and Domingos’ algorithm performs inference on the network returned in the first one, with the nodes in φ_2 set to their values. Their algorithm implementation uses Gibbs sampling. The basic Gibbs step consists of sampling one ground atom given its Markov blanket. The Markov blanket of a ground atom is the set of ground atoms that appear in some grounding of a formula with it. In a Markov network, the Markov blanket of a node is the set of all its immediate neighbors. Let X be the binary variable associated to a node in **MN**, and let $\mathcal{B}(X)$ be its Markov blanket. The probability that X is true, given the state **s** of $\mathcal{B}(X)$, can be estimated through Gibbs sampling as follows:

$$Pr(X = 1|\mathcal{B}(X) = \mathbf{s}) = \frac{e^{\sum_{i=1}^{M} w_i \cdot n_i\text{”}(X=1,\mathcal{B}(X)=\mathbf{s})}}{e^{\sum_{i=1}^{M} w_i \cdot n_i\text{”}(X=1,\mathcal{B}(X)=\mathbf{s})} + e^{\sum_{i=1}^{M} w_i \cdot n_i\text{”}(X=0,\mathcal{B}(X)=\mathbf{s})}} \tag{4}$$

In (4) n_i”$(X = 0, \mathcal{B}(X) = \mathbf{s})$ is the number of true features involving X, for $X = 1$ and $\mathcal{B}(X) = \mathbf{s}$. The estimated probability is the fraction of samples in which the conjunction is true, once the Markov chain has converged.

[2] In the following, we use greek letters to denote ground formulas, not to be confused with the rules in $\mathcal{K}$.

3.3 Learning

One way in which learning occurs is acquiring the weights, given the structure of the Markov logic network. Weights can be learned from databases under the *closed world* assumption: if an atom occurs in the database, it is true, otherwise it is assumed to be false. If there are n possible ground atoms, a database is effectively a vector $\mathbf{x} = (x_1,...,x_k,...,x_n)$, where x_k is the truth value of the k-th ground atom ($x_k = 1$ if the atom appears in the database, and $x_k = 0$ otherwise).

The weights could be learned through maximization of the log-likelihood with respect to the weights themselves, but this process is computationally intractable, and Richardson and Domingos propose a more efficient method, based on the optimization of the pseudo-likelihood:

$$Pr^*_w(\mathbf{X} = \mathbf{x}) = \prod_{k=1}^{n} Pr_w(X_k = x_k | \mathcal{B}_{\mathbf{x}}(X_k)), \quad (5)$$

where $\mathcal{B}_x(X_k)$ is the Markov blanket of X_k in data $\mathbf{x}$. Optimization of equation (5) is done using the limited-memory BFGS algorithm [LN89].

If the network structure has to be learned (totally or partially), any relational or ILP learner could be used to the purpose.

4 Bayesian Programs

Kersting and De Raedt [KdR07] have proposed an approach based on Bayesian networks, extending the traditional notions of atom, clauses, and interpretation of a clause. The core of their approach is the notion of *Bayesian program*.

4.1 Bayesian Programs

In Bayesian programs, the notion of *Bayesian atom* is introduced: it is built with a predicate and with terms, but it takes a finite set of possible values, instead of simply $true$ or $false$. For instance, in first order logic, the marital status of a person (single, married, divorced, ...) is represented either by introducing a predicate $single(X)$, $married(X)$, ...for each possible value, or by introducing a binary predicate $status(X,Y)$, where Y is instantiated by a constant representing the marital status. Kersting and De Raedt suggest to use a predicate $status$, where $status(X)$ can take a finite set of possible values $\{single, married, \ldots\}$[KdR07]. Let us notice that in this framework, a classical atom is a particular case of a Bayesian atom with two values $\{true, false\}$.

Definition 2. *A* Bayesian clause *c is an expression $A|A_1,\ldots,A_n$, where A, A_1, $\ldots A_n$ are Bayesian atoms. The atom A is called the* head *of c, written $head(c)$, whereas A_1, $\ldots$, A_n is called the* body *of c, written $body(c)$. When $n = 0$, such a clause is called a Bayesian fact. A* Bayesian program *is a finite set of Bayesian clauses. Moreover, for each clause c there exists a conditional probability distribution $cpd(c)$ that encodes $Pr(head(c)|body(c))$.*

It is assumed that Bayesian clauses are range-restricted, *i.e.*, all the variables that occur in A also occur in $A_1, \ldots, A_n$.

Example 3. By transforming the knowledge base $\mathcal{K}$ of Example 1 into a Bayesian program, we obtain four clauses:

$$\mathcal{P}_{\mathcal{K}} = \{[R(X,Y)|P(X),Q(Y)],[S(X,Y)|P(X),Q(Y)],[V(X,Y)|Q(X),Q(Y)],\\ [R(X,Y)|V(X,Y)]\}$$

Conditional probability distributions must be associated to the rules, in the form of tables.

4.2 Semantics

A directed graph $\mathcal{G}_{\mathcal{P}}$ is associated to a Bayesian logic program $\mathcal{P}$ as follows: the nodes of $\mathcal{G}_{\mathcal{P}}$ are the elements of $\mathcal{M}_{\mathcal{P}}$, and there is an edge from x to y if there exists a clause c of $Inst\text{-}rel(\mathcal{P})$ s.t. $x \in body(c)$ and $y = head(c)$.

An important point that differs from propositional Bayesian networks is that, given a node y, there may be several clauses with $head(c) = y$. In such cases, there is an edge starting from each ground atom occurring in the bodies of these clauses and ending at y, thus losing in the graphical representation the structured information encapsulated in the clauses.

As already mentioned, in a Bayesian logic program $\mathcal{P}$ a conditional probability distribution is associated to each clause. Nevertheless, as several clauses of $Inst\text{-}rel(\mathcal{P})$ may have the same head, it is necessary to define some *combining rule* [NH97] that maps finite sets of conditional probability distributions $P(A|A_{i_1},\ldots,A_{i_{n_i}})$, $i = 1,\ldots,m$, onto a single one $P(A|B_1,\ldots,B_k)$, where $\{B_1,\ldots,B_k\} \subseteq \cup_{i=1}^{m}\{A_{i_1},\ldots,A_{i_{n_i}}\}$. It is shown that if $\mathcal{M}_{\mathcal{P}} \neq \emptyset$, $\mathcal{G}_{\mathcal{P}}$ is acyclic, and each node is influenced by a finite set of random variables, then $\mathcal{P}$ specifies a unique probability distribution over $\mathcal{M}_{\mathcal{P}}$. Two rules that are widely employed in Bayesian networks are *Max* and *Noisy-Or*.

4.3 Inference

Each Bayesian logic program specifies a propositional Bayesian net, where inference can be done using standard available algorithms. The structure of the network follows from the semantics of the logic program, whereas the quantitative aspects are encoded in the conditional probability distributions associated to the nodes and in the combining rule. The stochastic variables associated to the nodes of the net are the atoms occurring in the *least Herbrand model* of the program, namely all ground atoms that are logically entailed by the program. In order to make inference it is then necessary to provide some ground facts, *i.e.*, the truth values of some of the elements of the Herbrand base.

When the logic program is encoded as a propositional Bayesian net, any inference algorithm can be used. Inference takes the form of answering a query, and this can be done with or without available evidence. More formally, a query is the expression:

$$Q \mid E_1,\ldots,E_m$$

where Q is a random variable and $m \geqslant 0$. Answering the query means to find the conditional probability distribution:

$$Pr(Q \mid E_1, \ldots, E_m)$$

Inference without evidence. Answering Q without evidence (the case $m = 0$) amounts to compute the probability distribution $Pr(Q)$ over the possible values assumed by Q. To compute this probability, it is not necessary to consider the whole network, but only the part of it containing the *relevant* nodes, *i.e.*, the nodes corresponding to the variables that influence Q. In order to answer the query (in the probabilistic sense), Kersting et al. [KDRK00] use a *pruned AND-OR tree*, which allows the probabilistic and logic computations to be combined. A pruned AND-OR tree represents all proofs of the query, because all branches leading to failures are pruned. It is interesting to note that each ground atom has a unique pruned AND-OR tree. When the same ground atom A occurs more than once in a tree, all nodes corresponding to it must be merged, so that the tree becomes an AND-OR graph.

For computing the probabilities, to each branch from an OR to an AND node the corresponding conditional probability distribution is associated. If Y is the random variable corresponding to the considered OR node, its conditional probability distribution is computed according to the combining rule for the predicate in Y.

Inference with evidence. Evidence takes the form of a set of ground random variables $\{E_1, E_2, ..., E_n\}$ and their associated values $\{e_1, e_2, ..., e_n\}$. The Bayesian network necessary to compute the probability of the query random variable Q is the union of all the pruned AND-OR graphs for the facts $\{Q, E_1, E_2, ..., E_n\}$. This network can be computed incrementally, starting from the graph for Q and adding those corresponding to each piece of evidence in turn. In this way, there is the guarantee that each node occurs only once in the graph. Once the global graph is built up, any Bayesian net inference engine can be used to compute $Pr(Q \mid E_1 = e_1, ..., E_n = e_n)$.

4.4 Learning

Learning a Bayesian program may consist of two parts: learning the clauses, and learning the associated conditional probability distributions. For the first part, ILP algorithms [MdR94] could be helpful. Once the Bayesian program is learned (or given), the probability distributions must be acquired. To this aim, Kerstings and de Raedt suggest that a technique proposed by Koller and Pfeffer [KP97], based on the EM approach, could be adapted. This algorithm makes two assumptions: (1) different data cases are independent, and (2) the combining rules are decomposable, *i.e.*, they can be expressed using a set of separate nodes corresponding to different influences, which are then combined in another node.

5 Comparison between the Two Models

In this section we will use the example introduced previously to make a comparison between the two models. Given the knowledge base $\mathcal{K}$ of Example 1, let us now associate to the rules of the knowledge base $\mathcal{K}$ some probability distributions, reported in

$P(X)$	$Q(Y)$	$R(X,Y)$	$P(X)$	$Q(Y)$	$S(X,Y)$
true	*true*	$(0.7,0.3)$	*true*	*true*	$(0.3,0.7)$
true	*false*	$(0.5,0.5)$	*true*	*false*	$(0.5,0.5)$
false	*true*	$(0.5,0.5)$	*false*	*true*	$(0.5,0.5)$
false	*false*	$(0.5,0.5)$	*false*	*false*	$(0.5,0.5)$

Fig. 3. Probability distributions associated to the the first two clauses (F_1 and F_2) in $\mathcal{K}$. The 0.5 values in the last three rows on the table indicate that there is total uncertainty about the consequence of a rule, when the premise is false.

$Q(X)$	$Q(Y)$	$V(X,Y)$	$V(X,Y)$	$R(X,Y)$
true	*true*	$(0.6,0.4)$	*true*	$(0.2,0.8)$
true	*false*	$(0.5,0.5)$		
false	*true*	$(0.5,0.5)$	*false*	$(0.5,0.5)$
false	*false*	$(0.5,0.5)$		

Fig. 4. Probability distributions associated to the second two clauses (F_3 and F_4) in $\mathcal{K}$. As in Figure 3, a false premise does not supply any information about the truth of the consequence.

Figures 3 and 4. In addition, it is necessary to provide a combining procedure for those rules that share the consequent. Let us assume here to use the *Max* combination rule. Moreover, let us assume, as an example, that:

$$Pr(P(a)=1)=0.4, \quad Pr(P(b)=1)=0.6, \quad Pr(Q(a)=1)=0.5, \quad Pr(Q(b)=1)=0.7$$

If we want to represent $\mathcal{K}$ and the probability distributions with the graphical model associated to Kersting and de Raedt's Bayesian program $\mathcal{P}_{\mathcal{K}}$, we obtain the graph $\mathcal{G}_{\mathcal{P}}$ depicted in Figure 5. Actually, this graph is reported here only for illustration purposes, because it is not really built up, as only the part relevant to perform a required inference is actually constructed.

As we may notice, the graph in Figure 5 has the same nodes as the one in Figure 1, but the edges (in addition of being oriented) are different. For what concerns the probability distributions, they can be embedded in the Bayesian model as they are, in the sense that the same tables, reported in Figure 3 and 4 as related to the rules of the knowledge base, can be considered associated to the corresponding edges in the graph, provided that just one node for each ground predicate occurs. The combining rule $\otimes$ must be applied when necessary. In our case, for instance:

$$Pr(R(a,b))|P(a),Q(b),V(a,b)) = Pr(R(a,b))|P(a),Q(b)) \otimes Pr(R(a,b))|V(a,b))$$

By numbering the node of the graph in Figure 5 from 1 to 16, we can write the usual joint probability distribution over all nodes:

$$Pr(X_1,\ldots,X_{16}) = \prod_{i=1}^{16} Pr(X_i \mid par(X_i)) \qquad (6)$$

Expression (6) is the analogue, in the Bayesian framework, of expression (4) in the Markovian framework, and it can be used to perform inference, even without evidence.

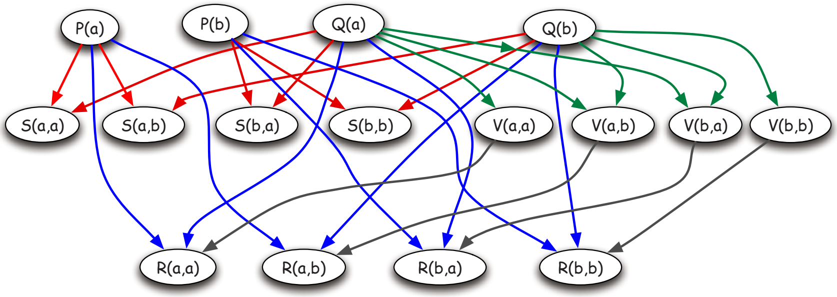


Fig. 5. Global graph associated to the Bayesian logical program reported in Example 3. As it may be seen, nodes $R(a,a), R(a,b), R(b,a)$, and $R(b,b)$ receive edges from the atoms occurring in the bodies of both rule F_1 and rule F_4.

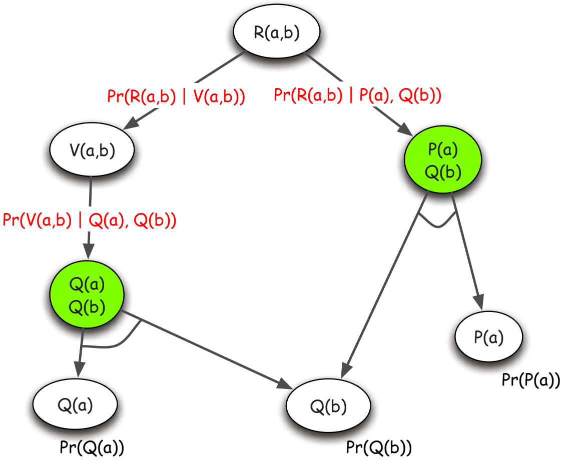


Fig. 6. AND-OR tree associate to the query $Q = R(a,b)$. In the tree, the two occurrences of the node $Q(b)$ have been merged. Nodes $[Q(a)Q(b)]$ and $[P(a)Q(b)]$ are AND nodes, *i.e.*, they are true if all the children are true. Probabilities written on the edges are conditional distributions, associated to the corresponding branches of the tree, whereas probabilities written outside the nodes are a priori probabilities, and are associated to leaves in the tree.

In the Markovian framework it is not immediate to translate the probability distributions given for $\mathcal{K}$ into the joint distribution (2).

Let us now consider the problem of answering the query $\varphi_1 = R(a,b)$, given that $Pr(P(a) = 1)) = 0.4, Pr(Q(b) = 1) = 0.7, Pr(Q(a) = 1) = 0.5$. In Figure 6 the AND-OR tree associated to the query is reported. Using the probabilities defined in Figures 3 and 4, we can construct the tables in Figures 7 and 8. We define by $R_1(a,b)$ the ground predicate obtained from formula F_1, and by $R_4(a,b)$ the one obtained from formula F_4. The probability $Pr(R(a,b) = false)$ is the complement to 1 of $Pr(R(a,b) = true)$. These probabilities must be computed for each row of the table in Figure 7, and the

$P(a)$	$Q(a)$	$Q(b)$	$P(a)Q(b)$	$Q(a)Q(b)$	$V(a,b) = (true, false)$
0.4	0.5	0.7	0.28	0.35	$(0.21, 0.14)$
0.4	0.5	0.3	0.12	0.15	$(0.075, 0.075)$
0.4	0.5	0.7	0.28	0.35	$(0.175, 0.175)$
0.6	0.5	0.7	0.42	0.35	$(0.21, 0.14)$
0.6	0.5)	0.7	0.42	0.35	$(0.175, 0.175)$
0.6	0.5	0.3	0.18	0.15	$(0.075, 0.075)$
0.4	0.5	0.3	0.12	0.15	$(0.075, 0.075)$
0.6	0.5	0.3	0.18	0.15	$(0.075, 0.075)$

Fig. 7. The AND-OR tree of Figure 6 is used to compute the probability distribution of the query $R(a,b)$. In this table, the probability distribution of the random variable $V(a,b)$ is computed for each possible combination of $P(a) \in \{1,0\}, Q(a) \in \{1,0\}$, and $P(b) \in \{1,0\}$. The values reported in the first three columns of the table derive from Tables 3 and 4.

$R_1(a,b) = (true, false)$	$R_4(a,b) = (true, false)$	$R(a,b) = true$
$(0.7, 0.3)$ 0.28	$(0.32, 0.68)$ 0.35	0.196
$(0.5, 0.5)$ 0.12	$(0.35, 0.65)$ 0.15	0.06
$(0.7, 0.3)$ 0.28	$(0.35, 0.65)$ 0.35	0.196
$(0.5, 0.5)$ 0.42	$(0.32, 0.68)$ 0.35	0.21
$(0.5, 0.5)$ 0.42	$(0.35, 0.65)$ 0.35	0.21
$(0.5, 0.5)$ 0.18	$(0.35, 0.65)$ 0.15	0.09
$(0.5, 0.5)$ 0.12	$(0.35, 0.65)$ 0.15	0.06
$(0.5, 0.5)$ 0.18	$(0.35, 0.65)$ 0.15	0.09

Fig. 8. The query $R(a,b)$ can be answered through two rules, F_1 and F_4 (see Example 1). The results from these two rules must be combined. In the third column only the probability $Pr(R(a,b) = true$ is reported.

results are reported in Figure 8.The third column is computed according to the following combining rule:

$$Pr(R(a,b) = 1) = Max\{Pr(R_1(a,b) = 1), Pr(R_4(a,b) = 1)\}$$

Concerning the Markovian framework, in order to estimate the probability of $R(a,b)$, we have to sample from its Markov blanket, knowing the state of this last (see, formula (6)). The rows in Figures 7 and 8 correspond to different states of the Markov network, depending on the truth of $P(a), Q(b)$ and $Q(a)$. Let us consider the case, as an example, of $P(a) = 1, Q(b) = 1$ and $Q(a) = 1$, which are then the evidence. First of all, we have to extract, using the first part of Richardson and Domingos' algorithm, the relevant part of the net, which is reported in Figure 9.

The probability distribution over the nodes in Figure 9, which is the marginal of distribution (4) with respect to the interested variables, is the following one:

$$Pr(\mathbf{X}' = \mathbf{x}') \equiv Pr(X_1 = x_1, X_2 = x_2, X_4 = x_4, X_5 = x_5, X_{15} = x_{15}) = \\ \frac{1}{Z'} e^{w_1 \cdot n'_1(\mathbf{x}')} e^{w_3 \cdot n'_3(\mathbf{x}')} e^{w_4 \cdot n'_4(\mathbf{x}')},$$

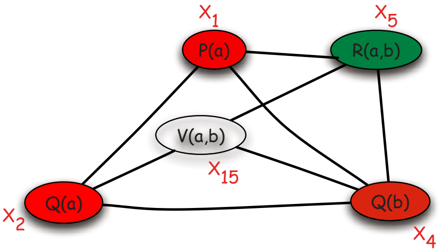


Fig. 9. Part of the Markov network of Figure 1 necessary to compute $Pr(R(a,b)|P(a),Q(a),Q(b),\mathbf{MN},\mathbf{C})$. Nodes $P(a),Q(b)$, and $Q(a)$ are evidence nodes, whereas $R(a,b)$ is the query node.

where $\mathbf{x}'$ is a subvector of $\mathbf{x}$ only involving the surviving variables, and n'_i is the number of cliques that are true in the subgraph of Figure 9. In order to proceed further, we have to assume to have estimated a suitable vector of weights $\mathbf{w}$. If this is the case, we may apply Gibbs samplig to the subgraph in Figure 9, obtaining thus an estimate of $Pr(R(a,b))$, using the same procedure exemplified in Section 3.

Let us now give a last example for illustrating the differences that appear when modelling a knowledge base in both frameworks.

Example 4 (Communication Modelling)
We aim at modelling different ways of communicating between people. Let us consider the following Markov Logic Network $\mathbf{MLN} = \{(F_i, w_i)|1 \leqslant i \leqslant 6\}$, constructed by adding weights to the rules of a knowledge base $\mathcal{K}$:

$$\begin{aligned}
(F_1, 1.8) &= phone(X,Y) :- \ friends(X,Y), communic(X,Y)\\
(F_2, 1.2) &= email(X,Y) :- \ friends(X,Y), communic(X,Y)\\
(F_3, 1.0) &= phone(X,Y) :- \ coworkers(X,Y), communic(X,Y)\\
(F_4, 2.0) &= email(X,Y) :- \ coworkers(X,Y), communic(X,Y)\\
(F_5, 2.0) &= coworkers(X,Z) :- \ coworkers(X,Y), coworkers(Y,Z)\\
(F_6, 0.8) &= friends(X,Y) :- \ coworkers(X,Y)
\end{aligned}$$

Let moreover $\mathbf{C} = \{\texttt{Ann}, \texttt{Bob}, \texttt{Tom}\}$ be a set of constants.

In the framework of Bayesian logic programs, such knowledge can be represented by the same set of formulas, but we can also use the notion of Bayesian atoms to represent it in a more concise way. We can for instance introduce a predicate *comean* taking different values *phone*, *email*, ..., leading thus to

$$\begin{aligned}
commean(X,Y) &\mid friends(X,Y), communic(X,Y)\\
commean(X,Y) &\mid coworkers(X,Y), communic(X,Y)\\
coworkers(X,Z) &\mid coworkers(X,Y), coworkers(Y,Z)\\
friends(X,Y) &\mid coworkers(X,Y)
\end{aligned}$$

A conditional probability must be associated to each of the clause. For instance, for the first clause, we could have:

$communic(Y,X)$	$friends(X,Y)$	$Pr(commean(X,Y)\|body)(phone,mail)$
$true$	$true$	$(0.7,0.3)$
$true$	$false$	$(0.5,0.5)$
$false$	$true$	$(0.5,0.5)$
$false$	$false$	$(0.5,0.5)$

There is still another way of representing this knowledge, namely by introducing a predicate *relation* that can take three values $\{friend, coworker, none\}$ The first two rules can then be written with a single rule. On the other hand, the last two rules must be generalized, thus requiring more probabilities to be given.

$$\begin{aligned} commean(X,Y) &\mid relation(X,Y), communic(X,Y) \\ relation(X,Z) &\mid relation(X,Y), relation(Y,Z) \\ relation(X,Y) &\mid relation(X,Y) \end{aligned}$$

$communic(Y,X)$	$relation(X,Y)$	$Pr(commean(X,Y)\|body)(phone,mail)$
$true$	$friend$	$(0.7,0.3)$
$true$	$coworker$	$(0.2,0.8)$
$true$	$none$	$(0.5,0.5)$
$false$	$friend$	$(0.5,0.5)$
$false$	$coworker$	$(0.5,0.5)$
$false$	$none$	$(0.5,0.5)$

$relation(X,Y)$	$relation(Y,Z)$	$Pr(relation(X,Z)\|body)(phone,mail)$
$true$	$friend$	$(0.7,0.3)$
$true$	$coworker$	$(0.2,0.8)$
$true$	$none$	$(0.5,0.5)$
$false$	$friend$	$(0.5,0.5)$
$false$	$coworker$	$(0.5,0.5)$
$false$	$none$	$(0.5,0.5)$

6 Conclusions

According to Section 4, the two models, Markov Logic Networks and Bayesian Logic Programs, are essentially different, at least for what concerns representation and inference. The Markov framework, being based on computing probabilities according to the numbers of true groundings of formulas in the knowledge base, assigns high probability to many groundings corresponding to false premises, whereas the Bayesian model suspends the judgement, giving a neutral probability assignment to these cases. For the user point of view, using Markov networks is then rather counterintuitive, because when many rules have a false premise, a drift towards neutrality (probability near 1/2 or near to some a-priori value) would be more intuitively acceptable than a high probability assignment.

In Richardson and Domingos' approach, general formulas are considered, but they do not address the semantics of negation. A weight is associated to a formula. All the instantiations of a same formula have the same weight.

In Kersting and de Raedt's approach only Datalog clauses are considered, and they distinguish two types of predicates: logical (which can be either *true* or *false*) and Bayesian (which can assume values in a given set). The definition of Bayesian atoms allows different probabilities to be assigned, depending on the values of the atoms in the bodies. On the other hand, this facility complicates the probability distribution tables.

Given a knowledge base, the Bayesian approach seems more natural and easier to be built up and understood. We may notice that the graphical models may be very large. In the Markovian approach the graph is built up and then it is reduced when evidence is given. In the Bayesian approach the complete graph is not constructed, but, adding the known facts to the knowledge base, only the relevant part of it is actually constructed.

Even though, formally, both Markov and Bayesian networks can be reduced to a common representation, important differences, at the semantic level, may direct the user to the choice of one or the other approach. One difference resides in the parameters that the user must provide, in order to build the model. In the Markov network approach, the user must provide the weight of the formulas in the knowledge base $\mathcal{K}$. Once the weight are provided, any query can be answered. As always with weights, the user may or may not be able to give useful weights; in this case, in addition, he/she may have an idea of the range of probabilities that he/she is expecting for a given query, given the evidence. To provide weights consistent with his/her expectations may be very difficult for the user. In the Bayesian approach, the user must provide the conditional probability distributions, which also may be difficult, even though these distributions are local, and hence more close to the user's understanding.

Both models can avoid asking the user for the parameters, and automated learning can be applied. In this case, the burden on the user is lifted, but still the interpretation of the results poses different challenges.

A final issue regarding the representation is the combination rule, which allows rules with the same consequent to be merged. In the Markov network approach, no explicit combination rule exists, because combination is taken into account in the assignment of values to the cliques' features. In the Bayesian model, the combination rule must be explicitely provided. Globally, the Bayesian model construction has more degrees of freedom.

Concerning inference, both models try to answer a query, given evidence. To answer a query means, in both, to compute the probability of a formula. The Bayesian approach restricts the Bayesian network to the atoms occurring in the least Herbrand model. The facts must be given in the knowledge base. In order to make inference on a given atom, all its proofs must be computed, considering Bayesian rules as logical rules and combining the associated probabilities. Both models are intractable, and must revert to approximations to make inferences. In the Markov network approach, approximate inference is done, by using Gibbs sampling on the Markov blanket of the nodes in the network. Then, the complexity of the inference depends on the size of the relevant part of the Markov network extracted from the global one by Richardson and Domingos' algorithm. In the worst case, the size os this subnetwork may coincide with that of the global one.

In the Bayesian approach it is not clear what approximations are suggested, except those that can be applied to generic Bayesian networks. Also in this case, the computational complexity of the inference is determined by the size of the network corresponding to the least Herbrand universe. The inference process, per se, has the same complexity as the best algorithm that work on Bayes nets.

Finally, learning is the more similar issue in the two approaches. In fact, learning the knowledge base can be done with an ILP learning algorithm in both cases, and then the weights or the conditional probability distributions can be estimated from a set of data. The Bayesian approach is described as an example of the learning from interpretations framework.

Acknowledgements. Christel Vrain'work is partly supported by the ANR project 071*N*: *Semi-supervised learning of graph structure by statistical and relational approaches: towards the identification of biological networks from heterogeneous data.*

References

[Cus01] Cussens, J.: Statistical aspects of stochastic logic programs. In: Jaakkola, T., Richardson, T. (eds.) Artificial Intelligence and Statistics 2001: Proceedings of the Eighth International Workshop, pp. 181–186. Morgan Kaufmann, San Francisco (2001)

[DR04] Domingos, P., Richardson, M.: Markov logic: A unifying framework for statistical relational learning. In: Proc. of the ICML Workshop on Statistical Relational Learning and its connections to other Fields (2004)

[FGKP99] Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI 1999), Stockholm, Sweden, pp. 1300–1309. Morgan Kaufmann, San Francisco (1999)

[Get06] Getoor, L.: An introduction to probabilistic graphical models for relational data. Data Engineering Bulletin 29(1) (March 2006)

[Hal96] Halpern, J.Y.: An analysis of first-order logics of probability. Artificial Intelligence 46, 311–350 (1996)

[Jae97] Jaeger, M.: Relational Bayesian networks. In: Geiger, D., Shenoy, P.P. (eds.) Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI 1997), pp. 266–273. Morgan Kaufmann, San Francisco (1997)

[Jae02] Jaeger, M.: Relational bayesian networks: a survey. Linkšping Electronic Articles in Computer and Information Science 6 (2002)

[KD05] Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: ICML 2005: Proceedings of the 22nd international conference on Machine learning, pp. 441–448. ACM Press, New York (2005)

[KdR07] Kersting, K., de Raedt, L.: Bayesian logic programming: Theory and tool. MIT Press, Cambridge (2007)

[KDRK00] Kersting, K., De Raedt, L., Kramer, S.: Interpreting bayesian logic programs. In: Getoor, L., Jensen, D. (eds.) Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data, Technical Report WS-00-06. AAAI Press, Menlo Park (2000)

[KP97] Koller, D., Pfeffer, A.: Learning probabilities for noisy first-order rules. In: Proc. IJCAI 1997 (1997)

[LN89] Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical Programming 45, 503–528 (1989)

[MdR94] Muggleton, S., de Raedt, L.: Inductive logic programming:theory and methods. J. of Logic Programming 19, 629–679 (1994)

[Mug00] Muggleton, S.H.: Learning stochastic logic programs. In: Getoor, L., Jensen, D. (eds.) Proceedings of the AAAI 2000 workshop on Learning Statistical Models from Relational Data. AAAI, Menlo Park (2000)

[NH97] Ngo, L., Haddaway, P.: Answering queries from context-sensitive probabilistic knowledge base. Theoretical Computer Science 171, 147–177 (1997)

[RD06] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)

Brave Induction

Chiaki Sakama[1] and Katsumi Inoue[2]

[1] Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ki@nii.ac.jp

Abstract. This paper considers the following induction problem. Given the background knowledge B and an observation O, find a hypothesis H such that a consistent theory $B \wedge H$ has a minimal model satisfying O. We call this type of induction *brave induction*. Brave induction is different from explanatory induction in ILP, which requires that O is satisfied in every model of $B \wedge H$. Brave induction is useful for learning *disjunctive* rules from observations, or learning from the background knowledge containing indefinite or incomplete information. We develop an algorithm for computing brave induction, and extend it to induction in answer set programming.

1 Introduction

Logical foundations for *induction* is one of the central topics in machine learning, and different theories of induction have been proposed in the literature [1,2,3,4,13,14,15,19,21,25]. A typical induction task constructs hypotheses to explain an observation (or examples) using the background knowledge. More precisely, given the background knowledge B and an observation O, a hypothesis H *covers* O under B if

$$B \wedge H \models O \tag{1}$$

where $B \wedge H$ is consistent. This style of induction is often called *explanatory induction* [8] and is usually used in *inductive logic programming* (ILP) [20].

By the definition, explanatory induction requires that a possible solution H together with B logically entails O. In other words, O is true in *every* model of $B \wedge H$. This condition is often too strong for building possible hypotheses, however. Suppose that there are 30 students in a class. Of which 20 are European, 7 are Asian, and 3 are American. The situation is represented by the background knowledge B and the observation O:

B : $\mathit{student}(1) \wedge \cdots \wedge \mathit{student}(30)$,

O : $\mathit{euro}(1) \wedge \cdots \wedge \mathit{euro}(20) \wedge \mathit{asia}(21) \wedge \cdots \wedge \mathit{asia}(27) \wedge \mathit{usa}(28) \wedge \cdots \wedge \mathit{usa}(30)$,

where each number represents individual students. In this case, the following clause, saying that every student is either European, Asian, or American, appears a good hypothesis:

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 261–278, 2008.

$$H: \ \ euro(x) \vee asia(x) \vee usa(x) \ \leftarrow \ student(x)\,. \tag{2}$$

Unfortunately, however, H does not satisfy the relation $B \wedge H \models O$. In fact, $B \wedge H$ has many models in which O is not true. An instance of such a model is: $\{\, student(1), \ldots, student(30),\, euro(1), \ldots, euro(30)\,\}$.

Explanatory induction in ILP has been mainly used for learning *Horn* theories. When the background knowledge B and a hypothesis H are Horn theories, $B \wedge H$ has the unique minimal model (or the least model), and the relation (1) represents that O is true in the least model. In this case, the relation (1) is necessary and sufficient to make O an explanatory consequence of $B \wedge H$. On the other hand, when B or H contains *indefinite* information, $B \wedge H$ becomes a *non-Horn* theory which has multiple minimal models in general. An observation O might be true in some minimal models of $B \wedge H$ but not every one. In this case, however, the relation (1) excludes a hypothesis H due to the existence of a single minimal model in which O is not true. As a result, meaningful hypotheses might be unqualified as presented above.

To cope with the problem, this paper introduces a weak form of induction called *brave induction*. In contrast to explanatory induction, brave induction defines that a hypothesis H covers O under B if O is true in *some minimal model* of $B \wedge H$. By the definition, brave induction is weaker than explanatory induction, and the hypothesis (2) becomes a solution of brave induction.

This paper introduces a logical framework of brave induction and develops a procedure for computing hypotheses in brave induction. The proposed framework is further extended to induction from *nonmonotonic* logic programs in *answer set programming* [16]. The rest of this paper is organized as follows. Section 2 introduces the framework of brave induction and develops its computational method. Section 3 applies brave induction to nonmonotonic logic programming. Section 4 discusses related issues, and Section 5 concludes the paper.

2 Brave Induction

2.1 Logical Framework

We first introduce a logical framework of induction considered in this paper. A *first-order language* $\mathcal{L}$ consists of an alphabet and all formulas defined over it. The definition is the standard one in the literature [20]. For induction we use a *clausal language* which is a subset of $\mathcal{L}$.

A *clausal theory* (or simply a *theory*) is a finite set of *clauses* of the form:

$$A_1 \vee \cdots \vee A_m \vee \neg A_{m+1} \vee \cdots \vee \neg A_n \ \ (n \geq m \geq 0)$$

where each A_i $(1 \leq i \leq n)$ is an atom. Any variable in a clause is assumed to be universally quantified at the front. A clause of the above form is also written as

$$A_1 \vee \cdots \vee A_m \leftarrow A_{m+1} \wedge \cdots \wedge A_n\,. \tag{3}$$

$A_1 \vee \cdots \vee A_m$ is the *head* of the clause, and $A_{m+1} \wedge \cdots \wedge A_n$ is the *body*. Given a clause C of the above form, *head*(C) represents the set $\{\, A_1, \ldots, A_m \,\}$ and

$body(C)$ represents the set $\{ A_{m+1}, \ldots, A_n \}$. A clause C is often identified with the set of literals $\{ A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n \}$. A *Horn clause* is a clause of the form (3) with $m \leq 1$. A *Horn theory* is a finite set of Horn clauses. A theory is identified with the conjunction of the clauses in it. A theory, a clause or an atom is *ground* if it contains no variable. A *ground substitution* θ replaces variables $x_1, \ldots, x_k$ occurring in a clause C (resp. an atom A) to ground terms $t_1, \ldots, t_k$ in $C\theta$ (resp. $A\theta$). A ground clause C is *prime* with respect to a theory T if $T \models C$ but $T \not\models C'$ for any $C' \subset C$. A *conjunctive normal form* (CNF) formula is a conjunction of disjunction of literals, and a *disjunctive normal form* (DNF) formula is a disjunction of conjunction of literals. A CNF formula or a DNF formula is *ground* if it contains no variable. A DNF formula $F = c_1 \vee \cdots \vee c_k$ is *irredundant* if $F \not\equiv F'$ for any $F' = c_1 \vee \cdots \vee c_{i-1} \vee c_{i+1} \vee \cdots \vee c_k$ $(1 \leq i \leq k)$.

The *domain* of a theory T is given as the *Herbrand universe* HU and an *interpretation* of T is defined as a subset of the *Herbrand base* HB. An interpretation I *satisfies* a ground clause (3) if $\{ A_{m+1}, \ldots, A_n \} \subseteq I$ implies $\{ A_1, \ldots, A_m \} \cap I \neq \emptyset$. An interpretation I satisfies a theory T if I satisfies every clause in T. In this case, I is a *model* of T and $Mod(T)$ represents the set of all models of T. A model $M \in Mod(T)$ is *minimal* if $N \subseteq M$ implies $M \subseteq N$ for any $N \in Mod(T)$. The set of minimal models of T is written as $MM(T)$. A theory T *entails* a formula F (written as $T \models F$) if F is true in any $I \in Mod(T)$. A theory T is *consistent* if $Mod(T) \neq \emptyset$; otherwise, T is *inconsistent*. A conjunction C of ground atoms is identified with the set of ground atoms in C.

Let B, O and H are all consistent theories, where B, O, and H are respectively called a *background knowledge*, an *observation*, and a *hypothesis*. We assume that B, O and H have the same HU and HB. The task of induction is to construct H when B and O are given. Formally, given the background knowledge B and an observation O, a hypothesis H *covers* O under B if

$$B \wedge H \models O \tag{4}$$

where $B \wedge H$ is consistent. This type of induction is called *explanatory induction* [8] or *learning from entailment* [3], and is usually used in *inductive logic programming* (ILP) [20]. As presented in the introduction, however, explanatory induction is too strong for handling indefinite disjunctive information. To relax the condition of explanatory induction, we introduce a weak form of induction.

Definition 2.1. (brave induction) Let B be the background knowledge and O an observation. A hypothesis H *covers* O under B in *brave induction* if a consistent theory $B \wedge H$ has a minimal model satisfying O. H is called a *solution* of brave induction.

The above definition requires that an observation is satisfied in some minimal model of a consistent theory $B \wedge H$. This is in contrast to the definition of explanatory induction in which an observation must be satisfied in every minimal model of $B \wedge H$. In this sense, explanatory induction of (4) is also called *cautious induction*, hereafter. These names are taken from brave/cautious reasoning in nonmonotonic logics [18] and disjunctive logic programs [6]. A formula F is a consequence of *cautious inference* in a theory T if it is true in every minimal model of T, while F is a consequence of *brave inference* in T if F is true in some minimal model of T.[1] When a theory contains indefinite or incomplete information, brave inference infers more results than cautious inference in general. Brave and cautious inferences have been used in different reasoning tasks of deduction and *abduction* in artificial intelligence. Thus, it is natural to apply brave inference to induction from non-Horn theories containing indefinite or incomplete information. The utility of brave induction has already been illustrated in the introductory example. Some properties of brave induction are provided. In what follows, B, O, and H represent the background knowledge, an observation, and a hypothesis, respectively.

Proposition 2.1. *Brave induction has a solution iff $B \wedge O$ is consistent.*

Proposition 2.2. *If H covers O under B in cautious induction, H is a solution of brave induction. The converse implication also holds when B is a Horn theory.*

Proposition 2.3. *If H is a solution of brave induction, $B \wedge H \wedge O$ is consistent.*

Proposition 2.4. *For any theory $H' \models H$ such that $B \wedge H'$ is consistent, if H is a solution of brave induction, so does H'.*

Proof. The result holds by $B \wedge H' \models B \wedge H$. □

Proposition 2.4 allows us to search the most specific solutions (under implication) rather than all theories. Brave induction is nonmonotonic, that is, two solutions cannot be merged in general.

Proposition 2.5. *The fact that both H_1 and H_2 are solutions of brave induction does not imply that $H_1 \wedge H_2$ is a solution.*

Example 2.1. Let $B = \{\, p(a) \leftarrow \}$ and $O = \{\, q(a) \vee r(a) \leftarrow, \quad \leftarrow q(a) \wedge r(a) \,\}$. Then, both $H_1 = \{\, q(x) \leftarrow p(x) \,\}$ and $H_2 = \{\, r(x) \leftarrow p(x) \,\}$ cover O under B in brave induction, but $H_1 \wedge H_2$ is not.

In Example 2.1, H_1 and H_2 cover O under B in cautious induction, but $H_1 \wedge H_2$ is not. Thus, explanatory induction is also nonmonotonic.

Proposition 2.6. *If H_1 and H_2 are solutions of brave induction, so is $H_1 \vee H_2$.*

Cautious induction also satisfies Proposition 2.6.

Proposition 2.7. *The fact that H covers both O_1 and O_2 under B does not imply H covers $O_1 \wedge O_2$ under B in brave induction.*

Example 2.2. Let $B = \{\, p(x) \vee q(x) \leftarrow r(x), \quad s(a) \leftarrow \}$, $O_1 = \{\, p(a) \,\}$, and $O_2 = \{\, q(a) \,\}$. Then, $H = \{\, r(x) \leftarrow s(x) \,\}$ covers both O_1 and O_2 under B in brave induction, but H does not cover $O_1 \wedge O_2$ under B.

[1] Cautious/brave inference is also called *skeptical*/*credulous* inference.

In cautious induction, on the other hand, if H covers both O_1 and O_2 under B, so does $O_1 \wedge O_2$. Thus, Proposition 2.7 provides a property that distinguishes brave induction from cautious one.

When B has a minimal model satisfying O, O is inferred by brave inference from B. In this case, $H = \mathit{true}$ covers O, which is a trivial and uninteresting solution. The problem of our interest is the case in which B has no minimal model satisfying O. In other words, $\neg O$ is derived from B under the *generalized closed world assumption* (GCWA) [17]. It is worth noting that explanatory induction in Horn theories finds a hypothesis H when a Horn theory B has no minimal model satisfying O. In this case, $\neg O$ is derived from B under the *closed world assumption* (CWA) [22]. Thus, brave induction in non-Horn theories is considered a natural extension of explanatory induction in Horn theories.

2.2 Computation

In this section, we develop an algorithm for computing brave induction. Throughout the section, we assume that (1) any observation O is a conjunction of ground atoms,[2] and (2) any hypothesis H is a finite clausal theory such that each clause has the non-empty head. The first condition is assumed as the normal problem setting in ILP [20]. The second condition is also natural because we are interested in getting any clause which derives an observation together with the background knowledge. The next proposition characterizes the brave induction problem.

Proposition 2.8. *Let B be the background knowledge, H a hypothesis, and O an observation. Then, $B \wedge H$ has a minimal model satisfying O iff there is a disjunction F of ground atoms such that $B \wedge H \models O \vee F$ and $B \wedge H \not\models F$.*[3]

Proof. ($\rightarrow$) Suppose that $B \wedge H$ has a minimal model M such that $M \models O$. Consider a disjunction F of ground atoms satisfying (i) $M \not\models F$ and (ii) $N \models F$ for any $N \in MM(B \wedge H)$ such that $N \not\models O$. Such F is constructed by picking up ground atoms from each $N \setminus M$. Then, $B \wedge H \models O \vee F$ holds. As $M \not\models F$, $B \wedge H \not\models F$.

($\leftarrow$) Suppose that $B \wedge H \models O \vee F$ holds for a disjunction F of ground atoms and $B \wedge H \not\models F$. If $B \wedge H$ has no minimal model satisfying O, $B \wedge H \models O \vee F$ implies $B \wedge H \models F$. This contradicts the assumption that $B \wedge H \not\models F$. □

Step 1: Computing ground hypotheses
By Proposition 2.8, a solution of brave induction is obtained by computing H satisfying

$$B \wedge H \models O \vee F \tag{5}$$

and

$$B \wedge H \not\models F\,. \tag{6}$$

[2] A conjunction O is identified with the set of ground atoms in it.

[3] Related results are shown in [9, Theorem 4.5] in the context of circumscription, and in [12, Corollary 3.5] in terms of abduction.

By (5), it holds that

$$B \wedge \neg O \models \neg H \vee F. \tag{7}$$

$\neg H \vee F$ is thus obtained by deduction from $B \wedge \neg O$. This technique is *inverse entailment* that is originally proposed by Muggleton for induction in Horn theories [19], and is later extended by Inoue to full clausal theories [13].

As H is a clausal theory, put

$$H = (\Sigma_1 \leftarrow \Gamma_1) \wedge \cdots \wedge (\Sigma_k \leftarrow \Gamma_k) \tag{8}$$

where Σ_i $(i = 1, \ldots, k)$ is a disjunction of atoms and Γ_i $(i = 1, \ldots, k)$ is a conjunction of atoms. It then becomes

$$\neg H = (\neg\Sigma_1 \wedge \Gamma_1) \vee \cdots \vee (\neg\Sigma_k \wedge \Gamma_k). \tag{9}$$

Since F is a disjunction of ground atoms, every formula $\neg H \vee F$ in (7) is a disjunctive normal form. From $B \wedge \neg O$, a number of DNF formulas could be deduced. Among them, we take DNF formulas obtained as follows. First, compute *prime CNF* formulas with respect to $B \wedge \neg O$. A prime CNF formula is a conjunction of prime clauses and is obtained by a system of *consequence-finding*. For this purpose, *SOL-resolution* by Inoue [10] is used. Second, construct a DNF formula as follows: given a prime CNF formula $c_1 \wedge \cdots \wedge c_k$, produce an irredundant DNF formula $d_1 \vee \cdots \vee d_l$ where d_i $(1 \leq i \leq l)$ contains a literal from each c_j $(1 \leq j \leq k)$. Then, $B \wedge \neg O \models d_1 \vee \cdots \vee d_l$ holds, and we identify the DNF formula $\neg H \vee F$ of (7) with $d_1 \vee \cdots \vee d_l$. After deriving such ground DNF formulas of the form $\neg H \vee F$, the next problem is to extract $\neg H$ from $\neg H \vee F$. This is simply done as follows. By the assumption, Σ_i in H is non-empty, so that $\neg H$ is a DNF formula in which each disjunct $\neg\Sigma_i \wedge \Gamma_i$ of (9) contains at least one negative literal. On the other hand, F is a disjunction of ground atoms. Thus, from the DNF formula $\neg H \vee F$, $\neg H$ is extracted by selecting disjuncts containing negative literals. As such, any ground DNF formula $\neg H$ is obtained. From this $\neg H$, we can obtain a clausal theory H such that $B \wedge H$ has a minimal model satisfying O (by Proposition 2.8).

Step 2: Generalization

H is a clausal theory containing no variable, so that H is generalized in the next step. Two cases are considered: (a) O contains a single predicate, and (b) O contains multiple different predicates. In case of (a), we apply Plotkin's *least generalization under subsumption (LGS)* [21] to H. The LGS of any finite set of clauses exists and is computed by the LGS algorithm in [20,21]. The result of LGS is written as $lgs(H)$. In case of (b), let n be the number of different predicates appearing in O. Then, O is partitioned into disjoint subsets:

$$O = O_1 \wedge \cdots \wedge O_n \tag{10}$$

where O_i $(1 \leq i \leq n)$ is a conjunction of ground atoms having the same predicate. Correspondingly, H is partitioned as

$$H = H_1 \wedge \cdots \wedge H_n \tag{11}$$

where H_i $(1 \leq i \leq n)$ is a conjunction of clauses whose heads contain the predicate in O_i. The LGS of each H_i is then computed and collected as

$$lgs(H) = lgs(H_1) \wedge \cdots \wedge lgs(H_n)\,. \tag{12}$$

Note that the equation (12) also represents the result of (a) when $n = 1$.

Step 3: Constructing a weak form of hypotheses
When an observation has some specific property, $lgs(H_i)$ is combined into a weaker formula.

Definition 2.2. (synchronous) Let $pred(A)$ be the predicate of an atom A, and $const(A)$ the set of constants in A. Given a set S of atoms, suppose two atoms A_1 and A_2 in S such that $pred(A_1) = p_1$, $pred(A_2) = p_2$, and $p_1 \neq p_2$. Then, p_1 and p_2 are *synchronous* in S if $const(A_1) \cap const(A_2) \neq \emptyset$. Otherwise, p_1 and p_2 are *asynchronous* in S. A set S is *asynchronous* if p_1 and p_2 are asynchronous in S for any different predicates p_1 and p_2 in S.

An observation O is *asynchronous* if O is an asynchronous set. Suppose that O is partitioned into n disjoint sets as (10) and H is partitioned as (11). In this case, take the *greatest specialization under implication (GSI)* [20] of clauses $lgs(H_1), \ldots, lgs(H_n)$. The GSI of any finite set of clauses exists and is computed by simply taking a disjunction as

$$gsi(lgs(H_1), \ldots, lgs(H_n)) = lgs(H_1) \vee \cdots \vee lgs(H_n). \tag{13}$$

By $lgs(H_i) \models gsi(lgs(H_1), \ldots, lgs(H_n))$ for $i = 1, \ldots, n$, the GSI (13) provides a formula which is weaker than each $lgs(H_i)$.

Step 4: Optimization
Hypotheses computed in the above two steps generally contain clauses or atoms that are useless or have no direct connection to explaining the observation O. To extract useful information, a method for optimization is provided.

Definition 2.3. (isolated) Let $term(A)$ be the set of terms appearing in an atom A. Then, two atoms A_1 and A_2 are *linked* if $term(A_1) \cap term(A_2) \neq \emptyset$. Given a clause C, an atom $A \in body(C)$ is *isolated* in C if there is no atom $A'(\neq A)$ in C such that A' and A are linked.

For any clause C in $lgs(H_i)$ $(1 \leq i \leq n)$,

1. remove any atom A from $head(C)$ such that $pred(A)$ is not included in O,
2. remove any atom A from $body(C)$ such that A is isolated in C.

The first reduction eliminates atoms in the head which do not contribute to the derivation of observations. The second reduction eliminates atoms in the body which have no connection to the observation. Let $lgs^*(H_i)$ be the result of such reduction over $lgs(H_i)$. When $B \wedge lgs(H_i)$ is consistent, the reduction is performed as far as $B \wedge lgs^*(H_i)$ is consistent. The algorithm (called Brain) for computing hypotheses is summarized in Figure 1.[4]

Now we show that Brain computes a solution for brave induction.

[4] Brain is named after BRAve INduction.

```
Procedure: BRAIN

Input : the background knowledge $B$ and an observation $O$;
Output : hypotheses $H^\wedge$ and $H^\vee$.

Step 1 : Compute ground and irredundant DNF formulas $\neg H \vee F$ from $B \wedge \neg O$, and
    extract $\neg H$ from $\neg H \vee F$.
Step 2 : Compute $lgs(H)$.
Step 3 : If $O$ is asynchronous and is partitioned into $O = O_1 \wedge \cdots \wedge O_n$,
    compute $gsi(lgs(H_1), \ldots, lgs(H_n))$.
Step 4 : If $B \wedge lgs^*(H_i)$ is consistent, put $H^\wedge = lgs^*(H_1) \wedge \cdots \wedge lgs^*(H_n)$ and
    $H^\vee = lgs^*(H_1) \vee \cdots \vee lgs^*(H_n)$.
```

Fig. 1. An algorithm for brave induction

Lemma 2.9. *Let B be the background knowledge and O an observation. Let $H^\wedge$ be a clausal theory obtained by* BRAIN*. If $B \wedge H^\wedge$ is consistent, $B \wedge H^\wedge$ has a minimal model satisfying O.*

Proof. (i) Suppose first that O contains a single predicate. Then, for each clause $C_i = \Sigma_i \leftarrow \Gamma_i$ $(1 \leq i \leq k)$ of (8), there is a ground substitution θ_i such that $lgs(H)\theta_i \subseteq C_i$ for the $lgs(H)$ of (12). Thus, $lgs(H) \models H$ and $B \wedge lgs(H) \models B \wedge H$. So $lgs(H)$ satisfies the relation (5). By $H^\wedge \models lgs(H)$, $H^\wedge$ also satisfies (5). Selecting a disjunction F of ground atoms such that $B \wedge H^\wedge \not\models F$, $H^\wedge$ satisfies the relation (6). Hence, the result holds by Proposition 2.8. (ii) Next, suppose that O contains multiple different predicates. Then, for each H_i of (11), $lgs(H_i) \models H$. This implies $lgs(H) \models H$ and $lgs(H)$ satisfies the relation (5). The rest of the proof is the same as (i). □

Lemma 2.10. *Let B be the background knowledge and O an asynchronous observation. Let $H^\vee$ be a clausal theory obtained by* BRAIN*. If $B \wedge H^\vee$ is consistent, $B \wedge H^\vee$ has a minimal model satisfying O.*

Proof. Let $O = O_1 \wedge \cdots \wedge O_n$. By Lemma 2.9, $B \wedge lgs^*(H_i)$ has a minimal model M_i satisfying O_i $(1 \leq i \leq n)$. Putting $M = \bigcup_{1 \leq i \leq n} M_i$, M satisfies O. As $B \wedge lgs^*(H_i) \models B \wedge lgs^*(H_1) \vee \cdots \vee lgs^*(H_n)$, M is a model of $B \wedge H^\vee$ satisfying O. Since the head of $H^\vee$ consists of atoms with different predicates, for any ground instance of $H^\vee$, we can select an atom $A \in M$ from the head of each clause whenever $A \in O$. Since O is an asynchronous set, two atoms A_1 and A_2 with different predicates are not selected from the same ground instance of $H^\vee$. Hence, M is a minimal model of $B \wedge H^\vee$. □

By Lemmas 2.9 and 2.10, we have the next result.

Theorem 2.11. *Any hypothesis computed by* BRAIN *becomes a solution of brave induction.*

Example 2.3. Consider the background knowledge B and the observation O:

$$\begin{aligned} B:&\ \ teacher(0) \wedge student(1) \wedge \cdots \wedge student(30),\\ O:&\ \ euro(1) \wedge \cdots \wedge euro(20) \wedge asia(21) \wedge \cdots \wedge asia(27) \wedge usa(28) \wedge \cdots \wedge usa(30). \end{aligned}$$

Brain computes candidate hypotheses as follows. First, $B \wedge \neg O$ entails the prime CNF formula $B \wedge \neg O$. From this, the ground and irredundant DNF formula $\neg H_1 \vee \neg H_2 \vee \neg H_3$ is obtained where

$$\begin{aligned} H_1 &= (\neg B \vee euro(1)) \wedge \cdots \wedge (\neg B \vee euro(20)),\\ H_2 &= (\neg B \vee asia(21)) \wedge \cdots \wedge (\neg B \vee asia(27)),\\ H_3 &= (\neg B \vee usa(28)) \wedge \cdots \wedge (\neg B \vee usa(30)). \end{aligned}$$

The LGS of each H_i becomes

$$\begin{aligned} lgs(H_1) &= \neg teacher(0) \vee \neg student(x) \vee euro(x),\\ lgs(H_2) &= \neg teacher(0) \vee \neg student(y) \vee asia(y),\\ lgs(H_3) &= \neg teacher(0) \vee \neg student(z) \vee usa(z). \end{aligned}$$

Then, $lgs(H) = lgs(H_1) \wedge lgs(H_2) \wedge lgs(H_3)$. On the other hand, as O is asynchronous, the greatest specialization becomes $gsi(lgs(H_1), \ldots, lgs(H_n)) = lgs(H_1) \vee lgs(H_2) \vee lgs(H_3)$.

Finally, the atom $teacher(0)$ is isolated in each $lgs(H_i)$ $(i = 1, 2, 3)$, so that it is remove from the body of each clause. As a result, $H^{\wedge}$ becomes

$$(euro(x) \leftarrow student(x)) \ \wedge\ (asia(x) \leftarrow student(x)) \ \wedge\ (usa(x) \leftarrow student(x)),$$

and $H^{\vee}$ becomes

$$euro(x) \vee asia(x) \vee usa(x) \leftarrow student(x).$$

Thus, $H^{\wedge}$ and $H^{\vee}$ become two solutions of brave induction.

In Example 2.3, $H^{\wedge}$ also becomes a solution of cautious induction, but $H^{\vee}$ is a solution inherent to brave induction.

3 Brave Induction in Nonmonotonic Logic Programming

As presented in Section 2, brave induction is useful for learning theories with indefinite or incomplete information. Incomplete information is also represented as *default* rule in logic programming. In this section, we consider brave induction in *nonmonotonic logic programs*.

3.1 Answer Set Programming

Answer set programming (ASP) [16] represents incomplete knowledge in a logic program and realizes nonmonotonic default reasoning. In ASP a logic program is described by an *extended disjunctive program* (EDP). An EDP (or simply a *program*) is a set of rules of the form:

$$L_1\,;\,\cdots\,;\,L_l \;\leftarrow\; L_{l+1}\,,\ldots,\,L_m,\,\mathit{not}\,L_{m+1}\,,\ldots,\,\mathit{not}\,L_n$$

$(n \geq m \geq l \geq 0)$ where each L_i is a positive/negative literal, i.e., A or $\neg A$ for an atom A. *not* represents *default negation* or *negation as failure* (NAF). $\mathit{not}\,L$ is called an *NAF-literal.* Literals and NAF-literals are called *LP-literals.* The symbol ";" represents disjunction and "," represents conjunction. The above rule is read "If all $L_{l+1},\ldots,L_m$ are believed and all $L_{m+1},\ldots,L_n$ are disbelieved, then some of $L_1,\ldots,L_l$ is believed". The left-hand side of the rule is the *head*, and the right-hand side is the *body.* For each rule r of the above form, $\mathit{head}(r)$, $\mathit{body}^+(r)$ and $\mathit{body}^-(r)$ denote the sets of literals $\{L_1,\ldots,L_l\}$, $\{L_{l+1},\ldots,L_m\}$, and $\{L_{m+1},\ldots,L_n\}$, respectively. Also, $\mathit{not\_body}^-(r)$ denotes the set of NAF-literals $\{\mathit{not}\,L_{m+1},\ldots,\mathit{not}\,L_n\}$. A disjunction of literals and a conjunction of (NAF-)literals in a rule are identified with its corresponding sets of literals. A rule r is often written as $\mathit{head}(r) \leftarrow \mathit{body}^+(r), \mathit{not\_body}^-(r)$ or $\mathit{head}(r) \leftarrow \mathit{body}(r)$ where $\mathit{body}(r) = \mathit{body}^+(r) \cup \mathit{not\_body}^-(r)$. A rule r is *disjunctive* if $\mathit{head}(r)$ contains more than one literal. A rule r is a *constraint* if $\mathit{head}(r) = \emptyset$; and r is a *fact* if $\mathit{body}(r) = \emptyset$. A program is *NAF-free* if no rule contains NAF-literals. A program, rule, or literal is *ground* if it contains no variable. A program P with variables is a shorthand of its *ground instantiation* $\mathit{Ground}(P)$, the set of ground rules obtained from P by substituting variables in P by elements of its Herbrand universe in every possible way. Two literals L_1 and L_2 have the same *sign* if both L_1 and L_2 are positive literals (or negative literals). A set S of ground literal is *consistent* if $L \in S$ implies $\neg L \notin S$ for any literal L; otherwise, S is *inconsistent.* Let L_0 be a ground literal and S a set of ground literals. Then, $L_1 \in S$ is *relevant* to L_0 if either (i) $\mathit{const}(L_0) \cap \mathit{const}(L_1) \neq \emptyset$, or (ii) for some literal $L_2 \in S$, $\mathit{const}(L_1) \cap \mathit{const}(L_2) \neq \emptyset$ and L_2 is relevant to L_0. Otherwise, $L_1 \in S$ is *irrelevant* to L_0.

The semantics of an EDP is defined by the *answer set semantics.* Let Lit be the set of all ground literals in the language of a program. Suppose a program P and a set of literals $S(\subseteq \mathit{Lit})$. Then, the *reduct* P^S is the program which contains the ground rule $\mathit{head}(r) \leftarrow \mathit{body}^+(r)$ iff there is a rule r in $\mathit{Ground}(P)$ such that $\mathit{body}^-(r) \cap S = \emptyset$. Given an NAF-free EDP P, let S be a set of ground literals that is (i) *closed* under P, i.e., for every ground rule r in $\mathit{Ground}(P)$, $\mathit{body}(r) \subseteq S$ implies $\mathit{head}(r) \cap S \neq \emptyset$; and (ii) *logically closed*, i.e., it is either consistent or equal to Lit. Given an EDP P and a set S of literals, S is an *answer set* of P if S is an answer set of P^S. A program has none, one, or multiple answer sets in general. The set of all answer sets of P is written as $\mathit{AS}(P)$. An answer set is *consistent* if it is not Lit. A program P is *consistent* if it has a consistent answer set; otherwise, P is *inconsistent.*

Example 3.1. The program:

$$\begin{aligned}&\mathit{tea}\,;\,\mathit{coffee} \leftarrow,\\&\mathit{milk} \leftarrow \mathit{tea},\ \mathit{not}\ \mathit{lemon},\end{aligned}$$

$$
\begin{aligned}
&lemon \leftarrow tea,\ not\ milk,\\
&milk \leftarrow coffee
\end{aligned}
$$

has the three answer sets: $S_1 = \{\, tea, milk \,\}$, $S_2 = \{\, tea, lemon \,\}$, and $S_3 = \{\, coffee, milk \,\}$, which represent possible options for drink.

3.2 Brave Induction in ASP

In this section, we consider the following problem setting:

- the background knowledge B is given as a consistent EDP,
- an observation O is given as a consistent set of ground literals,
- a hypothesis H is a consistent set of rules.

Then, brave induction in ASP is defined as follows.

Definition 3.1. (brave induction in ASP) Let B be the background knowledge and O an observation. A hypothesis H *covers* O under B in *brave induction* if $B \cup H$ has a consistent answer set S such that $O \subseteq S$.[5]

Cautious induction, by contrast, requests that $O \subseteq S$ holds for *every* consistent answer set S of $B \cup H$. Brave induction in ASP has properties similar to those of clausal theories. As the case of clausal theories, the problem of our interest is the case when B has no answer set including O.

In case of brave induction from clausal theories, inverse entailment is used for computing hypotheses. However, it is known that inverse entailment in classical logic is not applied to nonmonotonic logic programs [23]. We then consider another method for computing possible hypotheses.

Step 1: Computing ground hypotheses
Given an observation O, let $\Theta = \{\, L \mid L \in Lit$ and $pred(L)$ appears in $O \,\}$. Suppose that the background knowledge B has an answer set S. Then, construct a finite and consistent set R of ground rules satisfying the following conditions. For any rule $r \in R$,

1. $head(r) = \{L\}$ for any $L \in O$,
2. $body^+(r) = \{\, L \mid L \in S$ and L is relevant to the literal in $head(r) \,\}$.
3. $body^-(r) = \{\, L \mid L \in Lit \setminus (S \cup \Theta)$ and L is relevant to the literal in $head(r)$ and appears in $Ground(P) \,\}$.

The third condition requires that no rule contains default negation of literals in $S \cup \Theta$. The reason is that if $body^-(r)$ contains literals from S, $body(r)$ may contain both L in $body^+(r)$ and $not\, L$ in $body^-(r)$, which makes the rule meaningless. Also, if $body^-(r)$ contains literals from Θ, r may contain a *negative loop* that

[5] In nonmonotonic logic programming, logical connectives in classical logic are not used. So we write $B \cup H$ instead of $B \wedge H$.

would make a program inconsistent. By its construction, different hypotheses are constructed by different answer sets in general.

Step 2: Generalization
The notion of LGS is extended to rules containing default negation. It is done by syntactically viewing rules as "clauses". That is, identify disjunction ";" with the classical one "$\vee$, and any NAF-literal "$not\, p(t_1, \ldots, t_n)$" with a new atom "$not\_p(t_1, \ldots, t_n)$" with the predicate "$not\_p$". $\neg p$ is also considered a predicate "$\neg\_p$" and is considered a predicate different from p. With this setting, the LGS of a finite set of rules is defined in the same manner as the one in clausal theories [24]. The generalization phase is similar to the case of clausal theories. If O contains a single predicate, compute $lgs(R)$. Else if O contains multiple different predicates, O is partitioned into disjoint subsets $O = O_1 \cup \cdots \cup O_n$ where O_i $(1 \leq i \leq n)$ is a set of ground literals having the same predicate. Correspondingly, R is partitioned as $R = R_1 \cup \cdots \cup R_n$ where R_i $(1 \leq i \leq n)$ is a set of ground rules whose heads have the predicate in O_i. The LGS of each R_i is then computed and collected as $lgs(R) = lgs(R_1) \cup \cdots \cup lgs(R_n)$.

Step 3: Constructing a weak form of hypotheses
The GSI of two rules $Head_1 \leftarrow Body_1$ and $Head_2 \leftarrow Body_2$ is also defined as $Head_1\, ;\, Head_2 \leftarrow Body_1,\, Body_2$, by taking the disjunction/conjunction of two heads/bodies. When an observation O is an asynchronous set, the GSI of $lgs(R_1), \ldots, lgs(R_n)$ is constructed as

$$\begin{aligned} & gsi(lgs(R_1), \ldots, lgs(R_n)) \\ & = head(lgs(R_1))\, ;\, \cdots\, ;\, head(lgs(R_n)) \leftarrow body(lgs(R_1)),\, \ldots,\, body(lgs(R_n))\,. \end{aligned}$$

Step 4: Optimization
The notion of "isolated literal" in a rule is defined by replacing a clause with a rule, and an atom with a literal in Definition 2.3. Then, for any rule r in $lgs(R_i)$ $(1 \leq i \leq n)$, remove any literal L from $body(r)$ such that L is isolated in r. Let $lgs^*(R_i)$ be the result of such reduction over $lgs(R_i)$. When $B \cup lgs(R_i)$ is consistent, the reduction is performed as far as $B \cup lgs^*(R_i)$ is consistent.

The algorithm of brave induction in ASP (called Brain^{not}) is sketched in Figure 2. In what follows, we show that Brain^{not} computes hypotheses for brave induction in ASP. We say that O is *independent* of B if every predicate in O appears nowhere in B.

Proposition 3.1. *Let P be a consistent program. Suppose a consistent set R of rules such that for any $r \in R$, every predicate in $head(r)$ appears nowhere in P. Then, $P \cup R$ is consistent.*

Lemma 3.2. *Let B be the background knowledge and O an observation. Let $H^\wedge$ be a set of rules obtained by* Brain^{not}. *If O is independent of B, $B \cup H^\wedge$ has an answer set U such that $O \subseteq U$.*

Proof. Let S be an answer set of B. For any rule r in R, $head(r) \leftarrow body^+(r)$ is in R^S. Here, $body^+(r) \subseteq S$, $head(r) = \{L\}$, and $pred(L)$ appears nowhere in

```
Procedure: BRAIN$^{not}$

Input  : the background knowledge $B$ and an observation $O$;
Output : hypotheses $H^{\wedge}$ and $H^{\vee}$.

Step 1 : Select an answer set $S$ of $B$ and construct a set $R$ of rules.
Step 2 : Compute $lgs(R)$.
Step 3 : If $O$ is asynchronous and is partitioned into $O = O_1 \cup \cdots \cup O_n$,
    compute $gsi(lgs(R_1), \ldots, lgs(R_n))$.
Step 4 : If $B \cup lgs^*(R_i)$ is consistent, put $H^{\wedge} = lgs^*(R_1) \cup \cdots \cup lgs^*(R_n)$ and $H^{\vee} =$
    $\{\, head(lgs^*(R_1))\,;\, \cdots\,;\, head(lgs^*(R_n)) \leftarrow body(lgs^*(R_1)), \ldots, body(lgs^*(R_n)) \,\}$.
```

Fig. 2. An algorithm for brave induction in ASP

B. Put $T = S \cup \{\, L \mid L \in head(r) \text{ and } r \in R^S \,\}$. By $B^T \cup R^T = B^S \cup R^T$, T becomes a minimal closed set of $B^T \cup R^T = (B \cup R)^T$. Since O is independent of B, every predicate in $head(r)$ appears nowhere in P and $B \cup R$ is consistent (Proposition 3.1). As R contains rules having every literal in O, T is a consistent answer set of $B \cup R$ such that $O \subseteq T$. Next, we show that $B \cup lgs(R)$ has an answer set such that $O \subseteq U$. Let $R = R_1 \cup \cdots \cup R_n$. By the definition, $lgs(R_i)\theta \subseteq r$ for any $r \in R_i$ $(1 \leq i \leq n)$ with some ground substitution θ. Then, for any rule $r \in R_i$, $body^-(r) \cap S = \emptyset$ implies $body^-(lgs(R_i)\theta) \cap S = \emptyset$. So $B^S \cup R^S \subseteq B^S \cup lgs(R)^S$. Since O is independent of B, $lgs(R)^S \setminus R^S$ is a set of NAF-free rules whose heads have predicates appearing nowhere in B. Put $V = \{\, L \mid r \in lgs(R)^S \setminus R^S,\ head(r) = \{L\} \text{ and } body^+(r) \subseteq S \,\}$. Then, $B^{S \cup V} \cup lgs(R)^{S \cup V} = B^S \cup lgs(R)^S$ has a minimal closed set $U = S \cup V$. Since O is independent of B, $B^S \cup lgs(R)^S$ is consistent (Proposition 3.1). As $lgs(R)$ contains rules having every literal in O, U is a consistent answer set of $B \cup lgs(R)$ such that $O \subseteq U$. When $B \cup lgs^*(R_i)$ is consistent, U also becomes a consistent answer set of $B \cup H^{\wedge}$. Hence, the result follows. □

Lemma 3.3. *Let B be the background knowledge and $O = O_1 \cup \cdots \cup O_n$ an asynchronous observation. Let $H^{\vee}$ be a set of rules obtained by* BRAIN^{not}*. If O is independent of B, $B \cup H^{\vee}$ has an answer set U such that $O \subseteq U$.*

Proof. Let $r = gsi(lgs(R_1), \ldots, lgs(R_n))$. For some answer set S of B, $body^+(r\theta) \subseteq S$ and $body^-(r\theta) \subseteq Lit \setminus (S \cup \Theta)$ hold for any ground instance $r\theta$ of r. Since $head(r)$ consists of literals with different predicates, for any ground instance of $r\theta$, a set T of literals is constructed in a way that a literal $L \in T$ is selected from the head of each $r\theta$ whenever $L \in O$. Since O is an asynchronous set, two literals L_1 and L_2 with different predicates are not selected from the same ground instance of $gsi(lgs(R_1), \ldots, lgs(R_n))$. In this way, we can construct a minimal closed set $U = S \cup T$ of $B^U \cup \{r\}^U$. Since O is independent of B, $B^U \cup \{r\}^U$ is consistent (Proposition 3.1). Then, U becomes a consistent answer set of $B \cup \{\, gsi(lgs(R_1), \ldots, lgs(R_n)) \,\}$ and $O \subseteq U$. When $B \cup lgs^*(R_i)$ is consistent, U also becomes a consistent answer set of $B \cup H^{\vee}$. □

By Lemmas 3.2 and 3.3, we have the next result.

Theorem 3.4. *Any hypothesis computed by* BRAIN^{not} *becomes a solution of brave induction.*

Example 3.2. There are two couples, Adam and Nancy, and Bob and Jane. They plan to go to either sea or mountain on this weekend. Each couple can select one of them, but a husband and a wife go to the same place. The situation is represented as the background knowledge B:

$$\begin{aligned}
&s(x) \leftarrow not\ m(x),\\
&m(x) \leftarrow not\ s(x),\\
&c(a,n) \leftarrow, \quad c(b,j) \leftarrow,\\
&\leftarrow c(x,y), s(x), m(y),\\
&\leftarrow c(x,y), s(y), m(x)
\end{aligned}$$

where the predicates s, m and c mean *sea*, *mountain* and *couple*, respectively, and the constants a, n, b and j mean Adam, Nancy, Bob and Jane, respectively. B has four answer sets: $S_1 = \{ c(a,n),\ c(b,j),\ s(a),\ s(n),\ s(b),\ s(j) \}$, $S_2 = \{ c(a,n),\ c(b,j),\ s(a),\ s(n),\ m(b),\ m(j) \}$, $S_3 = \{ c(a,n),\ c(b,j),\ m(a),\ m(n),\ s(b),\ s(j) \}$, and $S_4 = \{ c(a,n),\ c(b,j),\ m(a),\ m(n),\ m(b),\ m(j) \}$.

Suppose the observation that Adam and Nancy are tanned, but Bob and Jane are not. It is represented as:

$$O = \{ t(a),\ t(n),\ \neg t(b),\ \neg t(j) \}$$

where the predicate t mean *tanned*.

BRAIN^{not} constructs candidate hypotheses as follows. First, an answer set of B, for instance S_2, is selected. A set R of rules is then constructed as:

$$\begin{aligned}
&t(a) \leftarrow c(a,n),\ s(a),\ s(n),\ not\ m(a),\ not\ m(n),\\
&t(n) \leftarrow c(a,n),\ s(a),\ s(n),\ not\ m(a),\ not\ m(n),\\
&\neg t(b) \leftarrow c(b,j),\ m(b),\ m(j),\ not\ s(b),\ not\ s(j),\\
&\neg t(j) \leftarrow c(b,j),\ m(b),\ m(j),\ not\ s(b),\ not\ s(j).
\end{aligned}$$

Next, the $lgs(R)$ is constructed as

$$\begin{aligned}
&t(x) \leftarrow c(a,n),\ s(x),\ not\ m(x),\\
&\neg t(y) \leftarrow c(b,j),\ m(y),\ not\ s(y).
\end{aligned}$$

Since O is an asynchronous set, $gsi(lgs(R_1), \ldots, lgs(R_n))$ is also constructed as

$$t(x)\,;\,\neg\, t(y) \leftarrow c(a,n),\ c(b,j),\ s(x),\ m(y),\ not\ m(x),\ not\ s(y).$$

Finally, isolated literals $c(a,n)$ and $c(b,j)$ are removed, and $H^{\wedge}$ and $H^{\vee}$ become $H^{\wedge} = \{ t(x) \leftarrow s(x),\ not\ m(x), \quad \neg t(y) \leftarrow m(y),\ not\ s(y) \}$ and $H^{\vee} = \{ t(x)\,;\,\neg\, t(y) \leftarrow s(x),\ m(y),\ not\ m(x),\ not\ s(y) \}$, respectively.

4 Discussion

There are some induction frameworks which relax explanatory induction in different ways. De Raedt and Dehaspe [3,4] introduce the framework of *learning from satisfiability* (LFS). Given the background knowledge B and an observation O, a hypothesis H covers O under B in LFS if $B \wedge H \wedge O$ is consistent. In other words, H covers O under B in LFS if $B \wedge H$ has a model satisfying O. By the definition, LFS is weaker than brave induction. That is, if a hypothesis H covers O under B in brave induction, H covers O under B in LFS (cf. Proposition 2.3). The converse implication does not hold in general. Compared with brave induction, LFS does not require the minimality of models. So any theory H becomes a solution as far as it is consistent with $B \wedge O$. This implies that any hypothesis which has no connection to $B \wedge O$ may become a solution. For instance, let $B = \{\, p(a) \,\}$ and $O = \{\, q(a) \,\}$. Then, $H_1 = \{\, r(b) \,\}$, $H_2 = \{\, s(x) \leftarrow r(x) \,\}$, $H_3 = \{\, \neg\, s(c) \,\}, \ldots$ are all solutions in LFS. Note that none of H_1, H_2, and H_3 becomes a solution of brave induction. As seen in the above example, LFS appears too weak for building useful hypotheses. Since it generally produces tons of useless hypotheses, additional conditions must be introduced to reduce the hypotheses space for practical usage. Brave induction is considered as a restricted version of LFS, that is, we imposed the condition of *minimality* on models of $B \wedge H$ satisfying O. Moreover, in [3] the authors say:

> One open question for further research is how learning from satisfiability (which employs a monotonic logic) could be used for inducing nonmonotonic logic programs.

This paper provides a solution in the context of brave induction.

Confirmatory induction or *descriptive induction* [15] provides a different method for induction. Given the background knowledge B and an observation O such that $B \wedge O$ is consistent, a hypothesis H *covers* E under B in confirmatory induction if $Comp(B \wedge O) \models H$ where $Comp$ represents Clark's predicate completion. When B is a set of definite clauses, any hypothesis H induced in explanatory induction and the *closed world assumption* becomes a solution of confirmatory induction [1]. For instance, given $B = \{\, human(Socrates) \,\}$ and $O = \{\, mortal(Socrates) \,\}$, both $H_1 = (mortal(x) \leftarrow human(x))$ and $H_2 = (human(x) \leftarrow mortal(x))$ become solutions of confirmatory induction, but only H_1 is the solution of explanatory induction and brave induction. On the other hand, explanatory induction and brave induction are not always stronger than confirmatory induction. For instance, let $B = \{\, p(x) \leftarrow q(x) \,\}$ and $O = p(a)$. Then, $H = q(a) \wedge q(b)$ becomes a solution of explanatory induction and brave induction, while H is not a solution of confirmatory induction. Thus, there is no relation between brave induction and confirmatory induction in general. Generally speaking, confirmatory induction does not explain why particular individuals are observed under the background knowledge, and the aim is to learn relationships between any of the concepts [8]. For induction in full clausal theories, Inoue [13] introduces *CF-induction* which extends Muggleton's inverse entailment to full clausal theories. However, CF-induction is cautious induction and is often too strong for learning indefinite theories. Induction in answer set programming is introduced by Sakama [25], but it is also cautious induction.

Brave induction guarantees the existence of a minimal model of $B \wedge H$ in which an observation O is satisfied. In this case, H covers the *positive* observation O under B. In ILP, on the other hand, *negative* observations as well as positive ones are also handled. Given a negative observation N, it is required that H *uncovers* N under B. This condition is logically represented as $B \wedge H \not\models N$. Definition 2.1 is extended to handle negative observations as follows.

Definition 4.1. Let B be the background knowledge, P a positive observation, and N a negative observation. A hypothesis H is a solution of brave induction if $B \wedge H$ has a minimal model M such that $M \models P$ and $M \not\models N$.

By putting $O = P \wedge \neg N$, the above definition reduces to Definition 2.1 and negative observations are handled within the framework of this paper.[6]

In this paper, we introduced induction algorithms which produce clauses or rules that define more than one predicate. The problem is known as *multiple predicate learning* (MPL) [2]. In MPL the order of learning different clauses affects the results of the learning tasks and even the existence of solutions, especially in the presence of negative observations [1]. We do not discuss details of the problem in this paper, and just impose the condition of consistency on $B \wedge lgs^*(H_i)$ in the first-order case. In case of ASP, independence of O with respect to B guarantees the consistency of $B \cup \{\, lgs^*(R)\,\}$ as far as B is consistent. Further discussion for MPL is left for future study.

We finally remark the computational complexity issue. First, in case of clausal theories (CT), brave induction has a solution H iff $B \wedge O$ is consistent (Proposition 2.1). Then, given a ground clausal theory B and a ground observation O, deciding the existence of solutions in brave induction is NP-complete. In case of ASP, brave induction has a solution H if $B \cup O$ has a consistent answer set. The decision problem is Σ_2^P-complete [6]. Next, we consider the task of identifying whether a theory H is a solution of brave induction. To this end, we consider a complementary problem: a ground clausal theory $B \wedge H$ has no minimal model satisfying a conjunction O of ground atoms. This is a task of the *extended GCWA* and is known Π_2^P-complete [6], so that the identification problem is Σ_2^P-complete. Deciding whether the ground program $B \cup H$ has a consistent answer set satisfying O is also Σ_2^P-complete [6]. On the other hand, in cautious induction, the decision problem for the existence of solutions has the same complexity in both CT and ASP. The decision problem for the identification of solutions is coNP-complete in CT, and Π_2^P-complete in ASP. Comparing those results, brave induction appears more expensive than cautious induction for identifying solutions in CT.

Brave and cautious inferences are widely used for commonsense reasoning from incomplete knowledge. In hypothetical reasoning, two different types of *abduction* under brave and cautious inferences are introduced by [7,11] under

[6] Strictly speaking, $\neg N$ requires *Skolemization* when a clausal theory N contains variables. For detailed technique, see [13].

the *stable model semantics* of logic programs. To the best of our knowledge, however, no studies introduce brave induction as a form of learning or learning from incomplete information. Since abduction and induction are both hypothetical reasoning which extend the background knowledge to explain observations, brave induction proposed in this paper has a right place and serves as a natural extension of brave abduction.

5 Conclusion

This paper introduced the framework of brave induction which is weaker than explanatory induction usually used in ILP. We developed an algorithm for computing brave induction in full clausal theories, and also extended the framework to induction in answer set programming. As argued in the paper, explanatory induction is often too strong for learning indefinite or incomplete theories. Learning from satisfiability, on the other hand, appears too weak as presented in Section 4. Brave induction has a position between learning from satisfiability and explanatory induction, and provides a moderate solution in the middle.

We are now seeking practical applications of brave induction. One of the candidates is *systems biology* which would have indefinite or incomplete information in the background knowledge and observations. A recent study [5] shows that an ILP approach is useful for finding causal relations between concentration changes of metabolities and enzyme activities. In [5] CF-induction [13] is used for learning hypotheses. Since CF-induction is cautious induction, there would be a room for brave induction to find new hidden hypotheses. A theoretical extension to *circumscriptive induction* [14] is also a topic for future research.

References

1. De Raedt, L., Lavrač, N.: The many faces of inductive logic programming. In: Komorowski, J., Raś, Z.W. (eds.) ISMIS 1993. LNCS, vol. 689, pp. 435–449. Springer, Heidelberg (1993)
2. De Raedt, L., Lavrač, N.: Multiple predicate learning in two inductive logic programming setting. Journal of the IGPL 4(2), 227–254 (1996)
3. De Raedt, L.: Logical settings for concept-learning. Artificial Intelligence 95, 187–201 (1997)
4. De Raedt, L., Dehaspe, L.: Learning from satisfiability. In: Proceedings of the 9th Dutch Conference on Artificial Intelligence, pp. 303–312 (1997)
5. Doncescu, A., Yamamoto, Y., Inoue, K.: Biological systems analysis using inductive logic programming. In: Proceedings of the 21st International Conference on Advanced Information Networking and Applications, pp. 690–695. IEEE Computer Society, Los Alamitos (2007)
6. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: propositional case. Annals of Mathematics and Artificial Intelligence 15, 289–323 (1995)
7. Eiter, T., Gottlob, G., Leone, N.: Abduction from logic programs: semantics and complexity. Theoretical Computer Science 189, 129–177 (1997)

8. Flach, P.A., Kakas, A.C.: Abductive and inductive reasoning: background and issues. In: Flach, P.A., Kakas, A.C. (eds.) Abduction and Induction — Essays on their Relation and Integration. Kluwer Academic, Dordrecht (2000)
9. Gelfond, M., Przymusinska, H., Przymusinski, T.: On the relationship between circumscription and negation as failure. Artificial Intelligence 38, 75–94 (1989)
10. Inoue, K.: Linear resolution for consequence finding. Artificial Intelligence 56, 301–353 (1992)
11. Inoue, K., Sakama, C.: A fixpoint characterization of abductive logic programs. Journal of Logic Programming 27(2), 107–136 (1996)
12. Inoue, K.: Automated abduction. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS (LNAI), vol. 2408, pp. 311–341. Springer, Heidelberg (2002)
13. Inoue, K.: Induction as consequence finding. Machine Learning 55, 109–135 (2004)
14. Inoue, K., Saito, H.: Circumscription policies for induction. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 164–179. Springer, Heidelberg (2004)
15. Lachiche, N.: Abduction and induction from a non-monotonic reasoning perspective. In: Flach, P.A., Kakas, A.C. (eds.) Abduction and Induction — Essays on their Relation and Integration. Kluwer Academic, Dordrecht (2000)
16. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence 138, 39–54 (2002)
17. Minker, J.: On indefinite data bases and the closed world assumption. In: Loveland, D.W. (ed.) CADE 1982. LNCS, vol. 138, pp. 292–308. Springer, Heidelberg (1982)
18. McDermott, D.: Nonmonotonic logic II: nonmonotonic modal theories. Journal of the ACM 29, 33–57 (1982)
19. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
20. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. LNCS (LNAI), vol. 1228. Springer, Heidelberg (1997)
21. Plotkin, G.D.: A note on inductive generalization. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 153–163. Edinburgh University Press (1970)
22. Reiter, R.: On closed world databases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 55–76. Plenum, New York (1978)
23. Sakama, C.: Inverse entailment in nonmonotonic logic programs. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 209–224. Springer, Heidelberg (2000)
24. Sakama, C.: Nonmonotonic inductive logic programming. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 62–80. Springer, Heidelberg (2001)
25. Sakama, C.: Induction from answer sets in nonmonotonic logic programs. ACM Transactions on Computational Logic 6(2), 203–231 (2005)

A Statistical Approach to Incremental Induction of First-Order Hierarchical Knowledge Bases

David J. Stracuzzi[1] and Tolga Könik[2]

[1] School of Computing and Informatics,
Arizona State University, Tempe, AZ 85287-8809, USA
david.stracuzzi@asu.edu

[2] Center for the Study of Language and Information
Stanford University, Stanford, CA 94305, USA
konik@stanford.edu

Abstract. Knowledge bases play an important role in many forms of artificial intelligence research. A simple approach to producing such knowledge is as a database of ground literals. However, this method is neither compact nor computationally tractable for learning or performance systems to use. In this paper, we present a statistical method for incremental learning of a hierarchically structured, first-order knowledge base. Our approach uses both rules and ground facts to construct succinct rules that generalize the ground literals. We demonstrate that our approach is computationally efficient and scales well to domains with many relations.

1 Introduction

Artificial intelligence researchers have long used first-order knowledge bases, which are collections of rules and facts, to support their systems and technology. Some systems use them as background to support decision making, while others perform theory revision on the knowledge base itself in an attempt to improve rule coverage. In this paper, we consider the problem of inducing a hierarchically structured, first-order knowledge base from a sequence of ground literals.

ILP systems are often characterized along four dimensions [1]: 1) batch versus incremental input, 2) interactive versus non-interactive learners, 3) theory revision versus rule induction systems, and 4) single versus multiple concept learners. Although these factors are largely independent, most ILP systems fall into one of two categories [2]. The first, known as empirical systems, are batch input, non-interactive, single concept learners that build their own concept definitions. The second category, known as interactive systems, take incremental input and user interaction to revise multiple concepts in existing theories.

In this paper we present SCALE, an incremental, non-interactive system that constructs new definitions for multiple concepts. This combination allows SCALE to operate in online environments. For example, suppose a robot explores its environment in an effort to induce rules that govern the relationships among perceived objects. Such a robot may perceive examples of many relationships

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 279–296, 2008.

(concepts) over time, but lack the ability to seek out specific examples or request counter-examples to induced rules. Scale would let the robot construct an intensional knowledge base to describe its environment. The robot can then apply this knowledge to infer relationships among previously unseen object, and use the results to determine which actions to take next.

2 Problem Formulation and Knowledge Representation

We formulate the knowledge base induction problem as follows. The system receives background knowledge in the form of many extensionally defined concepts given as ground literals, or their negations. The system may also receive intensionally defined concepts. The learning task is to construct an intensional definition for each extensionally defined concept that entails all of the positive ground literals and none of the negative literals.

Specifically, scale learns universally quantified definitions of the form

$$p(X,Y,Z) \overset{\theta}{\leftarrow} q_1(X,Y),\ q_2(Y,Z),\ \ldots,\ q_n(X,Z)$$

where each q_i has an associated weight w_i and $\overset{\theta}{\leftarrow}$ denotes *threshold-implication*. Threshold-implication defines the consequent to be true if the linear threshold relation $\sum_{i=1}^{n} w_i q_i \geq \theta$ is satisfied, where $q_i = 1$ if $q_i(\ldots)$ evaluates to true, and $q_i = -1$ otherwise. Thus, the classification function implemented by threshold-implication is equivalent to that of the perceptron [3], which is the simplest type of feedforward neural network. This work therefore claims both a connectionist and an inductive logic programming heritage.

Consider the following illustration of threshold-implication. Suppose we have the chess rule shown on line 1 of Table 1 for determining whether a move from one cell to another constitutes a legal move for a knight. Line 2 shows the expansion of this rule into a standard implication rule. Recall that predicate groundings that evaluate to *true* take the numeric value +1, while those that evaluate to *false* take the numeric value −1. If we bind $X \equiv \mathrm{BN} - \mathrm{B7}$ (black knight at B7) and $Y \equiv E - C5$ (empty cell at C5), then we get that $\mathit{knight}(\mathrm{BN} - \mathrm{B7}) \equiv +1$, $\mathit{Lshape}(\mathrm{BN} - \mathrm{B7}, \mathrm{E} - \mathrm{C5}) \equiv +1$, and $\mathit{canOccupy}(\mathrm{BN} - \mathrm{B7}, \mathrm{E} - \mathrm{C5}) \equiv +1$. Assuming that $w_1 = w_2 = w_3 = 1$ and $\theta = 3$, we can conclude that *knightMove*(BN-B7, E-C5) $\equiv +1$ as shown on lines 3 and 4 of Table 1.

Notice that the definitions learned by scale can represent more complex relationships than horn clauses. Threshold-implication can represent in a single

Table 1. Illustration of a threshold-implication rule evaluation

1	$\mathit{knightMove}(X,Y) \overset{\theta}{\leftarrow} \mathit{knight}(X), \mathit{Lshape}(X,Y), \mathit{canOccupy}(X,Y)$
2	$\mathit{knightMove}(X,Y) \leftarrow (w_1 \mathit{knight}(X) + w_2 \mathit{Lshape}(X,Y) + w_3 \mathit{canOccupy}(X,Y) \geq \theta)$
3	$\mathit{knightMove}(\mathrm{BN} - \mathrm{B7}, \mathrm{E} - \mathrm{C5}) \leftarrow (1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 \geq 3)$
4	$\mathit{knightMove}(\mathrm{BN} - \mathrm{B7}, \mathrm{E} - \mathrm{C5}) \leftarrow \mathit{true} \equiv +1$

clause any set of n-dimensional positive and negative ground literals that are separable by an $(n-1)$-dimensional hyperplane, where n indicates the number of antecedents as above. Literals with this property are often called *linearly separable* in the connectionist literature. This includes conjunctions, disjunctions, and other many simple functions such as $q_1 \wedge (q_2 \vee q_3)$.

Learned predicate definitions may use any intensionally defined predicates or ground facts (predicates for which no definition will be learned) from background knowledge as an antecedent, including predicates for which a definition was previously learned. Thus, the knowledge base is constructed in a bottom-up manner. Predicates for which no definition has yet been learned are not eligible to act as antecedents. As a result, Scale does not currently learn recursive rules. This is not a fundamental limitation of the algorithm, however, and we revisit the topic in our discussion of future work.

Scale's underlying numeric representation also provides a natural way to deal with numeric data. Combining constants that represent numbers or vectors with threshold-implication lets rules use constants as antecedents. Here, each constant or vector entry receives its own weight. Then, threshold-implication uses the numbers represented by the constant during evaluation in place of the usual ±1 values provided by predicate antecedents.

For example, in the chess experiments reported later, each constant represents a cell on the board using three integers. The first represents the piece occupying the cell with a range of 0–12, such that the cell may be empty (0), contain a white king (1), a black queen (8), and so on. The second and third integers have range 0–7 and represent the rank (row) and file (column) of the cell. Thus, the constant WK-A8 (white queen at A8) represents the vector <2, 0, 7>. Scale can then represent rules such as $blackPiece(X) \overset{\theta}{\leftarrow} X$, which expands to $blackPiece(X) \leftarrow (w_1X_1 + w_2X_2 + w_3X_3 \geq \theta)$, where X_i represents the i^{th} value of the vector bound to variable X. Setting $w_1 = 1$, $w_2 = 0$, $w_3 = 0$, and $\theta = 7$ then yields the desired rule.

3 Incremental Knowledge Base Induction

Scale relies on two key ideas to induce threshold-implication rules from a database of ground literals. First is that a continuous search space simplifies the first-order incremental rule induction problem. Scale's numeric representation lets the system make smaller changes for each observed ground literal than possible with a strictly logical representation. The second key idea is that the knowledge base must be organized with respect to the concept learning tasks. Hierarchical rule organization helps to reduce the learning complexity by reducing the number of candidate antecedents evaluated by the concept learning algorithm, resulting in a smaller search. Learning in scale may therefore be viewed as an incremental, bottom-up refinement of representation.

We begin our presentation of scale by formalizing the algorithmic data structures used to represent predicates, candidate hypotheses, and ground literals. We

Table 2. Symbols used in describing the SCALE algorithm

$P = (p, V_p, C_p, \boldsymbol{I_p}, \boldsymbol{w}_p, \theta)$ be a rule such that

- p is the consequent of P
- V_p is the set of arguments for p,
- C_p is the set of candidate antecedents,
- $\boldsymbol{I_p}$ is the vector of selected predicate antecedents,
- $\boldsymbol{w}_p$ is the weight vector, and
- θ is the threshold value.

$X = (p, B, d)$ be a ground literal such that

- p identifies the predicate (same as rule consequent),
- B is a set of bindings of V_p constants, and
- d is the Boolean value of p given B (-1 if negated literal, 1 otherwise).

then provide an overview of the main algorithm in terms of its four primary components. Finally, we consider the details of each component, and illustrate them with examples.

3.1 The SCALE Algorithm

Scale acquires each predicate definition through a combination of four component methods. Each component makes incremental revisions to the predicate definition at different levels of resolution. We begin our description by defining several symbols, shown in Table 2, used throughout the following discussion. These symbols identify specific implementational aspects of predicates and ground literals used by the algorithm, as shown in Table 3.

The first component acts at the broadest level, performing bottom-up evaluation (line 8) over all intensionally defined predicates in the knowledge base with respect to the constants bound in the current ground literal. The results are then used to update relevance scores (line 9) of each background predicate to the predicate that is currently being learned.

The second component selects candidate antecedents based on their relevance results. Candidate selection (line 15) determines both which predicates should serve as antecedents in a rule, and how the arguments of the antecedents should be bound to the arguments of the consequent.

The third component performs weight learning (line 10) to determine the specific rule that relates the antecedents to the consequent based on the observed ground literals. Also included with weight learning is the problem of determining if and when the current rule requires new antecedents. Until a rule is deemed sufficiently accurate, the first three components operate together on each ground literal (although new antecedents are selected only occasionally).

After a rule becomes sufficiently accurate, the final component begins working to fine tune the learned rule (line 17). The goal here is to remove any irrelevant or redundant antecedents. This helps to simplify the rule definition, to improve rule accuracy, and to improve the efficiency of the knowledge base. In the remainder of this section, we examine each component in detail.

Table 3. The SCALE algorithm

```
Given:
    Stream of ground literals $\mathcal{X}$
    Set of background predicate definitions $\mathcal{P}$

Algorithm: SCALE($\mathcal{P}, \mathcal{X}$)
    $\mathcal{U} \leftarrow \emptyset$
    for each $X_t = (p_t, B_t, d_t) \in \mathcal{X}$ do
        $P \leftarrow$ find $p_t$ in $\mathcal{P} + \mathcal{U}$
        if $p_t \notin \mathcal{P} + \mathcal{U}$ then                 // start a new rule
            $P \leftarrow$ new rule with consequent $p_t$, variables from $B_t$, and $C = \boldsymbol{I} = \boldsymbol{w} = \emptyset$
            $\mathcal{U} \leftarrow \mathcal{U} + \{P\}$
        if $P \in \mathcal{U}$ then                                  // still learning rule
            $\mathcal{E} \leftarrow$ EVALUATE-BOTTOM-UP($P, X_t$)
            UPDATE-RELEVANCE($\mathcal{E}, P, X_t$)
            UPDATE-WEIGHTS($P, d_t$)
            if $P$ is learned then
                $\mathcal{U} \leftarrow \mathcal{U} - \{P\}$
                $\mathcal{P} \leftarrow \mathcal{P} + \{P\}$
            if $P$ is unlearnable then
                SELECT-NEW-CANDIDATE($P$)
        if $P \in \mathcal{P}$ then                                  // rule learned, now fine tune
            REMOVE-IRRELEVANT-ANTECEDANTS($P, X_t$)
    end for each
    return $\mathcal{P}$
```

3.2 Bottom-Up Evaluation

The goal of bottom-up evaluation is to evaluate all possible bindings of the background predicates $\mathcal{P}$ with respect to a small set of constants. Here, the constants are those bound by B_t in the current ground literal, X_t. In this case, SCALE's numerical representation and hierarchically structured rule allow us to augment typical bottom-up evaluation algorithms to achieve greater efficiency.

The approach begins with a closed world assumption, meaning that any predicate-binding combination not proved true by our procedure is assumed false. This allows the procedure to avoid evaluating explicitly any consequent for which no antecedent can be found to support the implication. For example, at least one term in a conjunctive rule must evaluate to true before the system attempts to evaluate the entire rule. This idea is similar to that of spreading activation [4], which propagates signals along active paths only.

We begin describing the algorithm by defining two terms. First, let $\mathcal{G}$ be the set of low-level (ground) rules in $\mathcal{P}$. These are rules which rely only on ground facts or constants as antecedents. Second, we define the term *improve* with respect to predicate evaluation. The truth value of a bound predicate P improves the evaluation of P' if $P \in I_{P'}$ and the output value of P constitutes evidence that P' will output *true*. With respect to the numerical representation underlying each predicate, P improves P' if the weighted evaluation of P is greater than

Table 4. Bottom-up knowledge base evaluation algorithm

```
Given:
    Set $\mathcal{P}$ of background predicates
    Example $X = (p, d, B)$

Algorithm: BOTTOM-UP($\mathcal{P}, X$)                         // main loop
1   $\mathcal{E} \leftarrow \emptyset$
2   $\mathcal{G} \leftarrow$ set of ground predicates in $\mathcal{P}$
3   for each $P \in \mathcal{G}$ do
4       $\mathcal{B} \leftarrow$ set of all bindings between $B$ and $V_p$
5       for each $B' \in \mathcal{B}$ do
6           $\mathcal{E} \leftarrow \mathcal{E} +$ BU-EVALUATOR($P, B, B'$)
7   return $\mathcal{E}$

Algorithm: BU-EVALUATOR($P, B, B^*$)                           //recursive procedure
8   $\mathcal{B} \leftarrow$ set of bindings between $B$ and $V_p$ given $B^*$
9   $\mathcal{E}' \leftarrow \emptyset$
10  for each $B' \in \mathcal{B}$ do
11      bind $B'$ to $V_p$ and evaluate $P$
12      $\mathcal{E}' \leftarrow \mathcal{E}' + \{P\}$
13          for each $P'$such that$P \in I_{P'}$ do
14              if $P$ improves $P'$ then
15                  $B'' \leftarrow$ binding from $P$ to $P'$
16                  $\mathcal{E}' \leftarrow \mathcal{E}' +$ BU-EVALUATOR($P', B, B''$)
17  return $\mathcal{E}'$
```

zero. This brings the left side of the threshold-implication rule ($\sum_{j=0}^{n-1} w_j i_j \geq \theta$) closer to or beyond θ, which suggests that further computation is justified.

Table 4 shows the bottom-up evaluation algorithm, which collects the set of explicitly evaluated predicates (both *true* and *false*) into the set $\mathcal{E}$. The main loop (lines 1–7) iterates over the set $\mathcal{G}$ of ground predicates. Each predicate $P \in \mathcal{G}$ is evaluated with respect to all possible bindings between the constants B bound in the ground literal X, and V_P. If the evaluation improves any of P's successors, then the algorithm recurses and evaluates those successors. The bindings of P are mapped to its successor P'. If P' has more arguments than P, then the extra arguments are bound to objects in B before bottom-up evaluation continues (line 8). While not shown here, pilot experiments suggest that this bottom-up evaluation procedure reduces the number of rule evaluations substantially [5]. Further research on this topic to determine efficiency and prove completeness is required.

3.3 Selecting Rule Antecedents

We now turn to the problem of selecting which antecedents will form the basis for a predicate. This is a central problem in SCALE. An efficient selection algorithm may allow the system to learn new rules almost indefinitely. Otherwise, the system will become overwhelmed by the number of possibilities. SCALE's approach is to select a set of *related* antecedents for each predicate, not a minimal set. This allows useful redundancy and more flexibility in learning predicate definitions.

Having established the set $\mathcal{E}$ of background predicates explicitly evaluated via the bottom-up procedure, we now turn to the task of selecting specific antecedents for a given predicate. Our approach relies on the simplicity of threshold-implication, here implemented as a perceptron. More powerful connectionist algorithms can detect and represent subtle relationships between antecedents and consequent, but this makes predicting the usefulness of a given antecedent difficult. Threshold-implication can represent only simple relationships, so predicting which antecedents may be useful is relatively simple.

There are only two basic types of inputs to a perceptron. Both cases can be coarsely detected without training the perceptron. *Excitatory inputs* indicate when the perceptron should produce a positive output (bound predicate is *true*). To detect an excitatory input, SCALE computes the conditional probability $s^+ = \Pr[P_{out} = \mathit{true} | P_{in} = \mathit{true}, \mathrm{Data}]$ that the consequent is *true* given that the candidate antecedent is *true*. Probabilities closer to one indicate a stronger relationship (more overlap) between P_{in} and P_{out}.

Inhibitory inputs indicate when the perceptron should produce a negative output (bound predicate is false). Detecting inhibitory inputs is similar to detecting excitatory inputs. SCALE computes the conditional probability $s^- = \Pr[P_{out} = \mathit{true} | P_{in} = \mathit{false}, \mathrm{Data}]$ that the consequent is *true* given that the candidate antecedent is *false*. Note that while both measures produce values in the range zero to one, they are not complementary.

Antecedent selection now proceeds based on these detection measures. First, note that candidate antecedents are not yet linked as actual dependencies, and therefore have no entry in the predicate's weight vector. Now, for a given ground literal, the learned predicates are first evaluated bottom-up. In the remaining steps, only predicates explicitly evaluated by the bottom-up procedure are considered. The excitatory and inhibitory scores are computed over several training instances. Candidate antecedents with the highest score are then added as new antecedents to P_{out} when the weight learning component determines that a new antecedent is needed.

3.4 Weight Learning

Weight learning in SCALE addresses three problems. The first is to determine the relationship between the antecedents and the consequent to a rule. This is the process of learning the specific weight and threshold values used by threshold-implication. All updates are made incrementally, with each observed ground literal.

The weight vector $\boldsymbol{w}$ for a predicate P is initialized with small random values. Then, for each literal associated with P, the weights get updated via the perceptron rule [3]. Specifically,

$$w_i' = w_i + \alpha(d - p(\cdots))q_i(\cdots) \quad ,$$

where d is the value associated with the literal (-1 if negated, $+1$ otherwise), $p(\cdots)$ is the outcome of threshold-implication given the current weight vector, $q_i(\cdots)$ is the value of antecedent i, and $0 < \alpha \leq 1$ governs the magnitude of each update. A similar update also applies to the threshold value, θ. Assuming the positive and negative ground literals are linearly separable with respect to the selected antecedents, this update procedure is guaranteed to converge after observing a finite number of ground literals [3].

The second and third problems addressed by weight learning concern the decision of whether and when to add new antecedents to a rule. The decision is based on determining whether the current set of antecedents are sufficient to let the weights converge. When weights cannot converge, we call the rule *unlearnable* with respect to the currently selected antecedents. This then triggers selection of new antecedents, which continues in conjunction with weight learning until the weights converge, or there are no more available antecedents (there is no bound on the number of antecedents to a given rule). Likewise, when the weights have converged, we call the rule *learned*, which halts the antecedent selection process and initiates removal of any irrelevant antecedents. To meet these two requirements, SCALE uses a modification of the perceptron algorithm [3] originally developed for perceptron trees [6] for learning the antecedent weights.

Briefly, the Perceptron Convergence and Cycling Theorems [3] are combined with empirically established threshold values to provide definitions for *learned* and *unlearnable*. A predicate with n antecedents is learned if it correctly evaluates $l(n)$ consecutive ground literals. Similarly, the predicate is unlearnable if it makes $u(n)$ consecutive weight updates without exploring a new area of weight-space. Given this ability to recognize predicates as learned and unlearnable, each predicate must determine a set of antecedents that is sufficient for learning. Stracuzzi [5] provides details on the empirical determination of $l(n)$ and $u(n)$.

3.5 Removing Irrelevant Antecedents

After weight learning converges, the final component of removing (pruning) irrelevant and redundant antecedents begins. Pruning is important for several reasons. First, bottom-up evaluation is most efficient when the number of antecedents to a given predicate is small. Few unnecessary antecedents can quickly lead to many unnecessary predicate evaluations. Second, the presence of extra antecedents can affect rule generalization adversely by making the rule overly specific. Simplifying the rules also helps to improve interpretation by humans.

Scale employs an online version of the Randomized Variable Elimination (RVE) algorithm [7] for pruning. Briefly, RVE is a general-purpose feature selection algorithm motivated by the idea that, in the presence of many irrelevant variables (here, antecedents), the probability of successfully selecting several irrelevant variables simultaneously at random is quite high. The algorithm computes the cost of attempting to remove k input variables of n remaining variables given that r are relevant. A sequence of values for k (given n and r) is then found by minimizing the aggregate cost of removing all $N - r$ irrelevant inputs. Note that n represents the number of remaining variables, while N denotes the total number of variables in the original problem. Since r is not known in advance, methods for estimating its value are employed.

Rule simplification in SCALE relies heavily on weight learning. Here, a fully trained copy of the predicate is retained while various reductions to the set of antecedents are considered. This is an important step, as a fully trained copy of the rule remains available to act as an antecedent to other predicates, even while pruning occurs. SCALE therefore maintains knowledge stability [8] during structural revision, which has important implications for the success and efficiency of learning.

4 Experimental Evaluation

The objective of this work is to present a statistical method for incremental induction of a first-order knowledge base. In the following experiments, we will show that our approach can construct new, hierarchically structured rules that generalize the ground literals that were used to create them. We show that our method captures the domain structure necessary for producing a succinct and accurate knowledge base that generalizes well to new situations. We also demonstrate that our approach is computationally efficient.

4.1 Evaluation Measures and Protocol

In order to evaluate the generality and structure of the learned rules, along with the computational efficiency of the SCALE algorithm, we focus on the following three properties of the algorithm and its performance.

- *Rule accuracy* measures the ability of each predicate to correctly evaluate binding combinations not encountered during learning.
- *Rule complexity* measures the number of antecedents to each predicate and provides a view of the density of the learned knowledge base.
- *Search space size* counts the number of hypotheses considered during learning, providing a view of the algorithm's efficiency and scalability.

CPU time is not a consideration here, as SCALE and the algorithms against which we compare it are implemented in different programming languages with varying levels of optimization. Moreover, some of the algorithms operate on batch data, which trades less CPU time for greater memory requirements, while others operate in an online manner.

We apply the above measures to two systems in addition to SCALE. First is FOIL6.4, using default options [9]. FOIL is one of a few ILP systems with both the ability to learn multiple predicates from ground literals, and a ready-to-use implementation available. FOIL differs from SCALE in that it requires batch rather than online data, computes a total ordering over concepts prior to learning instead of ordering concepts by learnability, and uses a strictly symbolic representation instead of a combination of symbolic and subsymbolic representations.

The second system is the Alchemy implementation [10] of Markov logic networks [11]. Structure learning in Alchemy attempts to learn a global list of weighted, first-order clauses that implicitly cover all of the available ground literals. This approach is quite different from either SCALE or FOIL, which learn individual definitions for each predicate. Nevertheless, the result is an intensionally defined, first-order knowledge base that can be used to infer the truth value (or probability) of any ground literal given background knowledge.

The exact test procedure used for each learning system depends on whether the system is online or batch. The batch systems (FOIL and Alchemy) were provided with all ground literals for each predicate in the knowledge base at the beginning of execution. SCALE was presented with one literal at a time. Each literal was drawn at random and without replacement from the entire knowledge base of literals (all predicates) until all literals were presented. After this, all ground literals were replaced, and the cycle began again.

The ground literals used in our experiments were divided into separate sets for learning and for evaluation. The sets were selected at random over a uniform distribution for concepts with arity two or greater. However, since some concepts contain very lopsided distributions of positive and negative instances, we selected a fixed number of positive and negative ground literals for each concept. For arity one concepts, the training and testing examples were chosen manually. This is necessary because the number of unique examples is small, and a random selection is unlikely to be representative of the target concept. The objective in selecting data manually was to act as a terse instructor by choosing a small number of illustrative examples.

4.2 Chess Domain Overview

Chess presents a rich and structured domain. There are six pieces, each possessing its own rules of movement and distinct strategies. More importantly, the pieces often combine to produce complex tactical and strategic patterns.

As described earlier, each constant in the domain represents a vector of three integers which describe each cell on a standard 8 × 8 chess board. These cells then form the 832 (13 × 8 × 8) constants which may then be bound to the arguments of each predicate. The representation used here is not necessarily the most natural possible choice for the domain. However, we choose this approach to add additional levels of structure to the knowledge base in an effort to better demonstrate the structure-learning properties of the systems.

While SCALE supports this representation for domain constants directly, FOIL and Alchemy do not. We therefore provided three predicates in background knowledge that the two systems could use to transform the above representation into a representation that they supported. Pilot tests confirmed that this approach was effective for both systems.

There are 128 concepts for which definitions must be learned. Of these, 71 have arity one, with 28 concepts describing cell locations such as rank and file, and 43 describing cell occupants such as *king*, or *white queen*. The remaining 57 concepts have arity two, with 19 of these describing relations among cell locations, such as whether cells lie on the same rank, file or diagonal. The final 38 concepts test whether a move from one cell to another constitutes a legal move for a given piece, and whether a given piece can be captured. These concepts are the most complex, and depend heavily on the definitions of previously learned concepts. There are a total of 30,011 ground literals in the knowledge base used for training, and 342,567 literals in the knowledge base used for testing.

4.3 Results

The SCALE system learned definitions for all 128 predicates, producing a mean accuracy of 98.7% and a knowledge base with 12 layers of predicates and 612 dependencies (total antecedents over all predicates). The rules were produced from a search of 3,966 hypotheses, where each addition or subtraction of an antecedent from any rule constitutes one hypothesis. Analysis of the result shows that the system experienced the most difficulty for concepts related to long-distance piece movement (such as bishops, rooks, and queens) and piece capture. This is not surprising given that these concepts represent the most complex relations among pieces and locations.

These results are particularly impressive given that neither FOIL nor Alchemy completed learning in the full chess domain. FOIL was terminated after ten CPU days for lack of progress in computing its initial ordering over predicates. Alchemy was terminated after two days, at which point it had exhausted all available main memory on the host computer and caused the system to thrash. The Alchemy result is not surprising. The first-order nature of its external representation is deceiving. Internally the algorithm enumerates the first-order predicates into propositions prior to inference and learning. This underlying flat representation and the statistical nature of the algorithm are not well suited to highly structured domains or sparse data sets.

In an effort to produce more informative results, we next disregarded the last 38 concepts (all related to piece movement and capture) and ran the experiments again. Although Alchemy still required too much memory to complete the experiment, both SCALE and FOIL were successful. In this case, SCALE produced 100% accuracy on all 90 predicates, and created a knowledge base with three layers and 47 dependencies from a search of 1,081 hypotheses. FOIL produced a knowledge base with a mean accuracy of 79.2%, using 8 layers, 892 dependencies and 1,973 hypotheses.

We turn now to the task of finding and comparing trends in the searches conducted by the learning algorithms. Figure 1 plots hypothesis complexity, as measured by the number of dependencies, against time, as measured by the number of hypotheses considered. Both SCALE and FOIL increase hypothesis complexity at a gradual pace. This is deceptive, however. SCALE is training on all 90 concepts simultaneously, while FOIL trains in sequence. The difference becomes clear as SCALE begins to reduce hypothesis complexity while FOIL continues to increase. SCALE does a better job of selecting only relevant dependencies.

SCALE also experiences a long sequence of hypotheses with nearly constant complexity between 400 and 800 hypotheses considered. This indicates that the

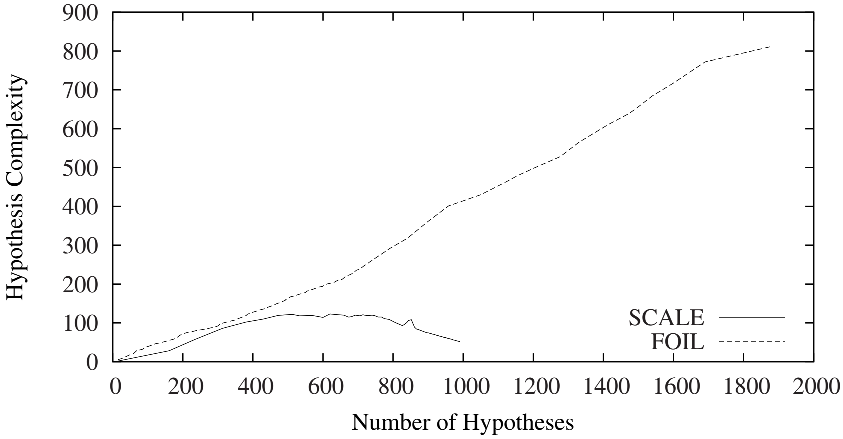


Fig. 1. Changes in hypothesis complexity during search

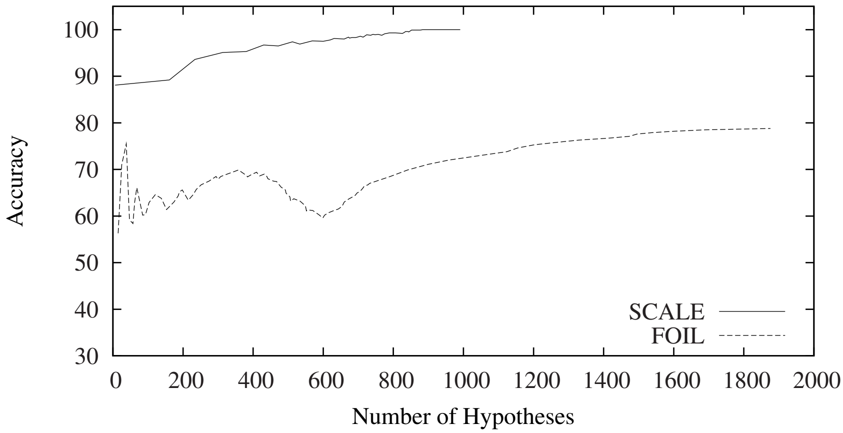


Fig. 2. Changes in hypothesis generalization levels during search

number of new dependencies added is offset by the number of dependencies removed. This is an attractive property in that it keeps evaluation costs low.

Figure 2 shows the generalization results of the algorithms. Scale improves its performance over time, because it attempts to learn all concepts simultaneously, and continues until all concepts are learned. So even though scale performs only local searches when attempting to find predicate definitions, the resulting change in performance is global in scope. Improvements to the definitions of lower-level predicates also improve performance at higher levels. Foil's performance is less monotonic. Predicate definitions are still induced via local search, but the predicates are learned in an automatically determined, fixed order. The result is an overall decline in generalization performance over time, because complex predicates are not considered or evaluated during early stages of induction.

Another interesting note concerns the substantially different order in which SCALE and FOIL learn concepts. FOIL attempts to learn the concepts approximately in order from simple to complex, while SCALE simply tries to learn all concepts simultaneously, but makes them available for use as background knowledge in whatever order the system acquires a sufficiently reliable predicate definition (as defined by the *learned* test). One may expect the two orderings to be similar, but in this case they are not, and the generalization results imply that SCALE's approach makes more sense for learning.

Figure 3 shows a scatter-plot comparison of SCALE and FOIL on the reduced chess domain. Each scatter-plot is divided in half by a diagonal line representing equal performance by the two algorithms. Each point represents one concept from either the small or large cards domain. Points located above the diagonal line indicate that SCALE's performance value on the given metric was higher, while points located below the line indicate that FOIL's metric value was higher. Higher values are better for accuracy, but lower values are preferred for number of antecedents. The number of hypotheses and number of layers metrics are

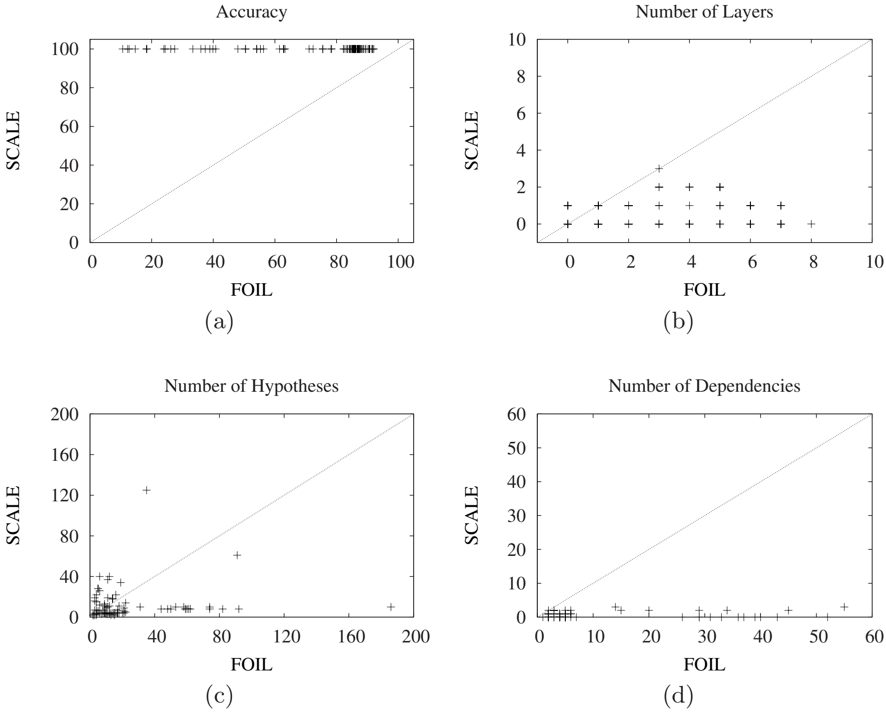


Fig. 3. Scatter-plots comparing concepts learned by SCALE and FOIL for the chess domains. The plots show concept accuracy (a), location in hierarchy (b), number of hypotheses required for learning (c), and number of dependencies or antecedents (d).

included here for comparison, but are not easily lumped into higher/lower is better groups. For example, less search is good (fewer hypotheses considered) provided that generalization and compactness does not suffer.

Panel (a) indicates SCALE's dominance over FOIL with respect to concept accuracy. Panel (b) shows that FOIL tends to require more representational layers than SCALE for a given concept. Panel (c) shows that both methods tend to require similar sized searches for learning, although the large cluster of points near the origin favors SCALE. Most important however, is panel (d), which shows that even though SCALE and FOIL consider similar numbers of hypotheses during learning, FOIL almost always requires more dependencies to represent the concepts. This is important because the number of dependencies influences the number of arguments that must be estimated, which in turn influences the amount of data required. The results therefore suggest that SCALE has smaller data requirements than FOIL.

5 Discussion

SCALE's strengths in constructing knowledge bases from ground literals stem from two main ideas. The first idea derives from SCALE's use of threshold-implication to model each predicate definition. This brings two important advantages. One is that threshold-implication can represent functions more general than the horn clauses used by many ILP systems. This provides greater flexibility in learning even while restricting each predicate to simple concepts. Other, more sophisticated forms of weighted-implication are possible, but deciphering the reasons for learning failure can be a challenge. For example, the learner's hypothesis space may be inappropriate, data may be insufficient, or the search space may be intractably large. SCALE's use of threshold-implication reduces the number of reasons why learning may fail. All concepts must be linearly separable, so when learning fails, we know that the predicate requires additional (or different) antecedents.

The other advantage of threshold-implication is that the weights associated with each antecedent provide for more a continuous relationship among the different hypotheses that a predicate may represent. Adjusting the weight of an antecedent provides a much finer level of control than simply adding or removing terms or clauses. The effect is to provide SCALE with the ability to experiment with relations among many antecedents simultaneously and efficiently. We can also view the weights as a approximate distance measure between distinct predicate definitions, allowing for a simple and direct form of comparison.

The second key idea is the approach of organizing the learned predicates hierarchically in the knowledge base to reduce the number of candidate dependencies that are explicitly evaluated by the learning algorithm. This impacts both the amount of computation needed to learn a model, and the quality of the learned model. Pearl [12] suggests that the number of models considered by an algorithm (along with the number of comparisons made [13]) influences the chance of overfitting. SCALE's approach to antecedent selection has the added benefit of trying to provide a small set of antecedents, rather than a minimal set. This provides flexibility and useful forms of redundancy to the learner, for example by providing alternate concept representations in case one of the selected predicates has an inaccurate model of its target concept.

5.1 Related Work

Blockeel and De Raedt [14] discuss the problem of converting extensional predicate definitions into intensional predicate definitions. Their approach uses the CLAUDIEN system for finding valid clauses in theory [15], plus a measure of rule quality based on the extent to which the rule compresses the knowledge base, to build intensional predicate definitions. A key difference between this approach and SCALE is that SCALE uses a combined numerical and logical representation. One consequence of this difference is that SCALE operates incrementally, while the CLAUDIEN-based approach requires batch data.

Few incremental ILP systems appear in the literature. They often assume noise-free data and need to verify changes to the hypothesis either by consulting an oracle [16,17] or by experimenting with the environment [18]. Such systems tend not to be robust due in part to their search of a discrete search space. Adding a single condition to a hypothesis often has great impact on its quality and is therefore difficult to justify with a single incremental example. In contrast, SCALE uses new examples to navigate a continuous space of numerical weights. The hypothesis structure changes only after sufficient numbers of examples are observed.

As an exception to the above, Basilio, Zaverucha, and Barbose [19] discuss the first-order cascade ARTMAP (FOCA) system, which is a neural network-based theory learning and refinement system. One difference between FOCA and SCALE is that the former uses fuzzy ARTMAP neural networks [20], which have representational properties similar to threshold-implication but different training and structural properties, to represent rules instead of perceptrons. A second difference is that FOCA learns only a single predicate during each run. Although multiple runs can be used to learn multiple predicates, this is different from SCALE's approach, which does not assume that a complete definition for one predicate will be learned before learning begins on another predicate definition.

Probabilistic logic learning frameworks [21] combine statistical methods with logical learning. For example, statistical logic programs [22] and Bayesian logic programs [23] associate logical structures with probabilities. Although SCALE also associates numbers with logical structures, its hypotheses are not probabilistic. SCALE uses a numeric representation only for incremental hypothesis space search. At any given time, the current hypothesis represents a complex logical formula. Most probabilistic logic systems either focus on parameter adjustment [24] or structure learning [25]. Interleaving the two remains an active research area [26], and SCALE has demonstrated that it can naturally combine parameter adjustment with structural learning.

5.2 Future Work

Two directions for future work concern learning rules with existentially quantified variables, and learning recursive rules. For the former problem, we can allow SCALE to consider creating quantified variables in antecedents in addition to adding antecedents with fully bound variables. This constitutes a substantial expansion of the search space however, and further study is required to determine how to achieve this expansion while still respecting SCALE's assumptions about predicate simplicity.

Recursive rule learning is already possible in SCALE, but requires further study to improve tractability. The system currently assumes that only successfully learned predicates may become antecedents for unlearned predicates. This simplifies the weight learning task because predicates with stable definitions provide consistent input signals. However, this assumption may also prevent SCALE from finding the most compact representation. In essence, SCALE enforces a partial ordering on the intermediate concepts based on learnability. Other orderings are possible and may lead to more efficient structures. Removing this assumption would both alleviate this problem and would allow for learning of recursive definitions. Reliance on inconsistent and noisy signals (unlearned predicates) will tend to require more examples, however, and constitutes a non-trivial expansion of the search space.

A third direction for expanding SCALE would be to embrace a probabilistic representation of the predicates. Swapping the perceptron learner for Naive Bayes would accomplish much of the change without altering representational power, but other questions remain. For example, the definitions of *learned* and *unlearnable* must be adjusted, but the mapping is not immediately clear. Similarly, the threshold at which a connection between one concept and another becomes useful (or necessary) is also unclear. The scoring functions used by Bayes nets solve exactly this problem, but the consequences of merging of these functions with the type of search conducted by SCALE are unclear.

6 Conclusion

Knowledge bases play an important role in a wide variety of work in artificial intelligence. In this work, we presented a statistical approach to building hierarchically structured, first-order knowledge bases. We showed that our approach can construct new rules from both background knowledge and ground literals to generalize the ground literals. We also demonstrated that our method produces succinct and accurate knowledge bases, and successfully captures the domain structure. Finally, we showed that our approach is computationally efficient and scales well to domains with many relations. Many lines of future development are available for SCALE, but our initial results show that the systems presents an excellent starting point for new research into knowledge base induction.

Acknowledgments

The authors thanks the anonymous reviewers for many helpful comments. Much of this work was completed while the first author was a graduate student at the University of Massachusetts, Amherst. This material is based in part on research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed on implied, of DARPA or the U. S. Government.

References

1. Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
2. De Raedt, L.: Inductive Theory Revision: An Inductive Logic Programming Approach. Academic Press, London (1992)
3. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry (Expanded Edition). MIT Press, Cambridge (1972)
4. Anderson, J.: A spreading activation theory of memory. Journal of Verbal Learning and Verbal Behavior 22, 261–295 (1983)
5. Stracuzzi, D.J.: Scalable Knowledge Acquisition Through Cumulative Learning and Memory Organization. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA (2006)
6. Utgoff, P.E.: Perceptron trees: A case study in hybrid concept representations. Connection Science 1(4), 377–391 (1989)
7. Stracuzzi, D.J., Utgoff, P.E.: Randomized variable elimination. Journal of Machine Learning Research 5, 1331–1364 (2004)
8. Quartz, S.R., Sejnowski, T.J.: The neural basis of development: A constructivist manifesto. Behavioral and Brain Sciences 20, 537–596 (1997)
9. Quinlan, J.R., Cameron-Jones, R.M.: Foil: A midterm report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 3–20. Springer, Heidelberg (1993)
10. Kok, S., Singla, P., Richardson, M., Domingos, P.: The Alchemy system for statistical relational AI. Technical report, University of Washington, Seattle, WA (2005), `http://www.cs.washington.edu/ai/alchemy/`
11. Richardson, M., Domingos, P.: Markov logic networks. Journal of Machine Learning Research 62(1-2), 107–136 (2006)
12. Pearl, J.: On the connection between the complexity and credibility of inferred models. Internation Journal of General Systems 4, 255–264 (1978)
13. Jensen, D.D., Cohen, P.R.: Multiple comparisons in induction algorithms. Machine Learning 38(3), 309–338 (2000)
14. Blockeel, H., De Raedt, L.: Inductive database design. In: Michalewicz, M., Raś, Z.W. (eds.) ISMIS 1996. LNCS (LNAI), vol. 1079, pp. 376–385. Springer, Heidelberg (1996)
15. De Raedt, L., Bruynooghe, M.: A theory of clausal discovery. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambéry, France, pp. 1058–1063. Morgan Kaufmann, San Francisco (1993)

16. Sammut, C., Banerji, R.B.: Learning concepts by asking questions. In: Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann, San Mateo (1986)
17. Shapiro, E.: An algorithm that infers theories from facts. In: Drinan, A. (ed.) Proceedings of the Seventh International Joint Conference on Artificial Intelligence, pp. 446–451. Morgan Kaufmann, San Francisco (1981)
18. Taylor, K.: Autonomous Learning by Incremental Induction and Revision. PhD thesis, Australian National University (1996)
19. Basilio, R., Zaverucha, G., Barbosa, V.C.: Learning logic programs with neural networks. In: Rouveirol, C., Sebag, M. (eds.) ILP 2001. LNCS (LNAI), vol. 2157, pp. 15–26. Springer, Heidelberg (2001)
20. Carpenter, amd Grossberg, G.A., Rosen, S., David, B.: Fuzzy ART: Fast, stable learning and categorization of analog patterns by an adaptive resonance system. Neural Networks 4, 759–771 (1991)
21. De Raedt, L., Kersting, K.: Probabilistic logic learning. SIGKDD Explorations 5(1), 31–48 (2003)
22. Muggleton, S.: Stochastic logic programs. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 254–264. IOS Press, Amsterdam (1996)
23. Kersting, K., De Raedt, L.: Bayesian logic programs. In: Cussens, J., Frisch, A. (eds.) Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming, pp. 138–155. Springer, Heidelberg (2000)
24. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning 44(3), 245–271 (2001)
25. Muggleton, S.H.: Learning stochastic logic programs. In: Getoor, L., Jensen, D. (eds.) Proceedings of the AAAI 2000 Workshop on Learning Statistical Models from Relational Data. AAAI Press, Menlo Park (2000)
26. Muggleton, S.H.: Learning structure and parameters of stochastic logic programs. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 198–206. Springer, Heidelberg (2003)

A Note on Refinement Operators for IE-Based ILP Systems

Alireza Tamaddoni-Nezhad and Stephen Muggleton

Department of Computing, Imperial College London
{atn,shm}@doc.ic.ac.uk

Abstract. ILP systems which use some form of Inverse Entailment (IE) are based on clause refinement through a hypotheses space bounded by a most specific clause. In this paper we give a new analysis of refinement operators in this setting. In particular, Progol's refinement operator is revisited and discussed. It is known that Progol's refinement operator is incomplete with respect to the general subsumption order. We introduce a subsumption order relative to a most specific (bottom) clause. This subsumption order, unlike previously suggested orders, characterises Progol's refinement space. We study the properties of this subsumption order and show that ideal refinement operators exist for this order. It is shown that efficient operators can be implemented for least generalisation and greatest specialisation in the subsumption order relative to a bottom clause. We also study less restricted subsumption orders relative to a bottom clause and show how Progol's incompleteness can be addressed.

1 Introduction

Searching a refinement lattice bounded below by a bottom clause is the basis of several state-of-the-art ILP systems (e.g. Progol [7], Aleph [11]). These systems use refinement operators together with a search method to explore a bounded hypotheses space. For example, in the default setting, Progol uses a A*-like search together with a top-down refinement operator. Progol's refinement operator is designed to avoid redundancy in a A*-like search. This, however, leads to incompleteness of Progol's refinement operator with respect to the general subsumption order. There have been previous attempts to characterise Progol's refinement space and also to address the Progol's incompleteness. In particular, special cases of subsumption have been suggested to characterise Progol's refinement space [1]. In this paper we give a new analysis of refinement operators in a Progol-like ILP system. We introduce a subsumption order relative to a most specific (bottom) clause. This subsumption order, unlike previously suggested orders, characterises Progol's refinement space. We show the existence of ideal refinement operators for this subsumption order. It is also shown that efficient operators can be designed for least generalisation and greatest specialisation in the subsumption order relative to a bottom clause. We also study less restricted subsumption orders relative to a bottom clause and show how Progol's incompleteness (due to the choice of ordering in the bottom clause) can be addressed.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 297–314, 2008.

The theoretical results presented in this paper can be applied to ILP systems which use Inverse Entailment (IE) as well as any other ILP system which uses a most specific clause to restrict the search space. In this paper we first review clause refinement in Progol as an example of an IE-based ILP system. Section 3, gives examples of Progol's incompleteness with respect to the general subsumption order. In order to characterise Progol's refinement, Sections 4 defines sequential subsumption order relative to a bottom clause and describes the properties of this subsumption order. In section 5, refinement operators are defined for subsumption order relative to a bottom clause and the properties of these operators are discussed. Section 6 describes less restricted subsumption orders relative to a bottom clause and shows how Progol's incompleteness can be addressed. Related work is discussed in Section 7. Section 8 concludes the paper.

2 Clause Refinement in Progol

We assume the reader to be familiar with the basic concepts from logic programming and inductive logic programming [8] and also the basic concepts from ordered sets and lattices [3]. The following definition is a reminder of the concept of refinement operators and several properties of these operators.

Definition 1 ([8,1]). *Let $\langle G, \succeq\rangle$ be a quasi-ordered set. A (downward) refinement operator for $\langle G, \succeq\rangle$ is a function ρ, such that $\rho(C) \subseteq \{D|C \succeq D\}$, for every $C \in G$.*

- *The sets of one-step refinements, n-step refinements and refinements of some $C \in G$ are respectively: $\rho^1(C) = \rho(C)$, $\rho^n(C) = \{D|$ there is an $E \in \rho^{n-1}(C)$ such that $D \in \rho(E)\}, n \geq 2$ and $\rho^*(C) = \rho^0(C) \cup \rho^1(C) \cup ..$*
- *A ρ-chain from C to D is a sequence $C = C_0, C_1, \ldots, C_n = D$, such that $C_i \in \rho(C_i - 1)$ for every $1 \leq i \leq n$.*
- *ρ is locally finite if for every $C \in G$, $\rho(C)$ is finite and computable.*
- *ρ is proper if for every $C \in G$, $\rho(C) \subseteq \{D|C \succ D\}$.*
- *ρ is is complete if for every $C, D \in G$ such that $C \succ D$, there is an $E \in \rho^*(C)$ such that $D \sim E$ (i.e. D and E are equivalent in the $\succeq$-order).*
- *ρ is weakly complete if $\rho^*(\Box) = G$, where $\Box$ is the top element of G.*
- *ρ is non-redundant if for every $C, D, E \in G$, $E \in \rho^*(C)$ and $E \in \rho^*(D)$ implies $C \in \rho^*(D)$ or $D \in \rho^*(C)$.*
- *ρ is ideal if it is locally finite, proper and complete.*
- *ρ is optimal if it is locally finite, non-redundant and weakly complete.*

We can define analogous concepts for the dual case of an upward refinement operator.

The Progol algorithm [7] is based on successive construction of definite clause hypotheses H from a language $\mathcal{L}$. H must explain the examples E in terms of background knowledge B. Each clause in H is found by choosing an uncovered positive example e and searching through the graph defined by the refinement ordering $\succeq$ bounded below by the bottom clause associated with e. We define this setting more formally as follows.

Definition 2 (Progol refinement setting). *Let $\mathcal{S} = \langle B, E, \mathcal{L}, \succeq \rangle$ be Progol's ILP setting as defined in [7]. Let E consist of a set of positive and negative examples (ground unit clauses). The empty clause, denoted by $\Box$, is the maximal$_{\succeq,\mathcal{L}}$ element in $\mathcal{L}$. The "bottom" clause, denoted by $\bot$, is the least$_{\succeq,\mathcal{L}}$ element such that $B, \bot \models e$. Refinement of clause C, denoted by $\rho(C)$, is the set of maximal$_{\succeq,\mathcal{L}}$ clauses D such that $C \succ D \succeq \bot$.*

The refinement operator in Progol is designed to avoid redundancy and to maintain the relationship $\Box \succeq H \succeq \bot$ for each clause H. Since $H \succeq \bot$, it is the case that there exists a substitution θ such that $H\theta \subseteq \bot$. Thus for each literal l in H there exists a literal l' in $\bot$ such that $l\theta = l'$. Clearly there is a uniquely defined subset $\bot(H)$ consisting of all l' in $\bot$ for which there exists l in H and $l\theta = l'$. A non-deterministic approach to choosing an arbitrary subset S' of a set S involves maintaining an index k. For each value of k between 1 and n, the cardinality of S, we decide whether to include the kth element of S in S'. Clearly, the set of all series of n choices corresponds to the set of all subsets of S. Also for each subset of S there is exactly one series of n choices. To avoid redundancy and maintain θ-subsumption of $\bot$ Progol's refinement operator maintains both k and θ. The refinement operator ρ defined in [7] allows more than one literal in H to be mapped to the same literal l' in $\bot$. However, in Progol's implementation of the refinement operator, index k is incremented after each step for the sake of efficiency. This means each literal of $\bot$ can be considered only once. In Appendix A, we give a revised definition (ρ_0) which describes the refinement operator as implemented in Progol.

3 Characterising Progol's Refinement

In this section we show that Progol's refinement cannot be described by the general subsumption order and that we need the notion of "sequential subsumption" in order to capture Progol's refinement. It can be shown that a refinement operator cannot be both complete and non-redundant [1]. However, a refinement operator can be weakly complete and non-redundant (optimal). As mentioned in the previous section, Progol's ρ is designed to be non-redundant and therefore it cannot be complete. However, it is known that Progol's refinement operator is also not weakly complete with respect to the general subsumption order [7]. This is demonstrated in the following example[1].

Example 1. Let B contain definitions for decrementation (dec), addition (plus) and the clause $mult(0, X, 0) \leftarrow$ with appropriate mode declarations M and let the example e be the clause $mult(1, 1, 1) \leftarrow$. Then $\bot$ is the clause

$$\begin{aligned}\mathrm{mult}(A, A, A) \leftarrow\ & \mathrm{dec}(A, B), \mathrm{plus}(B, A, A), \mathrm{plus}(B, B, B),\\ & \mathrm{mult}(B, A, B), \mathrm{mult}(B, B, B).\end{aligned}$$

[1] This example is a corrected version of Example 30 in [7].

Now consider clause C:

$$C = \mathrm{mult}(U, V, W) \leftarrow \mathrm{dec}(U, X), \mathrm{mult}(X, V, Y), \mathrm{plus}(Y, V, W).$$

Clause C is in $\mathcal{L}$, but given the ordering over $\bot$ there will be no element of Progol's $\rho^*(\square)$ containing this clause or a subsume-equivalent of this clause. $\diamond$

This first type of incompleteness is due to the choice of ordering in the bottom clause and the variable dependencies in the literals. As mentioned in the previous section, Progol's refinement uses an indexing over the literals and the literals in $\bot$ can only be considered from left to right.

As mentioned in the previous section, each literal from $\bot$ can be selected only once. This leads to the second type of incompleteness. The example below shows that Progol's refinement space is not a lattice with respect to the general subsumption, as the least general generalisation of clauses is not always in the refinement space.

Example 2. Let C, D and $\bot$ be clauses as defined below

$$\begin{aligned} C &= \mathrm{p}(X, Y) \leftarrow \mathrm{q}(X, X), \mathrm{q}(Y, W). \\ D &= \mathrm{p}(X, Y) \leftarrow \mathrm{q}(Z, X), \mathrm{q}(Y, Y). \\ \bot &= \mathrm{p}(X, Y) \leftarrow \mathrm{q}(X, X), \mathrm{q}(Y, Y). \end{aligned}$$

C and D can be generated by Progol's refinement given $\bot$, however, clause E below which is the least general generalisation (lgg) of C and D cannot be generated.

$$E = \mathrm{p}(X, Y) \leftarrow \mathrm{q}(Z, X), \mathrm{q}(U, U), \mathrm{q}(Y, W). \qquad \diamond$$

As another example of the second type of incompleteness, consider the following example adopted from [1].

Example 3. Let $\bot = \mathrm{p}(X) \leftarrow \mathrm{q}(X, X)$, then Progol's refinement only considers the following hypotheses.

$$\begin{aligned} C_1 &= \mathrm{p}(X) \\ C_2 &= \mathrm{p}(X) \leftarrow \mathrm{q}(X, X) \\ C_3 &= \mathrm{p}(X) \leftarrow \mathrm{q}(X, Y) \end{aligned}$$

However, the following clauses which subsume $\bot$ are not considered by Progol's refinement.

$$\begin{aligned} C'_1 &= \mathrm{p}(X) \leftarrow \mathrm{q}(X, Y), \mathrm{q}(Y, X) \\ C'_2 &= \mathrm{p}(X) \leftarrow \mathrm{q}(X, Y), \mathrm{q}(Y, Z), \mathrm{q}(Z, X) \\ &\ldots \end{aligned} \qquad \diamond$$

It has been suggested [1] that the second type of incompleteness is not a drawback as it can be justified by the examples and the MDL heuristic. In order to characterise Progol's refinement, the authors of [1] suggested a special case of subsumption, called weak subsumption, which does not allow substitutions that identify literals (i.e. for $C\theta$ there are no literals L_1 and L_2 in C such that $L_1\theta = L_2\theta$). For example, the clause $\mathrm{p}(X') \leftarrow \mathrm{q}(X', Y'), \mathrm{q}(Y', X')$ subsumes $\bot = \mathrm{p}(X) \leftarrow \mathrm{q}(X, X)$ with respect to the general subsumption, but it does not weakly subsume it. This is because substitution $\{X'/X, Y'/X\}$ identifies literals $\mathrm{q}(X', Y')$ and $\mathrm{q}(Y', X')$. The weak subsumption ordering, therefore, characterises the second type of incompleteness. However, it does not capture the incompleteness due to the ordering of the literals. For example, consider clauses C and $\bot$ in Example 1. C weakly subsumes $\bot$ but clause C is not considered by Progol's refinement.

As mentioned in the previous section, Progol's refinement operator scans $\bot$ from left to right and for each literal l' of $\bot$ decides whether to include a generalisation of it (i.e. l, where $l\theta = l'$) in H or not. $H\theta$ can be, therefore, characterised as a "subsequence" of $\bot$ rather than a "subset" of $\bot$. In the following sections we first define a special case of subsumption based on the idea of subsequences, and then we study the properties of this subsumption order. We show that Progol's refinement can be characterised by sequential subsumption relative to $\bot$. We also show that ideal refinement operators exist for this special case of subsumption.

4 Sequential Subsumption

Even though Progol does not use an explicit representation for ordered clauses, clauses in $\mathcal{L}$ are defined with a total ordering over the literals (Definition 21 in [7]). In order to characterise Progol's refinement we adopt an explicit representation for ordered clauses. The idea of ordered (or sequential) clauses is to consider a clause as a sequence of literals rather than a set of literals. This idea has been used in logic programming, in particular in the context of resolution (e.g. [2]). The concept of ordered clauses has been also used in ILP. For example, when defining upward refinement operators it is sometime necessary to duplicate literals in order to correctly invert an elementary substitution. Duplication of literals is not allowed for conventional clauses (which use a set notation) and therefore ordered clauses are used instead [8]. A subsumption relation for ordered clauses is studied in [4]. The difference between this subsumption order and the subsumption order considered in this paper is discussed in Section 7. There are also other applications of ordered clauses, for example in the context of data mining from sequential data(e.g. [6]). In this paper we use the same notion used in [8] and an ordered clause is represented as a disjunction of literals (i.e. $L_1 \vee L_2 \vee \cdots \vee L_n$). The set notation (i.e. $\{L_1, L_2, \ldots, L_n\}$) is used to represent conventional clauses.

Definition 3 (Ordered clause). *An ordered clause $\overrightarrow{C}$ is a sequence of literals $L_1, L_2, \ldots, L_n$ and denoted by $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_n$. The set of literals in $\overrightarrow{C}$ is denoted by C.*

Unlike conventional clauses, the order and duplication of literals matter for ordered clauses. For example, $\overrightarrow{C} = p(X) \vee \neg q(X)$, $\overrightarrow{D} = \neg q(X) \vee p(X)$ and $\overrightarrow{E} = p(X) \vee \neg q(X) \vee p(X)$ are different ordered clauses while they all correspond to the same conventional clause, i.e. $C = D = E = \{p(X), \neg q(X)\}$.

Selection of two clauses is defined as a pair of compatible literals and this concept was used by Plotkin to define least generalisation for clauses [9]. However, in this paper we use selections to define mappings of literals between two ordered clauses.

Definition 4 (Compatible literals). *Literals L and M are compatible if they have the same sign and predicate symbol.*

Definition 5 (Selection of clauses). *Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_n$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$ be ordered clauses. A selection of $\overrightarrow{C}$ and $\overrightarrow{D}$ is a pair (i, j) where L_i and M_j are compatible literals.*

Definition 6 (Selection function). *Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_n$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$ be ordered clauses. A set s of selections of $\overrightarrow{C}$ and $\overrightarrow{D}$ is called a selection function if it is a total function of $\{1, 2, \ldots, n\}$ into $\{1, 2, \ldots, m\}$.*

Example 4. Let $\overrightarrow{C} = L_1 \vee L_2 \vee L_3$ and $\overrightarrow{D} = M_1 \vee M_2 \vee M_3 \vee M_4$ be two ordered clauses and the set of all selections of $\overrightarrow{C}$ and $\overrightarrow{D}$ be $S = \{(1,1), (1,2), (2,1), (2,2), (3,4)\}$. Then, $s_1 = \{(1,1), (2,2), (3,4)\}$, $s_2 = \{(1,1), (2,1), (3,4)\}$ and $s_3 = \{(1,2), (2,1), (3,4)\}$ are examples of selection functions of $\overrightarrow{C}$ and $\overrightarrow{D}$. ◇

Definition 7 (Subsequence). *Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_l$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$ be ordered clauses. $\overrightarrow{C}$ is a subsequence of $\overrightarrow{D}$, denoted by $\overrightarrow{C} \sqsubseteq^s \overrightarrow{D}$, if there exists a strictly increasing selection function s such that for each $(i, j) \in s$, $L_i = M_j$.*

Example 5. Suppose that in Example 4, we have $L_1 = L_2 = M_1 = M_2$ and $L_3 = M_4$. $\overrightarrow{C}$ is then a subsequence of $\overrightarrow{D}$ because there exists strictly increasing selection function s_1 which maps each literal L_i from $\overrightarrow{C}$ to an equivalent literal M_j from $\overrightarrow{D}$. ◇

Definition 8 (Ordered substitution). *Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_l$ be an ordered clause and θ be a substitution. $\overrightarrow{C}\theta$ is defined as follows, $\overrightarrow{C}\theta = L_1\theta \vee L_2\theta \vee \cdots \vee L_l\theta$.*

Definition 9 (Sequential subsumption). *Let $\overrightarrow{C}$ and $\overrightarrow{D}$ be ordered clauses. We say $\overrightarrow{C}$ is a sequential generalisation of $\overrightarrow{D}$, denoted by $\overrightarrow{C} \succeq_s \overrightarrow{D}$, if there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{D}$. $\overrightarrow{C}$ is a proper sequential generalisation of $\overrightarrow{D}$, denoted by $\overrightarrow{C} \succ_s \overrightarrow{D}$, if $\overrightarrow{C} \succeq_s \overrightarrow{D}$ and $\overrightarrow{D} \not\succeq_s \overrightarrow{C}$. $\overrightarrow{C}$ and $\overrightarrow{D}$ are equivalent with respect to sequential subsumption, denoted by $\overrightarrow{C} \sim_s \overrightarrow{D}$, if $\overrightarrow{C} \succeq_s \overrightarrow{D}$ and $\overrightarrow{D} \succeq_s \overrightarrow{C}$.*

Example 6. Let $\overrightarrow{B} = p(X_1, Y_1) \vee q(X_1, Y_1) \vee r(X_1, Y_1) \vee r(Y_1, X_1)$, $\overrightarrow{C} = p(X_2, Y_2) \vee r(U_2, Y_2) \vee r(Y_2, V_2)$ and $\overrightarrow{D} = p(X_3, Y_3) \vee r(Y_3, V_3) \vee r(U_3, Y_3)$ be ordered clauses. Let $\theta_1 = \{X_2/X_1, Y_2/Y_1, U_2/X_1, V_2/X_1\}$, then $\overrightarrow{C}\theta_1$ is a subsequence of $\overrightarrow{B}$ and therefore $\overrightarrow{C} \succeq_s \overrightarrow{B}$. However, there is no substitution θ_2 such that $\overrightarrow{D}\theta_2$ is a subsequence of $\overrightarrow{B}$ and therefore $\overrightarrow{D} \not\succeq_s \overrightarrow{B}$. Note that for conventional clauses B, C and D we have $C\theta_1 \subseteq B$ and similarly for $\theta_2 = \{X_3/X_1, Y_3/Y_1, V_3/X_1, U_3/X_1\}$ we have $D\theta_2 \subseteq B$ and therefore $C \succeq B$ and $D \succeq B$. ◇

The following theorem shows the relationship between sequential subsumption and the general subsumption order.

Theorem 1. *Let $\overrightarrow{C}$ and $\overrightarrow{D}$ be ordered clauses. If $\overrightarrow{C} \succeq_s \overrightarrow{D}$, then $C \succeq D$.*

Proof. Suppose $\overrightarrow{C} \succeq_s \overrightarrow{D}$, then according to Definition 9 there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{D}$. Let $\overrightarrow{C}\theta = L_1\theta \vee L_2\theta \vee \cdots \vee L_l\theta$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$. Then for every literal $L_i\theta$ in $\overrightarrow{C}\theta$ there exists a literal M_j in $\overrightarrow{D}$ such that $L_i\theta = M_j$, and therefore $C\theta \subseteq D$. Hence, $C \succeq D$. □

Note that as shown in Example 6, the converse of Theorem 1 does not hold in general.

The languages which we consider in this paper (e.g. $\mathcal{L}$, $\overrightarrow{\mathcal{L}}_\perp$, etc.) correspond to a set of clauses which are generalisations of a flattened bottom clause [7]. Therefore, all clauses in $\mathcal{L}$, $\overrightarrow{\mathcal{L}}_\perp$, etc. are function-free and all substitutions we consider are variable substitutions[2]. Progol's refinement considers a subset of clauses in $\mathcal{L}$ which are sequential generalisation of the bottom clause. This subset of ordered clauses are defined as follows.

Definition 10 ($\overrightarrow{\mathcal{L}}_\perp$). *Let $\overrightarrow{\perp}$ be the bottom clause as defined in Definition 2. $\overrightarrow{C}$ is in $\overrightarrow{\mathcal{L}}_\perp$ if and only if there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{\perp}$.*

Let us consider the examples in the previous section with respect to Definition 10. In Example 1, if we now consider $\overrightarrow{C}$ and $\overrightarrow{\perp}$ as ordered clauses then $\overrightarrow{C} \notin \overrightarrow{\mathcal{L}}_\perp$, because there is no substitution θ such that $\overrightarrow{C}\theta$ can be a subsequence of $\overrightarrow{\perp}$. Similarly in Example 3, $\overrightarrow{C_1'} \notin \overrightarrow{\mathcal{L}}_\perp$, etc.

According to Definition 10, for each ordered clause $\overrightarrow{C}$ in $\overrightarrow{\mathcal{L}}_\perp$ there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{\perp}$. Thus, there exists a selection function s which maps each literal of $\overrightarrow{C}\theta$ to a literal of $\overrightarrow{\perp}$ and this selection function is strictly increasing. This implies that there is an injective mapping from the literals of $\overrightarrow{C}\theta$ to the literals of $\overrightarrow{\perp}$. Therefore, clause $\overrightarrow{C}$ can be encoded by the substitution θ and a set of integers K, i.e. the range of the selection function s. In Progol's refinement operator, θ and K are maintained for each clause in

[2] Substitution $\theta = \{v_j/u_j\}$ is a variable substitution if all v_j and u_j are variables.

order to decode the clause from $\overrightarrow{\bot}$. Moreover, according to Definition 21, refinements of a clause are constructed by adding literals which are generalisations of a literal from $\overrightarrow{\bot}$. These literals are generated by δ and they all correspond to the same literal l_k from $\overrightarrow{\bot}$. This means that a literal L_i from $\overrightarrow{C}$ is comparable (with respect to Progol's refinement) to a literal M_j from $\overrightarrow{D}$ if L_i and M_j are both mapped to the same literal of $\overrightarrow{\bot}$. This leads to more specific definitions for subsequence and sequential subsumption.

Definition 11 (Subsequence relative to $\bot$). *Let $\overrightarrow{\bot}$ and $\overrightarrow{\mathcal{L}}_\bot$ be as defined in Definition 10 and $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_l$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$ be ordered clauses in $\overrightarrow{\mathcal{L}}_\bot$ such that $\overrightarrow{C} \sqsubseteq^{s_1} \overrightarrow{\bot}$ and $\overrightarrow{D} \sqsubseteq^{s_2} \overrightarrow{\bot}$. $\overrightarrow{C}$ is a subsequence of $\overrightarrow{D}$ relative to $\bot$, denoted by $\overrightarrow{C} \sqsubseteq^{s}_{\bot} \overrightarrow{D}$, if there exists a strictly increasing selection function s such that for each $(i,j) \in s$, $L_i = M_j$ and there exists k, $1 \leq k \leq n$ and $(i,k) \in s_1$ and $(j,k) \in s_2$.*

In Definition 11 the selection function s maps literal L_i from $\overrightarrow{C}$ to an equivalent literal M_j from $\overrightarrow{D}$ if they both correspond to the same literal from $\overrightarrow{\bot}$.

Definition 12 (Sequential subsumption relative to $\bot$). *Let $\overrightarrow{\bot}$ and $\overrightarrow{\mathcal{L}}_\bot$ be as defined in Definition 10 and $\overrightarrow{C}$ and $\overrightarrow{D}$ be ordered clauses in $\overrightarrow{\mathcal{L}}_\bot$. We say $\overrightarrow{C}$ is a sequential generalization of $\overrightarrow{D}$ relative to $\bot$, denoted by $\overrightarrow{C} \succeq_\bot \overrightarrow{D}$, if there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{D}$ relative to $\bot$.*

As shown in Example 1, Progol's refinement cannot be weakly complete for $\langle \mathcal{L}, \succeq \rangle$, however, it can be weakly complete for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$.

Theorem 2. *ρ_0 is weakly complete for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$.*

A sketch proof for this theorem is given in the Appendix.

5 Ideal Refinement Operators for Sequential Subsumption Order Relative to $\bot$

In this section we define a refinement operator ρ_1 and show that ρ_1 is ideal for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$. First we define a mapping function which is used in the refinement operator. As mentioned in the previous section, in Progol's refinement operator, a substitution θ and a set of integers K are maintained for each clause in order to decode the clause from $\overrightarrow{\bot}$. In this setting, substitution θ maps variables from $\overrightarrow{C}$ to the variables of $\overrightarrow{\bot}$. The decoding, therefore, requires inverse substitution θ^{-1}. This can be achieved by maintaining the position of variables when the substitution θ is constructed [8]. However, in the mapping function used in this section, substitution θ maps variables from $\overrightarrow{\top}$ to the variables of $\overrightarrow{C}$, where $\overrightarrow{\top}$ is $\overrightarrow{\bot}$ with all variables replaced with new and distinct variables.

Definition 13 (Mapping function c). *Let $\overrightarrow{\bot}$ and $\overrightarrow{\mathcal{L}}_\bot$ be as defined in Definition 10, n be the number of literals in $\bot$. $\overrightarrow{\top}$ is $\overrightarrow{\bot}$ with all variables replaced with new and distinct variables. $\theta_\top$ is a variable substitution such that $\overrightarrow{\top}\theta_\top = \overrightarrow{\bot}$. Let θ be a variable substitution in Θ, where $\Theta = \{\theta' | \theta' \subseteq \hat{\theta}_\top$ and if $\{x/z, z/y\} \subseteq \theta'$ then $x/y \in \theta'\}$ and $\hat{\theta}_\top = \{y/x | \{x/z, y/z\} \subseteq \theta_\top\}$. Let $\mathcal{K}$ be power-set of $\{1, \ldots, n\}$. The mapping function $c : \mathcal{K} \times \Theta \rightarrow \overrightarrow{\mathcal{L}}_\bot$ is defined as follows:*

$$c(\langle K, \theta \rangle) = (\bigvee_{i=1}^{n} l_i | i \in K \text{ and } l_i \text{ is the ith literal of } \overrightarrow{\top})\theta.$$

The mapping function c, maps a tuple $\langle K, \theta \rangle$ into an ordered clause $\overrightarrow{C}$ in $\overrightarrow{\mathcal{L}}_\bot$. This mapping function also makes sure that the literals in $\overrightarrow{C}$ follow the same order as literals in $\overrightarrow{\bot}$. This condition is required for the refinement operator ρ_1 which is intended to be complete for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$.

The refinement operator ρ_1 is based on Laird's refinement operator [5] adopted for sequential subsumption and a refinement space bounded below by a bottom clause.

Definition 14 (ρ_1). *Let $\overrightarrow{\bot}$ and $\overrightarrow{\mathcal{L}}_\bot$ be as defined in Definition 10, $\overrightarrow{C}$ be an ordered clause in $\overrightarrow{\mathcal{L}}_\bot$, n be the number of literals in $\bot$, k be a natural number, $1 \leq k \leq n$, $\overrightarrow{\top}$, Θ and $\mathcal{K}$ be defined as in Definition 13. Let $K \in \mathcal{K}$, $\theta \in \Theta$, $\overrightarrow{C} = c(\langle K, \theta \rangle)$ and the mapping function c be defined as in Definition 13. $\langle \overrightarrow{C'}, K', \theta' \rangle$ is in $\rho_1(\langle \overrightarrow{C}, K, \theta \rangle)$ if and only if $\overrightarrow{C'} = c(\langle K', \theta' \rangle)$ and either*

1. *$K' = K \cup \{k\}$, $k \notin K$ and $\theta' = \theta$ or*
2. *$K' = K$, $\theta' = \theta\{y'/x'\}$ and $\{y'/x'\} \in \Theta$ where x' and y' are distinct variables in the k_1th and k_2th literals of $\overrightarrow{\top}$ respectively and k_1th and k_2th are in K'.*

In Definition 14, ρ_1 adds a most general literal from $\overrightarrow{\top}$ which has not been added before (item 1) or it applies an elementary variable substitution such that the clause subsumes $\overrightarrow{\bot}$ (item 2). We show that ρ_1 is ideal for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$. The completeness proof below is similar to the completeness proof for Laird's refinement operator [8,13] adopted for subsumption order relative to $\bot$.

Lemma 1. *Let $\overrightarrow{C}$, $\overrightarrow{D}$ be two ordered clauses in $\overrightarrow{\mathcal{L}}_\bot$ such that $\overrightarrow{C}\theta = \overrightarrow{D}$ for some substitution θ. Then, there exists a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{D}$.*

Proof. Suppose $\overrightarrow{C}$, $\overrightarrow{D}$ are ordered clauses and $\overrightarrow{C}\theta = \overrightarrow{D}$. Then according to Definition 8, $\overrightarrow{C}$ and $\overrightarrow{D}$ have the same predicate symbols at the same positions and therefore can be regarded as atoms. We need to show that there exists a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{D}$ by repeatedly selecting step 2 in Definition 14. The proof is then similar to the proof of this lemma for atoms (Theorem 4 in [10]). □

Lemma 2. *Let $\overrightarrow{C}$, $\overrightarrow{D}$ be two ordered clauses in $\overrightarrow{\mathcal{L}}_\bot$ such that $\overrightarrow{C}$ is a subsequence of $\overrightarrow{D}$ relative to $\bot$. Then, there exists a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{D}$.*

Proof. The proof is by induction on i the number of literals in $\overrightarrow{D}$ but not in $\overrightarrow{C}$. If $i = 0$ then $\overrightarrow{C} = \overrightarrow{D}$, and the empty chain satisfies the lemma. Assume for some j, $0 \leq j < i$, the lemma is true. This implies that there is a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{C}_j$ such that $\overrightarrow{C}_j$ is $\overrightarrow{C}$ with j literals inserted such that $\overrightarrow{C}_j$ is a subsequence of $\overrightarrow{D}$ relative to $\perp$. We show that there is a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{C}_{j+1}$. Let l be the leftmost literal in $\overrightarrow{D}$ which is not in $\overrightarrow{C}_j$. Given that $\overrightarrow{D} \in \overrightarrow{\mathcal{L}}_\perp$ we can assume that l is mapped to the k-th literal of $\perp$. We consider the following two cases: (a) if l is a most general literal with respect to $\overrightarrow{C}_j$, then l is the k-th literal of $\overrightarrow{\top}$ and using item 1 in the definition of ρ_1, $\langle \overrightarrow{C}_{j+1}, K', \theta \rangle \in \rho_1(\langle \overrightarrow{C}_j, K, \theta \rangle)$, where $K' = K \cup \{k\}$. (b) otherwise there is a most general literal l' such that $l'\theta' = l$. In this case, first using item 1 in the definition of ρ_1, $\langle \overrightarrow{C'}_{j+1}, K', \theta \rangle \in \rho_1(\langle \overrightarrow{C}_j, K, \theta \rangle)$ and then according to Lemma 1 (and using item 2 in the definition of ρ_1), $\langle \overrightarrow{C}_{j+1}, K', \theta'' \rangle \in \rho_1^*(\langle \overrightarrow{C'}_{j+1}, K', \theta \rangle)$, where $K' = K \cup \{k\}$ and $\theta'' = \theta\theta'$. Thus, in both cases (a) and (b), there exists a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{C}_{j+1}$ and this completes the proof. □

Theorem 3. *ρ_1 is complete for $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$.*

Proof. Let $\overrightarrow{C}$, $\overrightarrow{D}$ be two ordered clauses in $\overrightarrow{\mathcal{L}}_\perp$ such that, for some θ, $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{D}$ relative to $\perp$. If we define $\overrightarrow{E} = \overrightarrow{C}\theta$ then $\overrightarrow{E}$ and $\overrightarrow{C}$ satisfy Lemma 1, hence there is a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{E}$. $\overrightarrow{E}$ is a subsequence of $\overrightarrow{D}$ relative to $\perp$ and according to Lemma 2, there is a ρ_1-chain from $\overrightarrow{E}$ to $\overrightarrow{D}$. Thus, there is a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{D}$ via $\overrightarrow{E}$. □

According to Definition 14, the refinement operator ρ_1 works on an encoding of a clause, i.e. $\langle K, \theta \rangle$ rather than the clause itself. In the following we define the order relation for the encoding tuples $\langle K, \theta \rangle$, used in the mapping function c. Then, we show that the mapping function c is order-embedding.

Definition 15. *Let $\mathcal{K}$ and Θ be defined as in Definition 13 and $K_1, K_2 \in \mathcal{K}$ and $\theta_1, \theta_2 \in \Theta$. $\langle K_1, \theta_1 \rangle \subseteq \langle K_2, \theta_2 \rangle$ if and only if $K_1 \subseteq K_2$ and $\theta_1 \subseteq \theta_2$. $\theta_1 \subseteq \theta_2$ if and only if there exists a substitution θ such that $\theta_2 = \theta_1\theta$. $\langle K_1, \theta_1 \rangle \sim \langle K_2, \theta_2 \rangle$ if and only if $\langle K_1, \theta_1 \rangle \subseteq \langle K_2, \theta_2 \rangle$ and $\langle K_2, \theta_2 \rangle \subseteq \langle K_1, \theta_1 \rangle$.*

Theorem 4. *Let $\mathcal{K}$ and Θ and mapping function c be defined as in Definition 13 and $K_1, K_2 \in \mathcal{K}$ and $\theta_1, \theta_2 \in \Theta$. $c(\langle K_1, \theta_1 \rangle) \succeq_\perp c(\langle K_2, \theta_2 \rangle)$ if and only if $\langle K_1, \theta_1 \rangle \subseteq \langle K_2, \theta_2 \rangle$.*

Proof. $\Rightarrow$: Let $\overrightarrow{C}$, $\overrightarrow{D}$ be ordered clauses such that $\overrightarrow{C} = c(\langle K_1, \theta_1 \rangle)$ and $\overrightarrow{D} = c(\langle K_2, \theta_2 \rangle)$. Assume $\overrightarrow{C} \succeq_\perp \overrightarrow{D}$, then according to Theorem 3 there is a ρ_1-chain from $\overrightarrow{C}$ to $\overrightarrow{D}$. Let this ρ_1-chain be $C = C'_0 \succeq_\perp C'_1 \succeq_\perp \cdots \succeq_\perp C'_m = D$ where $\langle \overrightarrow{C'}_{i+1}, K'_{i+1}, \theta'_{i+1} \rangle \in \rho_1(\langle \overrightarrow{C'}_i, K'_i, \theta'_i \rangle)$, $0 \leq i < m$. According to the definition of ρ_1, in each refinement step either 1) $K'_i \subseteq K'_{i+1}$ and $\theta'_{i+1} = \theta'_i$ or 2) $K'_{i+1} = K'_i$ and $\theta'_i \subseteq \theta'_{i+1}$. Then it is always the case that $K'_i \subseteq K'_{i+1}$ and $\theta'_i \subseteq \theta'_{i+1}$, where $K'_0 = K_1, K'_m = K_2, \theta'_0 = \theta_1, \theta'_m = \theta_2$. Thus, $K_1 \subseteq K_2$ and $\theta_1 \subseteq \theta_2$.

$\Leftarrow$: Let $\overrightarrow{C} = c(K_1, \theta_1) = (\bigvee l_i | i \in K_1)\theta_1$ and $\overrightarrow{D} = c(K_2, \theta_2) = (\bigvee l_j | j \in K_2)\theta_2$ such that $K_1 \subseteq K_2$ and $\theta_1 \subseteq \theta_2$. According to Definition 15, $\theta_2 = \theta_1\theta$ for some substitution θ. Then, given $K_1 \subseteq K_2$, for every literal $l_i\theta_1$ from $\overrightarrow{C}\theta$, we have a literal $l_i\theta_1\theta$ from $\overrightarrow{D}$ where $l_i\theta_1$ and $l_i\theta_1\theta$ are both mapped to the same literal l_i from $\overrightarrow{\top}$ (Definition 13). Thus, $\overrightarrow{C}\theta$ is a subsequence of $\overrightarrow{D}$ relative to $\bot$ and therefore $\overrightarrow{C} \succeq_\bot \overrightarrow{D}$. □

In the following we show the properness and the idealness of ρ_1 for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$.

Lemma 3. *Let $\overrightarrow{C}$ and $\overrightarrow{D}$ be ordered clauses. $\overrightarrow{C} \sim_\bot \overrightarrow{D}$ if and only if $\overrightarrow{C}$ and $\overrightarrow{D}$ are alphabetical variants.*

Proof. $\Rightarrow$: Suppose $\overrightarrow{C} \sim_\bot \overrightarrow{D}$, then we have $\overrightarrow{C} \succeq_\bot \overrightarrow{D}$ and $\overrightarrow{D} \succeq_\bot \overrightarrow{C}$. Thus, there are substitutions θ_1 and θ_2 such that $\overrightarrow{C}\theta_1$ is a subsequence of $\overrightarrow{D}$ relative to $\bot$ and $\overrightarrow{D}\theta_2$ is a subsequence of $\overrightarrow{C}$ relative to $\bot$. Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_l$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$. Therefore, there are strictly increasing selection functions s_1 and s_2 such that for each $(i, j) \in s_1$, $L_i\theta_1 = M_j$ and for each $(i, j) \in s_2$, $M_i\theta_2 = L_j$. Given that s_1 and s_2 are strictly increasing functions, there is a one-to-one mapping between literals of $\overrightarrow{C}$ and $\overrightarrow{D}$ such that $m = n$, $L_i\theta_1 = M_i$ and $M_i\theta_2 = L_i$. Therefore it holds that $\overrightarrow{C}\theta_1 = \overrightarrow{D}$ and $\overrightarrow{D}\theta_2 = \overrightarrow{C}$. Hence, $\overrightarrow{C}$ and $\overrightarrow{D}$ are alphabetical variants.

$\Leftarrow$: Suppose $\overrightarrow{C}$ and $\overrightarrow{D}$ are alphabetical variants. Therefore there are substitutions θ_1 and θ_2 such $\overrightarrow{C}\theta_1 = \overrightarrow{D}$ and $\overrightarrow{D}\theta_2 = \overrightarrow{C}$. Then it follows from Definition 12 that $\overrightarrow{C} \succeq_\bot \overrightarrow{D}$ and $\overrightarrow{D} \succeq_\bot \overrightarrow{C}$ and therefore $\overrightarrow{C} \sim_\bot \overrightarrow{D}$. □

Lemma 4. *Let $\mathcal{K}$ and Θ and mapping function c be defined as in Definition 13 and $K, \{k\} \in \mathcal{K}$ such that $k \notin K$ and $\theta \in \Theta$. Then, $c(\langle K \cup \{k\}, \theta \rangle) \succ_\bot c(K, \theta)$.*

Proof. Suppose $c(\langle K \cup \{k\}, \theta \rangle) \not\succ_\bot c(K, \theta)$. We know from Theorem 4 that $c(\langle K \cup \{k\}, \theta \rangle) \succeq_\bot c(K, \theta)$, and therefore $c(\langle K \cup \{k\}, \theta \rangle) \sim_\bot c(K, \theta)$. According to Lemma 3, $c(\langle K \cup \{k\}, \theta \rangle)$ and $c(K, \theta)$ must be alphabetical variants, contradicting $k \notin K$. Thus, $c(\langle K \cup \{k\}, \theta \rangle) \succ_\bot c(K, \theta)$. □

Lemma 5. *Let $\mathcal{K}$ and Θ, $\top$ and mapping function c be defined as in Definition 13 and $K \in \mathcal{K}$, $\{y/x\}, \theta \in \Theta$ where x and y are distinct variables in the k_1th and k_2th literals of $\overrightarrow{\top}$ respectively and k_1th and k_2th are in K. Then, $c(\langle K, \theta\{y/x\} \rangle) \succ_\bot c(K, \theta)$.*

Proof. Suppose $c(\langle K, \theta\{y/x\} \rangle) \not\succ_\bot c(K, \theta)$. We know from Theorem 4 that $c(\langle K, \theta\{y/x\} \rangle) \succeq_\bot c(K, \theta)$, and therefore $c(\langle K, \theta\{y/x\} \rangle) \sim_\bot c(K, \theta)$. According to Lemma 3, $c(\langle K, \theta\{y/x\} \rangle)$ and $c(K, \theta)$ must be alphabetical variants. Thus, $\{y/x\}$ must be a renaming subsumption, i.e. x is either equal to y or it does not occur in $c(K, \theta)$, contradicting the assumption. Thus, $c(\langle K, \theta\{y/x\} \rangle) \succ_\bot c(K, \theta)$. □

Theorem 5. *ρ_1 is proper for $\langle \overrightarrow{\mathcal{L}}_\bot, \succeq_\bot \rangle$.*

Proof. If $\langle C', \theta', K' \rangle \in \rho_1(\langle C, \theta, K \rangle)$ is generated by item 1 in the definition of ρ_1, then $\overrightarrow{C} \succ_\perp \overrightarrow{D}$ follows from Lemma 4. If it is generated by item 2 in the definition of ρ_1, then $\overrightarrow{C} \succ_\perp \overrightarrow{D}$ follows from Lemma 5. □

Theorem 6. *ρ_1 is ideal for $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$.*

Proof. Locally finiteness follows from the definition of ρ_1 and the fact that there are finite number of literals and variables in $\perp$. Completeness and properness were proved in Theorem 3 and Theorem 5 respectively. □

In the following we study the morphism between $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$ and $\langle \mathcal{K} \times \Theta, \subseteq \rangle$. According to Theorem 4, the mapping function c is an order-embedding. The following theorem shows that c is also an order-isomorphism.

Theorem 7. *The mapping function $c : \mathcal{K} \times \Theta \rightarrow \overrightarrow{\mathcal{L}}_\perp$ as defined in Definition 13 is an order-isomorphism.*

Proof. According to Theorem 4, the mapping function c is an order-embedding, so we only need to prove that c is onto. Let $\overrightarrow{C}$ be an ordered clause in $\overrightarrow{\mathcal{L}}_\perp$, then according to Definition 10, there exist substitution θ and selection function s such that $\overrightarrow{C}\theta \sqsubseteq^s \overrightarrow{\perp}$. From Definition 13 we have $\overrightarrow{\top}\theta_\top = \overrightarrow{\perp}$ and therefore $\overrightarrow{C}\theta \sqsubseteq^s \overrightarrow{\top}\theta_\top$ and this implies $\overrightarrow{C} \sqsubseteq^s \top\theta_\top\theta^{-1}$. Thus, $\overrightarrow{C}$ can be defined as $\overrightarrow{C} = c(K, \theta') = (\bigvee l_i | i \in K)\theta'$, where $\theta' = \theta_\top\theta^{-1}$ and K is the range of the selection function s. □

The proposition below follows directly from Theorem 7.

Proposition 1. *Let $\mathcal{K}$ and Θ and mapping function c be defined as in Definition 13 and $K_1, K_2 \in \mathcal{K}$ and $\theta_1, \theta_2 \in \Theta$. $c(K, \theta) \sim_\perp c(K', \theta')$ if and only if $\langle K, \theta \rangle \sim \langle K', \theta' \rangle$.*

It is known that if a mapping is order-isomorphism it is also a lattice isomorphism.

Theorem 8 ([3]). *Let $\langle L, \wedge, \vee \rangle$ and $\langle K, \cap, \cup \rangle$ be lattices and $f : L \rightarrow K$. f is order-isomorphism if and only if it is a lattice isomorphism.*

According to this theorem the mapping c is a lattice isomorphism. Thus, it can be shown that $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$ and $\langle \mathcal{K} \times \Theta, \subseteq \rangle$ are two isomorphic lattices. This also means that the mapping c is a lattice homomorphism. The proposition below follows from c being a homomorphism.

Proposition 2. *Let $\mathcal{K}$ and Θ and mapping function c be defined as in Definition 13 and $K_1, K_2 \in \mathcal{K}$ and $\theta_1, \theta_2 \in \Theta$. Mapping c is joint-preserving and meet-preserving that is:*

1. *$lgg_\perp(c(\langle K_1, \theta_1 \rangle, c(\langle K_2, \theta_2 \rangle)) = c(\langle K_1 \cap K_2, \theta_1 \cap \theta_2 \rangle)$*
2. *$mgi_\perp(c(\langle K_1, \theta_1 \rangle, c(\langle K_2, \theta_2 \rangle)) = c(\langle K_1 \cup K_2, \theta_1 \cup \theta_2 \rangle)$*

According to Proposition 2, the least general generalisation ($lgg_\perp$) and the most specific instance ($mgi_\perp$) for $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$ can be defined based on the the joint and the meet operations for $\langle \mathcal{K} \times \Theta, \subseteq \rangle$. Note that if two lattices are isomorphic then for practical purposes they are identical and differ only in the notation of their elements. The morphism between $\langle \overrightarrow{\mathcal{L}}_\perp, lgg_\perp, mgi_\perp \rangle$ and $\langle \mathcal{K} \times \Theta, \cap, \cup \rangle$ is important from a practical point of view. For example, the least general generalisation (lgg) of clauses, introduced by Plotkin [9], is an important operator for ILP. However, in the general subsumption order the construction of lgg can be inefficient (e.g. the cardinality of the lgg of two clauses is bounded by the product of the cardinalities of the two clauses). On the other hand, efficient operators can be implemented for least generalisation and greatest specialisation in the sequential subsumption order relative to a bottom clause.

6 Subsumption Order Relative to $\perp$

The purpose of the previous sections was to characterise Progol's refinement and the subsumption sub-lattice which is searched by Progol. We defined sequential subsumption order relative to $\perp$ and studied the properties of this special case of subsumption. In this section we show how some limitations of Progol's refinement operator can be addressed by relaxing conditions of sequential subsumption. In particular we address the first type of incompleteness, which is believed to be more problematic than the second type of incompleteness[1]. In this section we define a refinement operator which is less restricted than ρ_1. As demonstrated in section 3, the first type of Progol's refinement incompleteness is due to the choice of ordering of literals in $\perp$ and the fact that clauses are considered as subsequences of $\perp$. This condition was embedded in the definitions of the sequential subsumption and the refinement operator ρ_1. However, more relaxed conditions can be defined for subsumption and refinement operators relative to $\perp$. Note that in the previous definitions and theorems we only needed to assume that the selection functions are injective so that we can encode every literal of a clause by a k index from $\perp$. Therefore a less restricted ordering can be defined by using a selection function which is injective rather than strictly increasing.

Definition 16 (Ordered subset). *Let $\overrightarrow{C} = L_1 \vee L_2 \vee \cdots \vee L_l$ and $\overrightarrow{D} = M_1 \vee M_2 \vee \cdots \vee M_m$ be ordered clauses. $\overrightarrow{C}$ is an ordered subset of $\overrightarrow{D}$, denoted by $\overrightarrow{C} \subseteq^s \overrightarrow{D}$, if there exists an injective selection function s such that for each $(i, j) \in s$, $L_i = M_j$.*

By choosing s to be an injective function, we make sure that clauses can still be encoded by a set of k indexes. However, these clauses do not need to follow the same order as literals in $\perp$. In the following, we give new definitions for the mapping function and the refinement operator for this less restricted subsumption order.

Definition 17 **($\mathcal{L}_\perp$).** *Let $\overrightarrow{\perp}$ be the bottom clause as defined in Definition 2. $\overrightarrow{C}$ is in $\mathcal{L}_\perp$ if and only if there exists a substitution θ such that $\overrightarrow{C}\theta$ is an ordered subset of $\overrightarrow{\perp}$.*

Definition 18 **(Mapping function c').** *Let $\overrightarrow{\perp}$ and $\mathcal{L}_\perp$ be as defined in Definition 17, n be the number of literals in $\perp$. Let $\overrightarrow{\top}$, $\theta_\top$, θ, Θ, $\mathcal{K}$ be as defined in Definition 13. The mapping function $c' : \mathcal{K} \times \Theta \rightarrow \mathcal{L}_\perp$ is defined as follows:*

$$c'(\langle K, \theta\rangle) = (\bigvee_{i \in K} l_i | l_i \textit{ is the ith literal of } \overrightarrow{\top})\theta.$$

In the definition of c', unlike in c, literals l_i do not need to follow the same order as literals in $\overrightarrow{\top}$. In the following we define a refinement operator, ρ_2, which is similar to ρ_1 but uses the mapping function c' instead of c.

Definition 19 **(ρ_2).** *Let $\overrightarrow{\perp}$ and $\mathcal{L}_\perp$ be as defined in Definition 17, $\overrightarrow{C}$ be an ordered clause in $\mathcal{L}_\perp$, n be the number of literals in $\perp$, k be a natural number, $1 \leq k \leq n$, $\overrightarrow{\top}$, Θ and $\mathcal{K}$ be defined as in Definition 18. Let $K \in \mathcal{K}$, $\theta \in \Theta$, $\overrightarrow{C} = c'(\langle K, \theta\rangle)$ and the mapping function c' be defined as in Definition 18. $\langle \overrightarrow{C'}, K', \theta'\rangle$ is in $\rho_2(\langle \overrightarrow{C}, K, \theta\rangle)$ if and only if $\overrightarrow{C'} = c'(\langle K', \theta'\rangle)$ and either*

1. *$K' = K \cup \{k\}$, $k \notin K$ and $\theta' = \theta$ or*
2. *$K' = K$, $\theta' = \theta\{y'/x'\}$ and $\{y'/x'\} \in \Theta$ where x' and y' are distinct variables in the k_1th and k_2th literals of $\overrightarrow{\top}$ respectively and k_1th and k_2th are in K'.*

The following example demonstrates how the first type of incompleteness (in Example 1) is addressed in ρ_2.

Example 7. Let $\overrightarrow{C}$ and $\overrightarrow{\perp}$ be as defined in Example 1. Progol's refinement cannot generate C (i.e. $C \notin \rho^*(\Box)$)) and also $\langle \overrightarrow{C}, K, \theta\rangle \notin \rho_1^*(\langle \Box, \emptyset, \emptyset\rangle)$. However, Table 1 shows that $\langle \overrightarrow{C}, K, \theta\rangle \in \rho_2^*(\langle \Box, \emptyset, \emptyset\rangle)$, where $K = \{1, 2, 5, 3\}$ and $\theta = \{V_4/V_1, V_{12}/V_5, V_{13}/V_2, V_6/V_{14}, V_7/V_2\}$ and $\overrightarrow{\top}$ is the clause:

$$\begin{aligned}\mathrm{mult}(V_1, V_2, V_3) \leftarrow \mathrm{dec}(V_4, V_5), \mathrm{plus}(V_6, V_7, V_8), \mathrm{plus}(V_9, V_{10}, V_{11}),\\ \mathrm{mult}(V_{12}, V_{13}, V_{14}), \mathrm{mult}(V_{15}, V_{16}, V_{17}).\end{aligned} \qquad \diamond$$

This example shows that ρ_2 can address the incompleteness of ρ demonstrated in Example 1. However, ρ_2 is also more redundant than ρ (e.g. different permutations of the same clause could be generated). On the other hand, as mentioned in Section 3, a refinement operator cannot be both complete and non-redundant. Given that in the new definitions the selection functions are injective, we can encode every literal of a clause by a k index from $\perp$. Therefore, the properties mentioned in Section 5 for the mapping function c also hold for c'. By different conditions on the selection functions in Definition 16 we can get different kind of subsumption orders. For example, if the selection function is monotonically increasing then we will have a subsumption order which allows each literal of $\perp$

Table 1. Application of ρ_2 in Example 7

C'	θ'	K'
$\Box$	$\emptyset$	$\emptyset$
$mult(V_1, V_2, V_3) \leftarrow$	$\emptyset$	$\{1\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_4, V_5)$	$\emptyset$	$\{1, 2\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5)$	$\{V_4/V_1\}$	$\{1, 2\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_{12}, V_{13}, V_{14})$	$\{V_4/V_1\}$	$\{1, 2, 5\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_5, V_{13}, V_{14})$	$\{V_4/V_1, V_{12}/V_5\}$	$\{1, 2, 5\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_5, V_2, V_{14})$	$\{V_4/V_1, V_{12}/V_5, V_{13}/V_2\}$	$\{1, 2, 5\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_5, V_2, V_{14}), plus(V_6, V_7, V_8)$	$\{V_4/V_1, V_{12}/V_5, V_{13}/V_2\}$	$\{1, 2, 5, 3\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_5, V_2, V_{14}), plus(V_{14}, V_7, V_8)$	$\{V_4/V_1, V_{12}/V_5, V_{13}/V_2, V_6/V_{14}\}$	$\{1, 2, 5, 3\}$
$mult(V_1, V_2, V_3) \leftarrow dec(V_1, V_5), mult(V_5, V_2, V_{14}), plus(V_{14}, V_2, V_8)$	$\{V_4/V_1, V_{12}/V_5, V_{13}/V_2, V_6/V_{14}, V_7/V_2\}$	$\{1, 2, 5, 3\}$

to be selected more than once[3]. This will address the second type of Progol's incompleteness mentioned before. However, the selection functions are not injective and therefore the encoding and the morphism we described in this paper are not applicable.

7 Related Work and Discussion

Progol's refinement operator and its incompleteness with respect to the general subsumption order were initially discussed in [7]. The purpose of the present paper was to characterising Progol's refinement space and to give an analysis of refinement operators for this space. In a previous attempt, the authors of [1] suggested weak subsumption for characterising Progol's refinement space. However, as we have shown in this paper, weak subsumption cannot capture all aspects of Progol's refinement. They also considered refinement operators which, as the operators considered in this paper, are based on Laird's operator. In this paper we used an encoding of clauses with respect to a bottom clause. In this encoding each clause is represented by a tuple $\langle K, \theta \rangle$ and it can be constructed from $\overrightarrow{\top}$ as described in Definition 13. This idea was first used in [12] where the substitution θ is encoded as a binding matrix which maps the variables of $\overrightarrow{\top}$ to the variables of a clause with respect to the bottom clause. The morphism between the lattice of variable bindings and the subsumption lattice was also studied in [12]. A subsumption relation for ordered clauses (i.e. ordered subsumption) is studied in [4]. It is shown that, in the defined subsumption, the least generalisation of two ordered clauses does not exist and that the subsumption testing for ordered clauses is NP-complete. The subsequence relation considered in [4], assumes a mapping function which is monotonically increasing (rather than strictly increasing). As mentioned in the previous section, this leads to a different subsumption from the one considered in this paper (i.e. sequential subsumption) and the results from this paper are not applicable.

[3] In this case, Definition 16 will be identical to the definition of subsequence considered in [4].

8 Conclusions

In this paper we have studied refinement operators for a hypotheses space bounded by a most specific (bottom) clause. We introduced a subsumption order relative to a bottom clause and demonstrated how Progol's refinement can be characterised with respect to this order. We also proved that ideal refinement operators exist for this order. It was shown that efficient operators can be defined for least generalisation and greatest specialisation in the subsumption order relative to a bottom clause. The theoretical results presented in this paper can be applied to ILP systems which use Inverse Entailment (IE) as well as any other ILP system which uses a bottom clause to restrict the search space.

Acknowledgments

We would like to thank the three anonymous reviewers for their comments. The first author was supported by the BBSRC grant BB/C519670/1. The second author would like to thank the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair.

References

1. Badea, L., Stanciu, M.: Refinement operators can be (weakly) perfect. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 21–32. Springer, Heidelberg (1999)
2. Chang, C., Lee, R.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, London (1973)
3. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press, Cambridge (2002)
4. Kuwabara, M., Ogawa, T., Hirata, K., Harao, M.: On generalization and subsumption for ordered clauses. In: Proceedings of the JSAI 2005 Workshops, pp. 212–223 (2006)
5. Laird, P.D.: Learning from good data and bad. PhD thesis, Yale University (1987)
6. Lee, S.D., De Raedt, L.: Constraint Based Mining of First Order Sequences in SeqLog. Database Support for Data Mining Applications, 155–176 (2003)
7. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
8. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. LNCS (LNAI), vol. 1228. Springer, Heidelberg (1997)
9. Plotkin, G.D.: Automatic Methods of Inductive Inference. PhD thesis, Edinburgh University (August 1971)
10. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 5, pp. 135–151. Edinburgh University Press, Edinburgh (1969)
11. Srinivasan, A.: The Aleph Manual. University of Oxford, Oxford (2007)

12. Tamaddoni-Nezhad, A., Muggleton, S.H.: Searching the subsumption lattice by a genetic algorithm. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 243–252. Springer, Heidelberg (2000)
13. van der Laag, P.: An Analysis of Refinement Operators in Inductive Logic Programming. Tinbergen Institute Research Series, Rotterdam (1995)

A Progol's Refinement Operator ρ_0

The following definition describes a bottom clause $\bot_i$ for a depth-bounded mode language $\mathcal{L}_i(M)$ as defined in [7]. In this paper, we refer to $\bot_i$ and $\mathcal{L}_i(M)$ as $\bot$ and $\mathcal{L}$ respectively.

Definition 20. Most-specific clause $\bot_i$. *Let h, i be natural numbers B be a set of Horn clauses, $e = a \leftarrow b_1, .., b_n$ be a definite clause, M be a set of mode declarations containing exactly one modeh m such that $a(m) \succeq a$ and $\bot$ be the most-specific (potentially infinite) definite clause such that $B \wedge \bot \wedge \overline{e} \vdash_h \Box$. $\bot_i$ is the most-specific clause in $\mathcal{L}_i(M)$ such that $\bot_i \succeq \bot$.*

The refinement operator ρ defined in [7] allows more than one literal in a clause to be mapped to the same literal in $\bot$. However, in Progol's implementation of the refinement operator, index k is incremented after each step for the sake of efficiency. This means each literal of $\bot$ can be considered only once. In the following, we give a revised definition (ρ_0) which describes the refinement operator as implemented in Progol. This also includes a revised definition for function δ.

Definition 21. Progol refinement operator ρ_0. *Let h, i, B, e, M and $\bot_i$ be defined as in Definition 20 and let n be the cardinality of $\bot_i$. Let k be a natural number, $1 \leq k \leq n$. Let C be a clause in $\mathcal{L}_i(M)$ and θ be a substitution such that $C\theta \subseteq \bot_i$. Below a literal l corresponding to a mode m_l in M is denoted simply as $p(v_1, .., v_m)$ despite the sign of m_l and function symbols in $a(m_l)$. A variable is splittable if it corresponds to a +type or -type in a modeh or if it corresponds to a -type in a modeb. $\langle C', \theta', k' \rangle$ is in $\rho_0(\langle C, \theta, k \rangle)$ if and only if either*

1. *$C' = C \vee l$, $k' = k + 1$, $k < n$ and $\langle l, \theta' \rangle$ is in $\delta(\theta, k)$ and $C' \in \mathcal{L}_i(M)$ or*
2. *$C' = C$, $k' = k + 1$, $\theta' = \theta$ and $k < n$.*

$\langle p(v_1, .., v_m), \theta'_m \rangle$ is in $\delta(\theta, k)$ if and only if $l_k = p(u_1, .., u_m)$ is the kth literal of $\bot_i$, $\theta'_0 = \theta$ and θ'_j for each j, $1 \leq j \leq m$ is defined as follows:

1. *if $v_j/u_j \in \theta'_{j-1}$ then $\theta'_j = \theta'_{j-1}$ or*
2. *if u_j is splitable then $\theta'_j = \theta'_{j-1} \cup \{v_j/u_j\}$ where v_j is a new variable not in dom(θ'_{j-1}).*

In Definition 21, the refinement operator ρ_0, as ρ in [7], is defined for clauses in $\mathcal{L}_i(M)$. However, ρ_0 can be also defined for clauses in $\overrightarrow{\mathcal{L}}_\bot$ if we let C and C' to be ordered clauses in $\overrightarrow{\mathcal{L}}_\bot$ and $\overrightarrow{C}\theta$ be a subsequence of the bottom clause (rather than a subset). In this case, it can be shown that ρ_0 is weakly complete for $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$.

Theorem 2. ρ_0 is weakly complete for $\langle \overrightarrow{\mathcal{L}}_\perp, \succeq_\perp \rangle$.

Sketch proof. We need to show that $\rho_0^*(\langle \Box, \emptyset, 1\rangle) = \overrightarrow{\mathcal{L}}_\perp$. We show that for each $\overrightarrow{C} \in \overrightarrow{\mathcal{L}}_\perp$, there exists a ρ_0-chain from $\Box$ to $\overrightarrow{C'}$ where $\overrightarrow{C'}$ and $\overrightarrow{C}$ are alphabetical variants. The proof is by induction on i, the number of literals in $\overrightarrow{C}$. If $i = 0$ then $\overrightarrow{C} = \Box$, and the empty chain satisfies the theorem. Assume for some j, $0 \leq j < i$, that the lemma is true. This implies that there is a ρ_0-chain from $\Box$ to $\overrightarrow{C}_j$ such that $\overrightarrow{C}_j$ is an ordered clause in $\overrightarrow{\mathcal{L}}_\perp$ with j literals added from $\overrightarrow{C}$. Therefore, there is a substitution θ such that $\overrightarrow{C}_j\theta$ is a subsequence of $\perp$ and we assume that the j-th literal of $\overrightarrow{C}_j$ is mapped to the k-th literal of $\perp$. Let $\overrightarrow{C}_{j+1} = \overrightarrow{C}_j \vee l$, where l is the leftmost literal of $\overrightarrow{C}$ which is not in $\overrightarrow{C}_j$ and l is mapped to the k'-th literal of $\perp$, where $k < k'$ (because $\overrightarrow{C}_j$ and $\overrightarrow{C}_{j+1}$ are sequential generalisations of $\perp$). Then there exists a ρ_0-chain from $\langle \overrightarrow{C}_j, \theta, k\rangle$ to $\langle \overrightarrow{C}_j, \theta, k'\rangle$ by repeatedly selecting item 2 in the definition of ρ_0 in order to skip $k' - k$ literals of $\perp$. According to the definition of δ, there exists $\langle l', \theta'\rangle$ in $\delta(\theta, k')$ such that, by construction, l and l' are variants. Therefore, by selecting item 1 in the definition of ρ_0, $\overrightarrow{C'}_{j+1} = \overrightarrow{C}_j \vee l$ is a variant of $\overrightarrow{C}_{j+1} = \overrightarrow{C}_j \vee l$, where $\langle \overrightarrow{C'}_{j+1}, \theta', k' + 1\rangle \in \rho_0(\langle \overrightarrow{C}_j, \theta, k'\rangle)$. Thus, there is a ρ_0-chain from $\Box$ to a a variant of $\overrightarrow{C}_{j+1}$ and this completes the proof. $\Box$

Learning Aggregate Functions with Neural Networks Using a Cascade-Correlation Approach

Werner Uwents[1] and Hendrik Blockeel[1,2]

[1] Department of Computer Science, Katholieke Universiteit Leuven
[2] Leiden Institute of Advanced Computer Science, Leiden University

Abstract. In various application domains, data can be represented as bags of vectors. Learning functions over such bags is a challenging problem. In this paper, a neural network approach, based on cascade-correlation networks, is proposed to handle this kind of data. By defining special aggregation units that are integrated in the network, a general framework to learn functions over bags is obtained. Results on both artificially created and real-world data sets are reported.

1 Introduction and Context

In general, two types of approaches to relational data mining can be distinguished. In the first type, the propositionalization approach, each data element is *summarized* into a vector of fixed length. We will refer to the components of this vector as features. In the second type, the direct approach, no summarization is performed and structured data elements are handled directly. A search is performed through a large hypothesis space, which may contain for instance (sets of) logical clauses.

The hypotheses searched by a relational learner could themselves be considered features of the data, i.e., the search for a suitable hypothesis can be seen as a search in a feature space. For instance, in the context of ILP, typically the final model built is a set of clauses, where the clauses are learned one by one. The single clauses could be said to be boolean features, combined into a disjunction.

From this point of view, propositionalization approaches are in principle equally powerful as direct approaches. In practice, including a separate feature for each clause in the search space in the fixed-size vector is often not feasible because the feature space is too large and might even be infinite. Direct approaches can be seen as “lazy propositionalization” approaches: they perform a greedy search through the feature space, gradually constructing relevant features.

If we look at the kind of features that are constructed by a relational learner, an essential property of these features is that they map *sets* of objects to a single scalar value. Such functions are called aggregate functions and they play a key role in relational learners. For instance, in ILP, if we have a clause `happy_father(X) :- child(Y,X)`, the “feature” constructed is essentially of the form $\exists y : Child(y, x)$, which tests if the set of all y's related to x through the *Child* relation is empty or not.

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 315–329, 2008.

As has been pointed out by Blockeel and Bruynooghe [1], the features constructed by relational learning systems are generally of the form $\mathcal{F}(\sigma_C(S))$ with S a set of objects, C a condition defined over single objects, σ the selection operator from relational algebra, and $\mathcal{F}$ an aggregate function, which maps a set to a scalar. In the above clause, S is the set of children of x, C is true, and $\mathcal{F}$ is the "there exists" operator ($\exists$). For the clause `happy_father(X) :- child(Y,X), age(Y,A), A < 12`, S and $\mathcal{F}$ are the same, but C is now a condition on the age of the children.

Blockeel and Bruynooghe further pointed out that ILP systems typically construct structurally complex conditions but always use the same, trivial, aggregate function, namely $\exists$. The importance of using more complex aggregate functions has been recognized by many people [2,3]. Most systems that handle such complex aggregate functions follow the propositionalization approach. The reason for this is that the structure of the search space of features of the form $\mathcal{F}(\sigma_C(S))$ becomes much more complex and difficult to search when $\mathcal{F}$ can be something else than $\exists$ [4]. But to keep the propositionalization approach feasible, a limited set of $\mathcal{F}$ functions still needs to be used, and the number of different C considered for σ_C must remain limited. For instance, Krogel and Wrobel [3] allow a single attribute test in C, but no conjunctions.

Given the limitations of the propositionalization approach, it is useful to study how direct approaches could include aggregate functions other than $\exists$. More recently, methods for learning relevant features of the form $\mathcal{F}(\sigma_C(S))$ have been proposed. Vens, Van Assche et al. [5,6] proposed a random forest approach that avoids the problems of searching a complex-structured search space, while Vens [4] studied the monotonicity properties of features of the form $\mathcal{F}(\sigma_C(S))$ and showed how efficient refinement of such features is possible for the most commonly occurring aggregate functions.

In parallel, Uwents et al. studied to what extent subsymbolic concepts on relational data can be learned using neural network approaches. Recurrent neural networks were first proposed to learn the aggregate features, leading to the concept of relational neural networks [7]. While a regular network maps one input vector to an output vector, recurrent networks can map a sequence of input vectors to a single output vector. This property was exploited to handle sets of vectors, the elements of which were input in random order in the network.

From the explanation above, it is clear that learning aggregate features from sets is a crucial part of any relation learner. In this paper, we therefore focus on the subsymbolic learning of aggregate functions as such, without considering the possibility of having many different relationships. This means that the general relational learning setting is restricted to the situation where there is just a single one-to-many relationship. This relationship results in bags of vectors with associated target vectors. This resembles the multi-instance setting, because in multi-instance learning, one also deals with data sets containing a bag of vectors for each data instance. However, in multi-instance learning the hypothesis that should be learned has some restrictions. Each bag is classified as positive or negative but this classification can be reformulated in terms of a classification of the individual vectors in the bag. If all vectors are negative, the bag as a whole will also be classified as negative. If there is at least one positive vector, the bag will be classified as positive. Learning aggregate functions in general is more complicated because these restrictions do no longer apply and the class of possible aggregate functions is very broad and diverse. In this paper, we will consider the use of neural networks for learning this kind of general aggregate functions. For the multi-instance setting, the use of neural networks has been explored before by Ramon and De Raedt [9]. They make use of a softmax function in the network, similar to what will be used in our aggregate cascade-correlation networks as explained in section 2.2.

The remainder of the paper is structured as follows. In section 2 we review cascade-correlation networks and we adapt these networks so that they can represent aggregation functions, this by introducing a limited number of aggregation units in them. A training procedure for such networks is also described. In section 3 we experimentally evaluate the method, to conclude in section 4.

2 Aggregate Cascade-Correlation Networks

Cascade-correlation networks are a special kind of neural networks, constructed one unit at a time. In the next subsection, the original cascade-correlation algorithm will be discussed. After that, a number of new units, capable of aggregating, will be presented. These units will then be integrated in an adapted version of cascade-correlation, resulting in a network structure that can learn concepts with aggregation. The resulting networks are called aggregate cascade-correlation networks (ACCNs).

2.1 The Original Cascade-Correlation Network

The idea behind the original cascade-correlation algorithm [10] is to learn not only the weights, but also the structure of the network at the same time. This is done in a constructive way, meaning that only one neuron at a time is trained and then added to the network. One starts with a network without any hidden unit, and then hidden neurons are added, one by one, until some stopping criterion is satisfied. Once a hidden neuron has been added to the network, its weights remain fixed throughout the rest of the procedure. This also means that, besides the actual input vector, the output values of these existing hidden units can be used as extra inputs for any new hidden neuron. At the output, a linear function can be used. A schema of the network is shown in figure 1. For an input vector x_p, the output values $o_{p,k}$ of the network are then computed as

$$o_k(x_p) = \sum_{i=1}^{N} v_{k,i} x_{p,i} + \sum_{i=1}^{H} v_{k,i} h_i(x_p) + v_{k,N+H+1} \tag{1}$$

$$h_i(x_p) = \sigma(\xi_i(x_p)) \tag{2}$$

$$\xi_i(x_p) = \sum_{j=1}^{N+i} w_{i,j}\theta_{i,j}(x_p) \tag{3}$$

$$\theta_i(x_p) = [x_{p,1}, \ldots, x_{p,N}, h_1(x_p), \ldots, h_{i-1}(x_p), 1] \tag{4}$$

with N the number of inputs and H the number of hidden units, while $w_{i,j}$ and $v_{k,i}$ are the weights for the hidden and the output units.

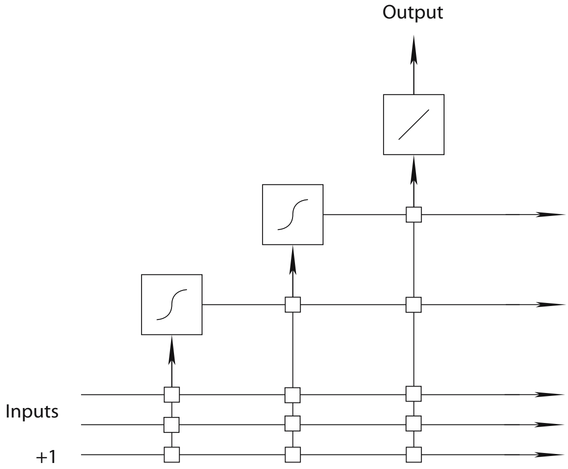


Fig. 1. Schema of the feedforward cascade-correlation network

The training of the network is done in two alternating phases. Before a new hidden neuron is added, its weights are trained while keeping the weights of all other hidden units fixed. This training is not done by minimizing the squared error between target and output, but by maximizing the correlation with the residual error. The residual error is defined as the difference between the actual target value and the output of the existing network, before adding the new neuron. Instead of the real correlation, a slightly different measure S is taken, in which some normalization factors are omitted and the absolute value is taken:

$$S = \sum_{k=1}^{K} |s_k| \tag{5}$$

$$s_k = \sum_{p=1}^{P} \left(h_{H+1}(x_p) - \overline{h_{H+1}(x_p)}\right) \left(e_k(x_p) - \overline{e_k(x_p)}\right) \tag{6}$$

$$e_k(x_p) = o_k(x_p) - t_{p,k} \tag{7}$$

with K the number of outputs, P the number of patterns in the training data, h_{H+1} the new candidate unit and t_p the target vector for pattern p. When this S value is maximized, the output of the new hidden neuron will correlate well with the residual error. The key idea here is that a unit that correlates well with the residual error, will help to reduce the output error when added to the network. The maximization is done by computing the gradient and performing some form of gradient descent. This gradient is computed as follows:

$$\frac{\partial S}{\partial w_i} = \sum_{k=1}^{K} \sum_{p=1}^{P} sign(s_k)\, \sigma'(\xi_{H+1}(x_p))\, (e_k(x_p) - \overline{e_k(x_p)})\, \theta_{H+1,i}(x_p) \tag{8}$$

Instead of training only one candidate neuron at a time, a pool of neurons, initialized with random weights, can be trained. At the end, the best one is selected. This increases the chance that a good candidate will be found. Once the best candidate is selected and added to the network, the output weights for the updated network can be trained. If a linear function is used at the outputs, the output weights can be obtained by simple linear regression.

2.2 Cascade-Correlation with Aggregation Units

The concept of cascade-correlation networks can be extended to networks for learning aggregate functions. The crucial difference is that instead of the simple hidden neurons, units that can process bags are used. For the rest, the network and the training of it works in the same way as for the feedforward networks. The data set now consists of bags $b_p = \{x_{p,1}, \ldots, x_{p,m}\}$, where $x_{p,i}$ are vectors of size N, with associated target vectors t_p. Because the input is a bag of vectors instead of one single vector, it can no longer be used as direct input for the output units, and so this is dropped from the equation. Each input bag $b_p = \{x_{p,1}, \ldots, x_{p,m}\}$ of a pattern p will be processed by the hidden units. Each time a vector of the bags has been processed, an intermediate output value for the hidden units can be computed, yielding a sequence of m values for each hidden unit. The final value is used by the output units, but the whole sequence of values can also be used by new hidden units. A schema of an aggregate cascade-correlation network for 2 input vectors is shown in figure 2. The network then becomes:

$$o_k(b_p) = \sum_{i=1}^{H} v_{k,i} h_{i,m}(b_p) + v_{k,H+1} \tag{9}$$

$$h_{i,j}(b_p) = g_i(\{\xi_i(x_{p,1}), \ldots, \xi_i(x_{p,j})\}) \tag{10}$$

$$\xi_i(x_{p,j}) = \sum_{l=1}^{N+i} w_{i,l}\theta_{i,l}(x_{p,j}) \tag{11}$$

$$\theta_i(x_{p,j}) = [x_{p,1}, \ldots, x_{p,N}, h_{1,j}(x_p), \ldots, h_{i-1,j}(x_p), 1] \tag{12}$$

with g_i a special function. Different functions can be used for g_i, as long as they perform some kind of aggregation. One could take the sum or the max function for instance. Another possibility is to use a locally recurrent neuron. The only condition imposed on the g_i function is that it must be derivable to allow gradient training. Different types of units can easily be combined in the

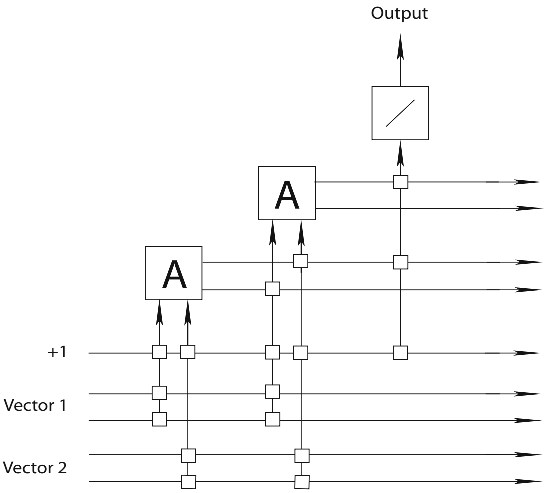


Fig. 2. Schema for an aggregate cascade-correlation network with 2 input vectors

network. For every new unit, several candidates of different types can be trained and the best one selected and added to the network. The four different types that will be considered here, are explained below.

1. The simplest type of unit is probably the *sum* unit:

$$g_i^{sum}(b_p) = \sum_{j=1}^{m} \sigma(\xi_i(x_{p,j})) \tag{13}$$

$$\frac{\partial}{\partial b_p} g_i^{sum}(b_p) = \sum_{j=1}^{m} \sigma'(\xi_i(x_{p,j})) \tag{14}$$

2. Somewhat more complicated is the max unit because the maximum function is not easily derivable. To circumvent this, one could use the softmax function. This is the same approach as in [9]. The *smx* unit is then defined as:

$$g_i^{smx}(b_p) = \frac{1}{C} \log \left(\sum_{j=1}^{m} e^{C \cdot \sigma(\xi_i(x_{p,j}))} \right) \tag{15}$$

$$\frac{\partial}{\partial b_p} g_i^{smx}(b_p) = \sum_{j=1}^{m} \frac{e^{C \cdot \sigma(\xi_i(x_{p,j}))}}{\sum_{l=1}^{m} e^{C \cdot \sigma(\xi_i(x_{p,l}))}} \sigma'(\xi_i(x_{p,j})) \tag{16}$$

C is a constant and the larger the value of C, the closer this function will approximate the real max function.

3. Instead of choosing a value for C that is large enough to give a good approximation, one could take the limit for C going to infinity, which gives the real max function and its derivative:

$$max\{x_1, \ldots, x_n\} = \lim_{C \to \infty} \frac{1}{C} \sum_{i=1}^{n} e^{C \cdot x_i} \tag{17}$$

$$\frac{\partial}{\partial x_j} max\{x_1, \ldots, x_n\} = \begin{cases} 1 & \text{if } x_j = max\{x_1, \ldots, x_n\} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

A possible disadvantage with this exact function is that the derivative is not continuous anymore, which can deteriorate learning. The corresponding *max* unit then becomes:

$$g_i^{max}(b_p) = max(\{\sigma(\xi_i(x_{p,1})), \ldots, \sigma(\xi_i(x_{p,k}))\}) \tag{19}$$

$$\frac{\partial}{\partial b_p} g_i^{smx}(b_p) = \sigma'(\xi_i(x_{p,l^*})) \quad \text{with } \sigma(\xi_i(x_{p,l^*})) = g_i^{max}(b_p) \tag{20}$$

4. The fourth type of unit considered here, is the locally recurrent unit. This is a standard feedforward unit with one recurrent connection with itself added. This *lrc* unit is defined as:

$$g_i^{lrc}(b_p) = \sigma(\xi_i(x_{p,k}) + w_r g_i^{lrc}(b_p \backslash \{x_{p,k}\})) \tag{21}$$

Gradient computation for this type of unit is done using backpropagation through time [11].

New types of units could easily be invented if necessary, but only these four will be considered in the rest of the paper.

2.3 Aggregate Cascade-Correlation Training

With all parts of the aggregate cascade-correlation network explained, it only remains to discuss the training of the network in more detail. Each time a new unit should be added to the hidden layer, a pool of units is created of the four types discussed in the previous subsection. Weights are initialized randomly. After that, all units in the pool are trained for a number of iterations, similar to backpropagation. This training is basically a gradient ascent, maximizing the correlation with the outputs as given in formula 5. The computation of the gradient depends of course on the type of unit. The gradient ascent itself is actually done by using resilient propagation, as described in [12]. This method has the advantage that the step size is determined automatically and convergence is faster than for a fixed step size. The basic idea is to increase the step size when the sign of the gradient remains the same, and decrease the step size when the sign changes.

When all units in the pool have been trained, the best one is chosen. In this case, the best unit is the one with the highest correlation. To be able to compare units of different types with each other, the absolute value of the real correlation has to be computed, and not the S-value from formula 5 in which some normalization constants were omitted. When the unit with the highest correlation has been chosen, it is installed in the network and the output weights have to be learned again. Because linear activation functions are used for the output units, the output weights can be determined with least squares linear regression.

In the ideal case, when there is enough data available, a validation set can be used to determine when to stop adding new units. If this is difficult, there is also an alternative stopping criterion. Typically, the first units added to the network will have a high correlation. When more units are added, the correlation will decrease until no more reduction can be made. One can stop training when the correlation is below a certain threshold or does not decrease significantly anymore.

3 Experiments

In this section, a number of experimental results will be discussed. First, a series of experiments is carried out on artificially created data sets. After that, the method is evaluated on some real-world data sets.

3.1 Simple Aggregates

A simple experiment to examine the capacity of the aggregate cascade-correlation network, is to create artificial data with predefined aggregate functions and train the networks on it. The data consists of bags with a variable number of elements. Each element of the bag is a vector with five components. Only the first or the first and second component are relevant for the target value, depending on the aggregate function under consideration. The values of these components are randomly generated, but in such a way that the target values are uniformly distributed over the possible target values. All the other components are filled with uniformly distributed random numbers from the interval $[-1, 1]$. It is very likely that he number of vectors in the bags influences the difficulty of the learning task, so different sizes are tested. The data sets denoted as small contain 5 to 10 vectors per bag, the medium data sets 20 to 30 and the large ones 50 to 100. Each data set contains 3000 bags. A range of different aggregate functions are used to construct the data sets:

1. **count:** the target is the number of vectors in the bag.
2. **sum:** the target is the sum of all values of the first component of the bag vectors.
3. **max:** the target is the maximum value of the first component of the bag vectors.
4. **avg:** the target is the average value of the first component of the bag vectors.
5. **stddev:** the target is the standard deviation of the values of the first component of the bag vectors.

6. **cmpcount:** the target is the number of bag vectors for which the value of the first component is smaller than the value of the second component.
7. **corr:** the target is the correlation between the first two components of the bag vectors.
8. **even:** the target is one if the number of positive values for the first component is even, and zero if it is odd.
9. **distr:** the target is one if the values of the first component come from a Gaussian distribution, and zero if they are from a uniform distribution.
10. **select:** the target is one if at least one of the values of the first component lies in a given interval, and zero otherwise.
11. **conj:** the target is one if there is at least one vector in the bag for which the the first and the second component lie in a certain interval.
12. **disj:** the target is one if there is at least one vector in the bag for which the first or the second component lies in a certain interval.

The first 7 data sets have a numerical target, the others a nominal target. In case of a nominal target, the number of positive and negative examples are equal. The experiments are done using 10-fold cross-validation. One fold is used as test set, 7 folds are used for training and 2 folds are used as validation set to determine when to stop adding units. The maximum number of hidden units is limited to 10. The number of candidate units trained in every step is 20, which means that there are five units of every type. Each unit is trained for 500 iterations, which should be more than enough to have converged to optimal weights. For the data sets with nominal target, the accuracy is reported and for the sets with numerical targets the mean squared error is given. Standard deviation is reported as well. The results are summarized in table 1.

From the results, it is clear that most functions can be learned very well. Only the *even* function is really impossible to learn it seems. For the *distr* function, the number of vectors must be large to be able to learn it well. This makes sense because it is easier to say whether a bag of values comes from a normal or uniform distribution if the bag is larger than when it is rather small. In table 2, results are given using fully recurrent networks, similar to what is used in relational neural networks [7]. Compared with these results, it is clear that ACCNs perform better. One of the major problems with these recurrent networks, is the decreasing performance on larger bags. If we look at the results for the select data sets for instance, then the accuracy on the data set with small bags is still reasonable for the recurrent network, although the accuracy for the ACCNs is better. But for the data sets with larger bags, the accuracy goes down for the recurrent networks while it remains about the same for the ACCNs. Overall, it is clear that ACCNs are a better choice than the recurrent networks.

3.2 Trains

The trains data sets are also artificially created data sets containing a number of trains. Every train consists of a number of cars, carrying some load. Some of the trains are eastbound, the others are westbound. The direction of the trains

Table 1. Results for the simple aggregate data sets with aggregate cascade-correlation networks. Accuracies or mean squared errors are given, depending on whether the target is numeric or nominal, together with the standard deviations. The columns small, medium and large refer to the size of the bags in the data sets.

		small	medium	large
MSE	count	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	sum	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	max	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	avg	0.04 (0.03)	0.01 (0.01)	0.01 (0.01)
	stddev	0.02 (0.02)	0.01 (0.01)	0.00 (0.00)
	cmpcount	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	corr	0.04 (0.03)	0.02 (0.02)	0.01 (0.01)
accuracy	even	51.59 (0.05)	51.10 (0.04)	51.34 (0.10)
	distr	66.48 (0.05)	77.97 (0.07)	84.23 (0.09)
	select	99.21 (0.07)	99.13 (0.10)	98.83 (0.10)
	conj	100.00 (0.04)	100.00 (0.00)	99.89 (0.10)
	disj	99.13 (0.06)	98.57 (0.10)	96.54 (0.11)

Table 2. Results for the simple aggregate data sets with fully recurrent networks. Accuracies or mean squared errors are given, depending on whether the target is numeric or nominal, together with the standard deviations. The columns small, medium and large refer to the size of the bags in the data sets.

		small	medium	large
MSE	count	0.06 (0.10)	0.26 (0.01)	0.23 (0.01)
	sum	0.00 (0.00)	0.09 (0.10)	0.23 (0.01)
	max	0.02 (0.00)	0.01 (0.00)	0.01 (0.01)
	avg	0.02 (0.00)	0.01 (0.01)	0.01 (0.01)
	stddev	0.03 (0.00)	0.04 (0.00)	0.04 (0.01)
	cmpcount	0.02 (0.01)	0.01 (0.01)	0.11 (0.07)
	corr	0.07 (0.01)	0.08 (0.04)	0.04 (0.01)
accuracy	even	52.64 (0.01)	50.19 (0.02)	50.23 (0.01)
	distr	51.62 (0.05)	63.49 (0.02)	58.70 (0.02)
	select	90.55 (0.03)	63.47 (0.03)	50.42 (0.02)
	conj	94.43 (0.01)	68.09 (0.10)	49.11 (0.05)
	disj	72.08 (0.03)	51.24 (0.05)	50.32 (0.02)

is what has to be learned and this target concept is based on the properties of the cars of a train and their loads. The cars of the train constitute a bag for each train. A data generator [13] for this train problem was used to create 12 data sets with different properties. Sets 1 to 4 consist of short trains, having 2 to 6 cars. Data sets 5 to 8 are similar to sets 1 to 4, except that they contain longer trains. Each of these trains consists of 20 to 29 cars. The used concepts are the same as for sets 1 to 4, except that the numbers in the aggregations are adapted to the longer length of the trains. Data sets 9 to 12 contain noisy data.

This means that a number of samples have been mislabeled. The used concepts for the different data sets are as follows:

1. Trains having at least one circle load are eastbound, the others are westbound.
2. Trains having at least one circle or rectangle load and at least one car with peaked roof or 3 wheels are eastbound, the others are westbound.
3. Trains having more than 8 wheels in total are eastbound, the others are westbound.
4. Trains having more than 3 wheels in total and at least 2 rectangle loads and maximum 5 cars are eastbound, the others are westbound.
5. Same concept as for set 1.
6. Same concept as for set 2.
7. Trains having more than 53 wheels in total are eastbound, the others are westbound.
8. Trains having more than 45 wheels in total and at least 10 rectangle loads and maximum 27 cars are eastbound, the others are westbound.
9. Same concept as for set 1, but with 5% noise.
10. Same concept as for set 1, but with 15% noise.
11. Same concept as for set 3, but with 5% noise.
12. Same concept as for set 3, but with 15% noise.

The training setup is the same as for the simple aggregate data sets. The results for ACCNs and fully recurrent networks are given in table 3. It is clear that most concepts can be learned well with ACCNs. Most of the data sets without noise have an accuracy very close to 100%. Only for set 8, which has the most difficult concept, is it impossible to get close to perfect accuracy. For the data sets with noise, the accuracies are all close to 100% minus the percentage of noise, which means that the method is noise-resistant. Compared with the fully recurrent networks, the results for ACCNs are always better again. Sometimes the difference is quite spectacular. Also for these data sets, the size of the bags has an important influence on the accuracy for the recurrent networks.

3.3 Musk

Musk is a well-known multi-instance data set [14]. Each data instance stands for a molecule, represented by a bag of all its possible conformations. A conformation is described by 166 numerical features. The molecules have to be classified as musk or non-musk. The data set consists of two parts. The first part contains 92 molecules, the second part 102. In each bag, there are between 2 and 40 conformations for the first part, and between 1 and 1044 for the second part.

Experiments were carried out using 10-fold cross-validation. For the ACCNs, a pool of 20 neurons and 500 training iterations are used in every step. The value of the correlation is used as stopping criterion as described above. The results for the musk data sets can be found in table 4. The accuracies for ACCNs are not bad but not that excellent either compared with the other methods. Some

Table 3. Average accuracy and standard deviation using 10-fold cross-validation for the trains data sets. ACCNs stands for aggregate cascade-correlation networks and FRNs for fully recurrent networks.

	ACCNs	FRNs
1	99.8 (0.2)	99.7 (0.1)
2	100.0 (0.0)	97.1 (0.2)
3	100.0 (0.0)	99.9 (0.1)
4	98.7 (0.4)	92.7 (0.3)
5	99.7 (0.6)	92.5 (0.2)
6	99.4 (0.5)	73.9 (0.3)
7	99.3 (0.6)	51.3 (0.5)
8	84.6 (8.4)	71.0 (0.3)
9	95.8 (0.3)	93.6 (0.2)
10	85.7 (0.1)	81.7 (0.4)
11	95.6 (0.2)	93.0 (0.3)
12	85.3 (0.2)	82.0 (0.3)

Table 4. Accuracies and 95% confidence intervals for the musk data sets using 10-fold crossvalidation. Results for other methods than ACCN are obtained from [14].

	musk 1	musk 2
iterated discrim APR	92.4 [87.0-97.8]	89.2 [83.2-95.2]
GFS elim-kde APR	91.3 [85.5-97.1]	80.4 [72.7-88.1]
GFS elim-count APR	90.2 [84.2-96.3]	75.5 [67.1-83.8]
GFS all-positive APR	83.7 [76.2-91.2]	66.7 [57.5-75.8]
all-positive APR	80.4 [72.3-88.5]	72.6 [63.9-81.2]
backpropagation	75.0 [66.2-83.8]	67.7 [58.6-76.7]
C4.5 (pruned)	68.5 [40.9-61.3]	58.8 [49.3-68.4]
ACCN	85.3 [83.1-87.5]	75.5 [72.6-78.4]

of these methods were specifically designed for multi-instance problems while ACCNs are more general. This could make the multi-instance methods better suited for this task. On the other hand, ACCNs might also suffer from poor learning because of relatively few data or skewness in the data. The networks are all very small, in most cases with just one hidden unit. If one looks at the type of unit selected, then this is almost always a *max* or *smx* unit. This is the most logical choice in case of a multi-instance problem.

3.4 Thioredoxin-Fold Proteins

In this classification task, proteins have to be classified as belonging to the Thioredoxin-fold family or not [15]. It is difficult to do this based on the primary sequence of the proteins, for instance by using hidden Markov models, because there is a low conservation of the primary sequence in this family of proteins. One approach to deal with this problem has been to transform the data into bags of vectors. In [15], this transformation is done in three steps. First, the primary sequence motif, which is known to exist in all Thioredoxin-fold proteins, is identified. Around this motif, aligned subsequences are extracted. Finally, these windows are mapped to 8-dimensional numeric properties. For further details about this transformation, see [15]. The relevant transformation is referred to as motif-based alignment.

The result of this transformation is a data set containing 193 proteins, each described by a bag of 8-dimensional feature vectors. Of these 193 proteins, 25 are labeled positive and 168 negative. The bags contain 35 to 189 vectors. Two different experiments were carried out, the same as in [15]. In the first, simple setting, the data set is divided in three parts, using two of them for training and one for testing. The training setup for the ACCNs is the same as for the musk data sets. The results are given in table 5. One can see that the results are worse than for a multi-instance learner. Standard deviations are also quite large, especially for the true positive rate. The skewness of the data set is probably responsible for these problems.

Table 5. Results for 3-fold cross-validation on the Thioredoxin-fold proteins data set. True positive and true negative rates are given, together with the standard deviation over ten runs for ACCNs. MIL denotes a multi-instance learner and this result was reported in [15].

	TP	TN
MIL	0.74	0.88
ACCN	0.614 (0.147)	0.838 (0.071)

Results were also obtained for a second, more difficult experiment carried out in [15], in which 5 of the 25 positive proteins were removed, retaining only the 20 most dissimilar ones. The negative examples are divided in 8 folds, the positive examples in 20 folds of 1 example each. A jack-knife test is performed, taking one of the negative folds and one of the positive folds as test set, the other 19 positive folds and one of the negative folds as training set. This yields a total of 160 experiments. Results for this experiment are shown in table 6. The true positive rate is better now than for MIL, but the true negative rate is worse. Overall, the situation seems to be more or less the same as for the musk data sets where it was also not possible to achieve the same accuracy as multi-instance methods.

3.5 Financial

For this experiment, the financial data set of the discovery challenge organized at PKDD'99 and PKDD'00 is used [16]. The data set contains 8 different relations in total and the goal is to classify loans as good or bad. There are 234 loans, 203 of which are good and 31 bad. This means that the default accuracy is already 86.8%. Each loan is linked to an account and for every account there is a bag

Table 6. Results for jack-knife tests on the Thioredoxin-fold proteins data set. True positive and true negative rates are given, together with the standard deviation over all 160 experiments for ACCNs. MIL stands for a multi-instance learner and this result was reported in [15].

	TP	TN
MIL	0.75	0.75
ACCN	0.850 (0.035)	0.681 (0.028)

Table 7. Results on the financial data set using 10-fold cross-validation. Results for FORF-LA, DINUS-C, RELAGGS and PROGOL are obtained from [17] and [3].

	accuracy
FORF-LA	90.8 (1.7)
DINUS-C	85.1 (10.3)
RELAGGS	88.0 (6.5)
PROGOL	86.3 (7.1)
ACCN	87.2 (2.3)

of transactions. The number of transactions varies between 93 and 594. Only these bags will be used by the ACCNs to learn a classification of the loans, all the other information is ignored because we only work with a single bag for each data instance. It can be hoped however that these transactions contain the most important information to learn the target function. The experiments were done using 10-fold cross-validation. Results are summarized in table 7. The accuracy for the ACCNs is not significantly higher than the default accuracy, but neither is the accuracy of the other methods, except FORF-LA. As said before, these other methods also use more information than only the bags of transactions.

4 Conclusion

In this paper, a method for learning functions over bags with neural networks has been proposed. The method is based on cascade-correlation networks, extended with aggregation units that are able to process bags of vectors. This yields a general framework, extendable with new types of units if necessary.

Experimental results on artificial data sets are encouraging and indicate that this method is indeed capable of learning a wide variety of aggregate functions over bags. Compared with recurrent networks on these data sets, ACCNs perform significantly better, especially when the size of the bags is large. Experiments on real-world data sets show that the method performs still reasonably well in the multi-instance setting, although not as well as specific methods for this kind of problems. Some properties of these data sets might make it more difficult to train ACCNs on this data. The result for the financial data set is not that good, but similar to other methods like RELAGGS and FORF.

References

1. Blockeel, H., Bruynooghe, M.: Aggregation versus selection bias, and relational neural networks. In: Getoor, L., Jensen, D. (eds.) IJCAI 2003 Workshop on Learning Statistical Models from Relational Data, SRL 2003, Acapulco, Mexico (2003)
2. Knobbe, A., Siebes, A., Marseille, B.: Involving aggregate functions in multi-relational search. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) PKDD 2002. LNCS (LNAI), vol. 2431, pp. 287–298. Springer, Heidelberg (2002)
3. Krogel, M.A., Wrobel, S.: Transformation-based learning using multirelational aggregation. In: Rouveirol, C., Sebag, M. (eds.) ILP 2001. LNCS (LNAI), vol. 2157, pp. 142–155. Springer, Heidelberg (2001)
4. Vens, C., Ramon, J., Blockeel, H.: Refining aggregate conditions in relational learning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 383–394. Springer, Heidelberg (2006)
5. Vens, C., Van Assche, A., Blockeel, H., Dzeroski, S.: First order random forests with complex aggregates. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 323–340. Springer, Heidelberg (2004)
6. Van Assche, A., Vens, C., Blockeel, H., Dzeroski, S.: First order random forests: Learning relational classifiers with complex aggregates. Machine Learning 64(1-3), 149–182 (2006)
7. Uwents, W., Blockeel, H.: Classifying relational data with neural networks. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 384–396. Springer, Heidelberg (2005)
8. Uwents, W., Monfardini, G., Blokeel, H., Scarsello, F., Gori, M.: Two connectionists models for graph processing: An experimental comparison on relational data. In: MLG 2006, Proceedings on the International Workshop on Mining and Learning with Graphs, pp. 211–220 (2006)
9. Ramon, J., De Raedt, L.: Multi instance neural networks. In: Raedt, L.D., Kramer, S. (eds.) Proceedings of the ICML-2000 workshop on attribute-value and relational learning, pp. 53–60 (2000)
10. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems. Denver 1989, vol. 2, pp. 524–532. Morgan Kaufmann, San Mateo (1990)
11. Werbos, P.J.: Back propagation through time: What it does and how to do it. Proceedings of the IEEE 78, 1550–1560 (1990)
12. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Proc. of the IEEE Intl. Conf. on Neural Networks, San Francisco, CA, pp. 586–591 (1993)
13. Michie, D., Muggleton, S., Page, D., Srinivasan, A.: To the international computing community: A new east-west challenge. Technical report, Oxford University Computing Laboratory, Oxford, UK (1994)
14. Dietterich, T.G., Lathrop, R.H., Lozano-Perez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence 89(1-2), 31–71 (1997)
15. Wang, C., Scott, S.D., Zhang, J., Tao, Q., Fomenko, D.E., Gladyshev, V.N.: A study in modeling low-conservation protein superfamilies. Technical Report UNL-CSE-2004-0003, University of Nebraska (2004)
16. Berka, P.: Guide to the financial data set. In: Siebes, A., Berka, P. (eds.) The ECML/PKDD 2000 Discovery Challenge (2000)
17. Vens, C.: Complex aggregates in relational learning. PhD thesis, Department of Computer Science, KULeuven (2007)

Learning Block-Preserving Outerplanar Graph Patterns and Its Application to Data Mining

Hitoshi Yamasaki[1], Yosuke Sasaki[1], Takayoshi Shoudai[1],
Tomoyuki Uchida[2], and Yusuke Suzuki[2]

Department of Informatics, Kyushu University
Fukuoka 819-0395, Japan
{h-yama,yosuke.sasaki,shoudai}@i.kyushu-u.ac.jp Department of Intelligent
Systems, Hiroshima City University
Hiroshima 731-3194, Japan
{uchida,y-suzuki}@hiroshima-cu.ac.jp

Abstract. An outerplanar graph is a planar graph which can be embedded in the plane in such a way that all of vertices lie on the outer boundary. Many chemical compounds are known to be expressed by outerplanar graphs. We proposed a block preserving outerplanar graph pattern (bpo-graph pattern, for short) as a graph pattern common to a set of outerplanar graphs like a dataset of chemical compounds. In this paper, firstly we give a polynomial time algorithm for finding a minimally generalized bpo-graph pattern explaining a given set of outerplanar graphs. Secondly we give a pattern mining algorithm for enumerating all maximal frequent bpo-graph patterns in a set of outerplanar graphs. Finally, in order to show the performance of the pattern mining algorithm, we report experimental results on chemical datasets.

Keywords: pattern discovery, graph structured pattern, inductive inference, outerplanar graph, graph mining.

1 Introduction

Large amount of data having graph structures, called semi-structured data, such as map data, CAD, biomolecular, chemical molecules, the World Wide Web are stored in databases. For example, Web documents and almost chemical compounds in NCI dataset [4], which is one of popular graph mining datasets, are known to be expressed by ordered trees and outerplanar graphs, respectively. An outerplanar graph is a planar graph which can be embedded in the plane in such a way that all of vertices lie on the outer boundary. In Fig. 1, we give five graphs G, G_1, G_2, G_3, G_4 as examples of outerplanar graphs.

In the fields of data mining, many researchers have been developing graph mining techniques based on computational and algorithmic learning theory. Horváth et al. [3] proposed an efficient graph mining algorithm for enumerating all frequent subgraph patterns, called d-tenuous outerplanar graphs, in a given set of outerplanar graphs. In [8], we introduced a graph pattern, called a block preserving outerplanar graph pattern (bpo-graph pattern, for short), having an outerplanar graph structure and structural variables. A variable is a list of at most 2

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 330–347, 2008.

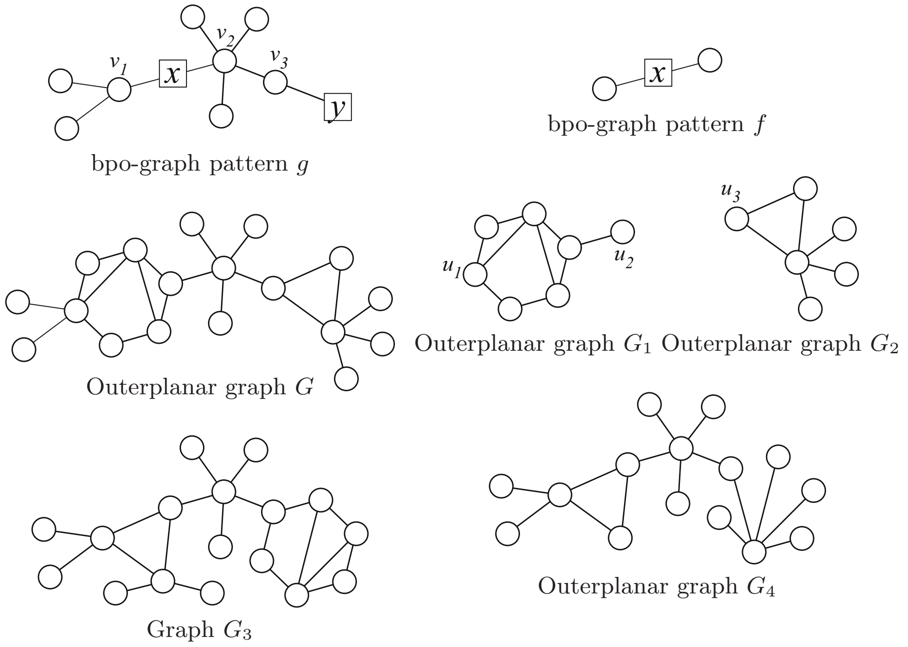


Fig. 1. Outerplanar graphs G, G_1, G_2, G_3, G_4 and bpo-graph patterns g, f. A variable is drawn by a box with lines to its elements. The label inside a box represents the variable label of the variable.

vertices and can be replaced with an arbitrary connected outerplanar graph. In Fig. 1, as examples of bpo-graph patterns, we give a bpo-graph pattern f having only one variable labeled with x and a bpo-graph pattern g having variables (v_1, v_2) and (v_3) labeled with x and y, respectively. Moreover, the outerplanar graph G is obtained from g by replacing variables (v_1, v_2) and (v_3) with outerplanar graphs G_1 and G_2, respectively. In [8], we proposed a polynomial time matching algorithm for deciding, given an outerplanar graph G and a bpo-graph pattern p, whether or not G is obtained from p by replacing all variables in p with certain connected outerplanar graphs, and by using the matching algorithm, we gave an Apriori-like algorithm for enumerating all frequent bpo-graph patterns in a given finite set of outerplanar graphs. Unfortunately, since the number of frequent graph patterns with respect to a given set of outerplanar graphs is huge in general, the practical time complexity of the graph mining algorithm tends to be quite high.

The purpose of this paper is to present an efficient algorithm for enumerating all meaningful frequent bpo-graph patterns from a given finite set of outerplanar graphs. A bpo-graph pattern p is said to be *minimally generalized* with respect to a given finite set S of outerplanar graphs if $S \subseteq L(p)$ and there is no bpo-graph pattern q such that $S \subseteq L(q) \subsetneq L(p)$, where for a bpo-graph pattern g, $L(g)$ denotes the set of all outerplanar graphs obtained from g by replacing all variables in g with arbitrary connected outerplanar graphs. Hence, it is natural that a minimally generalized bpo-graph pattern is more suitable for explaining a given set of outerplanar graphs. For example, since there exist bpo-graph patterns like g in Fig. 1 such that $\{G,\ G_3,\ G_4\} \subseteq L(g) \subsetneq L(f)$, the bpo-graph pattern f in Fig. 1 is not minimally generalized with respect to the set $\{G,\ G_3,\ G_4\}$, where $G,\ G_3,\ G_4$ are in Fig. 1. Moreover, since the bpo-graph pattern f matches all outerplanar graphs except outerplanar graphs consisting of only one vertex, we can see that f is meaningless and g is a more meaningful knowledge representation than f. Based on this idea, first of all, we consider a problem of finding a minimally generalized bpo-graph pattern explaining a given set of outerplanar graphs. Then, by presenting a polynomial time algorithm for solving this problem, we show that the class of bpo-graph pattern languages is polynomial time inductively inferable from positive data. Secondly, in order to achieve our purpose of this paper, we present an Apriori-like algorithm for enumerating all maximal frequent bpo-graph patterns for a given set of outerplanar graphs based on our first result in computational learning theory. Finally, in order to show the performance of the pattern mining algorithm, we report experimental results on subsets of NCI chemical dataset [4].

As related works, we proposed an interval graph pattern and presented a polynomial time algorithm for finding a minimally generalized interval graph pattern explaining a given finite set of interval graphs [14]. In the framework of inductive inference, we gave polynomial time learning algorithms for tree patterns with internal structured variables [11]. In [12], Takami et al. proposed a polynomial time learning algorithm for two-terminal series parallel graph patterns. In the framework of exact learning model, Okada et al. [7] showed polynomial time learnabilities of finite unions of graph structured datasets.

This paper is organized as follows. In Section 2, we define a bpo-graph pattern as an outerplanar graph having structured variables a minimally generalized bpo-graph pattern. In Section 3, we propose a polynomial time algorithm for finding one of minimally generalized bpo-graph patterns explaining a given set of outerplanar graphs. In Section 4, we propose a pattern mining algorithm for enumerating all maximal frequent bpo-graph patterns in a set of outerplanar graphs. In Section 5, lastly, we evaluate the performance of our pattern mining algorithm.

2 Graph Pattern

Let Λ and Δ be two alphabets. Each symbol in Λ and Δ is called a *vertex label* and an *edge label*, respectively. Let G be an undirected graph. In this paper, G is called a (Λ, Δ)*-labeled graph* if all vertices and edges in G are labeled with symbols in Λ and Δ, respectively. We denote by $V(G)$ the set of vertices in G and by $E(G)$ the set of edges in G. A *graph pattern* is defined as a graph-structured pattern with internal variables, which represents characteristic common structures in graph-structured data. In [13], we introduced a general graph-structured pattern, called a term graph, in order to design efficient algorithms for computational problems on graphs. We define a class of graph patterns as a special class of term graphs as follows. Let X be an infinite alphabet where $X \cap (\Lambda \cup \Delta) = \emptyset$, whose elements are called *variable labels*.

Definition 1 (Graph pattern). Let G be a (Λ, Δ)-labeled graph. A *variable* of G is a list of different vertices of $V(G)$, which is denoted by $(v_1, \dots, v_\ell)$ $(\ell \geq 1)$, where $v_i \neq v_j$ if $i \neq j$ $(1 \leq i, j \leq \ell)$. The vertices in a variable are called *ports* of the variable. Each variable is labeled with a variable label in X. If a variable has only one port, it is called a *terminal variable*, otherwise called an *internal variable.* A triplet $p = (V, E, H)$ is called a (Λ, Δ)*-labeled graph pattern* if (V, E) is a (Λ, Δ)-labeled graph and H is a set of variables of (V, E).

For a set or a list S, we denote by $|S|$ the number of members in S. Below (Λ, Δ)-labeled graphs and (Λ, Δ)-labeled graph patterns are simply called graphs and graph patterns, respectively, when the label sets are clear from the context. Let p be a graph pattern. We denote by $V(p)$, $E(p)$ and $H(p)$ the sets of all vertices, edges and variables of p, respectively. And we denote by $\lambda_p(v)$ the label of $v \in V(p)$ of p and by $\delta_p(e)$ the label of $e \in E(p)$ of p. The *degree* of v, denoted by $d_p(v)$, is the total sum of edges adjacent to v and internal variables including v, that is $d_p(v) = |\{\{u, v\} \mid \{u, v\} \in E(p)\} \cup \{h \mid h \in H(p),\ |h| \geq 2,\ h \text{ contains } v\}|$.

A *graph pattern* p is called a *linear* (*or regular*) *graph pattern* if all variables in $H(p)$ have mutually distinct variable labels in X. Let p and q be linear graph patterns. We say that p is *isomorphic* to q, denoted by $p \equiv q$, if there exists a bijection $\psi : V(p) \to V(q)$ such that (1) for any $v \in V(p)$, $\lambda_p(v) = \lambda_q(\psi(v))$, (2) $\{u, v\} \in E(p)$ if and only if $\{\psi(u), \psi(v)\} \in E(q)$, (3) for any $\{u, v\} \in E(p)$, $\delta_p(\{u, v\}) = \delta_q(\{\psi(u), \psi(v)\})$, and (4) for $\ell \geq 1$, $(v_1, v_2, \dots, v_\ell) \in H(p)$ if and only if $(\psi(v_1), \psi(v_2), \dots, \psi(v_\ell)) \in H(q)$.

Assumption. All graph patterns in this paper are linear. Then we call linear graph patterns graph patterns simply.

Definition 2 (Binding). Let p and q be graph patterns and x a variable label in X. Let $\sigma = (u_1, \dots, u_k)$ be a list of k distinct vertices in q. The form $x := [q, \sigma]$ is called a *binding* for x. We can apply a binding $x := [q, \sigma]$ to a variable $h = (v_1, \dots, v_\ell)$ in p which is labeled with x if the binding $x := [q, \sigma]$ satisfies that (1) $\ell = k$ and (2) $\lambda_q(u_i) = \lambda_p(v_i)$ for all i $(1 \leq i \leq \ell = k)$. A new graph pattern $p\{x := [q, \sigma]\}$ is obtained by applying the binding $x := [q, \sigma]$ to p in the following way. Let $h = (v_1, \dots, v_\ell)$ be a variable in p with the variable label x. Let q' be one copy of q and $u'_1, \dots, u'_k$ the vertices of q' corresponding to $u_1, \dots, u_k$ of q, respectively. For the variable $h = (v_1, \dots, v_\ell)$, we attach q' to p by removing the variable h from H_p and by identifying the vertices $v_1, \dots, v_\ell$ with the vertices $u'_1, \dots, u'_k$ of q', respectively.

A *substitution* θ is a finite collection of bindings $\{x_1 := [q_1, \sigma_1], \cdots, x_m := [q_m, \sigma_m]\}$, where $x_1, \dots, x_m$ are mutually distinct variable labels in X. A graph pattern $p\theta$ is obtained by applying all the bindings $x_1 := [q_1, \sigma_1], \dots, x_m := [q_m, \sigma_m]$ on p simultaneously.

Below we regard all graphs as graph patterns without variables. We say that a graph pattern p *matches* a graph G if there exists a substitution θ such that

$p\theta \equiv G$. As an example of graph patterns, in Fig. 1, we give a graph pattern g having variables (v_1, v_2) and (v_3) labeled with x and y, respectively. The graph pattern $g\{x := [G_1, (u_1, u_2)], y := [G_2, (u_3)]\}$ is isomorphic to the graph G in Fig. 1 where G_1 and G_2 are graphs in Fig. 1, and therefore g matches G. The matching problem for a graph pattern p and a graph G is to decide whether or not p matches G. In general, this problem is NP-complete. It remains NP-complete if G is a tree and p contains internal variables which have more than 3 ports [6]. If G is a tree and any variable in p has just 2 ports, the matching problem is computable in polynomial time [5].

Let G be a connected graph. G is said to be *biconnected* if for any two vertices in V, there exists a cycle which contains the two vertices. For a subset U of V, an *induced subgraph* by U, denoted by $G[U]$, is a subgraph $G[U] = (U, \{\{u, v\} \in E(V) \mid \text{both } u \text{ and } v \text{ are in } U\})$. An induced subgraph $G[U]$ is said to be a *block* if it is biconnected and there is no proper superset U' of U such that $G[U']$ is biconnected. An edge which does not belong to any block is called a *bridge*. For a vertex v of a connected graph G, v is called a *cutpoint* if $G[V(G) - \{v\}]$ is unconnected. An *outerplanar graph* is a planar graph which can be drawn in the plane in such a way that all vertices have a border with the outer face. We denote by $\mathcal{O}$ the set of all outerplanar graphs.

In order to make our discussion simpler, we assume that all outerplanar graphs are connected.

Definition 3 (Block preserving outerplanar graph patterns). A graph pattern p is a *block preserving outerplanar graph pattern*, *bpo-graph pattern* for short, if p satisfies the following three conditions.

1. Any internal variable has just 2 ports.
2. A graph $G_p = (V(p), E(p) \cup \{\{u, v\} \mid (u, v) \in H(p)\})$ is outerplanar.
3. Each port of any internal variable is either a cutpoint or a vertex of degree 1 in G_p.

Since all internal variables in a bpo-graph pattern are bridges in G_p, we call an internal variable in a bpo-graph pattern a *bridge variable*. For example, a graph pattern g in Fig. 1 is a bpo-graph pattern and a variable (v_1, v_2) of g is a bridge variable. We denote by $\mathcal{O}_\mathcal{P}$ the set of all bpo-graph patterns.

We have the following theorem.

Theorem 1 ([8]). *The matching problem for a bpo-graph pattern $p \in \mathcal{O}_\mathcal{P}$ and an outerplanar graph $G \in \mathcal{O}$ is computable in $O(nN^2\sqrt{d_{\max}})$ time, where n and N are the numbers of vertices of p and G, respectively, and $d_{\max}$ is the maximum degree of G.*

For a bpo-graph pattern $p \in \mathcal{O}_\mathcal{P}$, *the bpo-graph pattern language of p* is defined as $L(p) = \{G \in \mathcal{O} \mid \text{there exists a substitution } \theta \text{ such that } G \equiv p\theta\}$. Moreover the class of bpo-graph pattern languages is defined as $L_{\mathcal{O}_\mathcal{P}} = \{L(p) \mid p \in \mathcal{O}_\mathcal{P}\}$. Below we assume that there exists no terminal variable which has a common port with a bridge variable. This assumption is reasonable because if it exists,

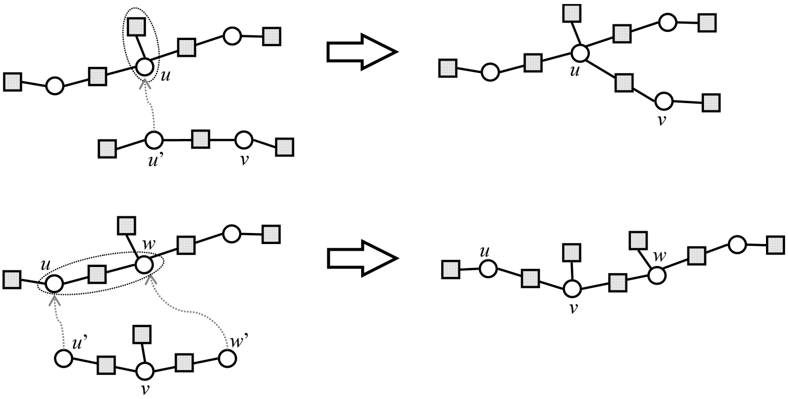


Fig. 2. Labeled vertex addition: The upper figure shows the case of a terminal variable and the lower shows the case of a bridge variable

any binding to it can be supplemented in a binding to the bridge variable. For a bpo-graph pattern $p \in \mathcal{O}_{\mathcal{P}}$ and a set of outerplanar graphs $S \subseteq \mathcal{O}$, p is said to be a *minimally generalized bpo-graph pattern explaining* S if $S \subseteq L(p)$ and there is not a bpo-graph pattern $q \in \mathcal{O}_{\mathcal{P}}$ such that $S \subseteq L(q) \subsetneq L(p)$.

3 A Polynomial Time Algorithm for Finding a Minimally Generalized BPO-Graph Pattern

Given a positive integer K and a set of outerplanar graphs S, the problem of deciding whether or not there is a minimally generalized bpo-graph pattern p such that the number of variables in p is at most K is NP-complete even if all outerplanar graphs in S are trees [10]. In this section, we give a polynomial time algorithm for finding one of minimally generalized bpo-graph patterns explaining a given set of outerplanar graphs. First of all, we define the following four procedures for a bpo-graph pattern p and a specified variable $h \in H(p)$, which are called *refinement operators*. Let Λ' and Δ' be finite subsets of Λ and Δ, respectively. And let Υ be the finite set of blocks.

1. **Λ'-Labeled Vertex Addition**
   Let v be a new vertex which has a vertex label in Λ', and u' and w' the other two new vertices. If h is a terminal variable (u), return $p\{x(h) := [q, (u')]\}$ where $q = (\{u', v\}, \emptyset, \{(u', v), (u'), (v)\})$. If h is a bridge variable (u, w), return $p\{x(h) := [q, (u', w')]\}$ where $q = (\{u', v, w'\}, \emptyset, \{(u', v), (v, w'), (v)\})$ (see Fig. 2).
2. **Υ-Block Replacement**
   Let B be a bpo-graph pattern $(V(B), E(B), H(B))$ where $(V(B), E(B))$ is a block in Υ and $H(B) = \{(v) \mid v \in V(B)\}$. Let u' and w' be distinct two vertices in $V(B)$. If h is a terminal variable (u) and $\lambda_B(u') = \lambda_p(u)$,

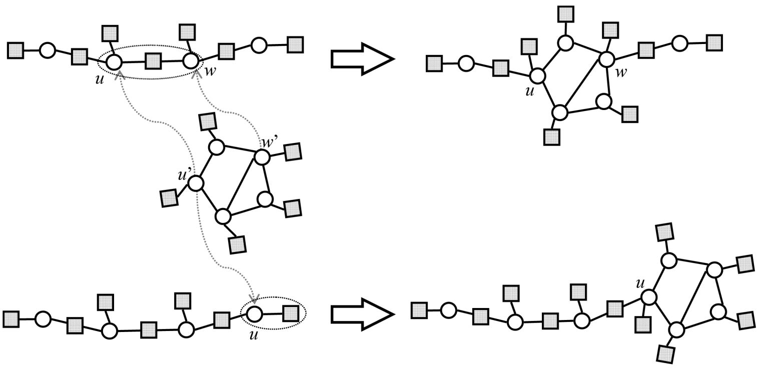


Fig. 3. Block replacement: The upper figure shows the case of a bridge variable and the lower shows the case of a terminal variable

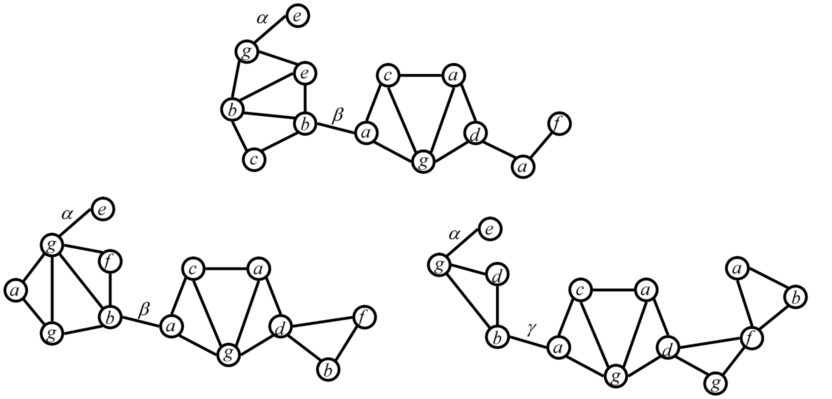


Fig. 4. An example set of outerplanar graphs: We assume that the edges with no label have a unique edge label except for α, β and γ

return $p\{x(h) := [B, (u')]\}$. If h is a bridge variable (u, w), $\lambda_B(u') = \lambda_p(u)$ and $\lambda_B(w') = \lambda_p(w)$, return $p\{x(h) := [B', (u', w')]\}$ where $B' = (V(B), E(B), H(B) - \{(u'), (v')\})$ (see Fig. 3).

3. **Δ'-Labeled Edge Replacement**
Let u' and w' be two new vertices and $\{u', w'\}$ a new edge which has an edge label in Δ'. If h is a bridge variable (u, w), return $p\{x(h) := [(\{u', w'\}, \{\{u', w'\}\}, \emptyset), (u', w')]\}$.
4. **Terminal Variable Deletion**
Let u' be a new vertex. If h is a terminal variable (u), return $p\{x(h) := [(\{u'\}, \emptyset, \emptyset), (u')]\}$.

Next we give an algorithm for finding a minimally generalized bpo-graph pattern by using the above four refinement operators.

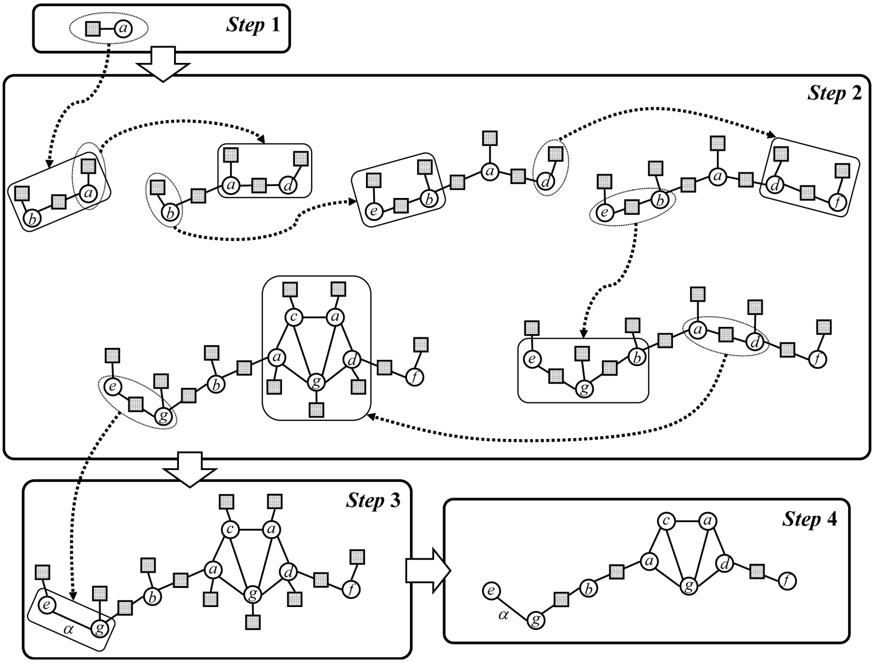


Fig. 5. An example process of finding a minimally generated bpo-graph pattern explaining an example set of outerplanar graphs in Fig. 4

Algorithm 1

Input: a nonempty set of outerplanar graphs $S \subseteq \mathcal{O}$.
Output: a minimally generalized bpo-graph pattern explaining S.

Step 0. Let Λ_S, Δ_S, and Υ_S be the sets of vertex labels, edge labels, and blocks, respectively, which appear in all outerplanar graphs in S.

Step 1. Let $p = (\{u\}, \emptyset, \{(u)\})$ where u is a new vertex whose label is in Λ_S. Let $H = \{(u)\}$.

Step 2. Let h be a variable in H. Let p' be a bpo-graph pattern which is obtained from p by applying either **Λ_S-Labeled Vertex Addition** or **Υ_S-Block Replacement** to h. If p' explains S, update p to p'. If h is a bridge variable, when an operator succeeds or all operators fail, delete h from H. If h is a terminal variable, only when all operators fail, delete h from H. If there are variables newly added to p by the above refinement operators, the variables are also added to H. Repeat this step until H becomes empty.

Step 3. For each bridge variable h in p, if a bpo-graph pattern p' obtained from p by applying **Δ_S-Labeled Edge Replacement** to h explains S, update p to p'.

Step 4. For each terminal variable h in p, if a bpo-graph pattern p' obtained from p by applying **Terminal Variable Deletion** to h explains S, update p to p'.

We have the following lemma.

Lemma 1. *If a set of edge labels is infinite, for a given nonempty set of outerplanar graphs S, Algorithm 1 outputs a minimally generalized bpo-graph pattern explaining S.*

Proof. Let p be a bpo-graph pattern computed by Algorithm 1 for an input S. Let p_2 and p_3 be temporary bpo-graph patterns of p immediately after Steps 2 and 3 respectively. Since no new vertex is added at Steps 3 and 4, $|V(p)| = |V(p_2)| = |V(p_3)|$ holds. We show that if there exists a bpo-graph pattern q such that $S \subseteq L(q) \subseteq L(p)$, $q \equiv p$ holds. We note that no bpo-graph pattern has a terminal variable connecting to a bridge variable with a common port.

Since $L(q) \subseteq L(p)$, $|V(q)| \geq |V(p_2)|$ holds. Let c be an edge label which does not belong to Δ_S. Let θ be a substitution for p which replaces all bridge variables with edges of label c and all terminal variable with single vertices. Since $q\theta \in L(p)$ and $L(p) \subseteq L(p_2)$, there is a substitution θ_2 for p_2 such that $q\theta \equiv p_2\theta_2$. All bindings in θ_2 which contain an edge of label c must be bindings for bridge variables in p_2. Let θ_2^x be a substitution which is obtained from θ_2 by replacing all edges appearing in θ_2 with bridge variables and removing all bindings for terminal variables in θ_2. Since $S \subseteq L(q) \subseteq L(p_2\theta_2^x) \subseteq L(p_2)$ and refinement operators Λ_S**-Labeled Vertex Addition** and Υ_S**-Block Replacement** can not apply to p_2 any more, all bpo-graph patterns in bindings in θ_2^x are bpo-graph patterns consisting of two vertices and one bridge variable connecting the two vertices. Therefore $|V(q)| = |V(p_2)|$Cand then we have $|V(q)| = |V(p)|$.

Since $L(q) \subseteq L(p) \subseteq L(p_3)$, there is a substitution θ_3 for p_3 such that $q\theta \equiv p_3\theta_3$. Let θ_3^x be a substitution which is obtained from θ_3 by replacing all edges of label c in θ_3 with bridge variables and removing all bindings for terminal variables in θ_3. We have $p_3 \equiv p_3\theta_3^x$, and then there is a bijection $\zeta : V(q) \to V(p_3)$ satisfying the following conditions.

1. For all $v \in V(q)$, $\lambda_q(v) = \lambda_{p_3}(\zeta(v))$.
2. If $(v, v') \in H(q)$ $(v \neq v')$ then $(\zeta(v), \zeta(v')) \in H(p_3)$ holds.
3. If $\{v, v'\} \in E(q)$ then either $\{\zeta(v), \zeta(v'))\} \in E(p_3)$ with $\delta_{p_3}(\{\zeta(v), \zeta(v')\}) = \delta_q(\{v, v'\})$, or $(\zeta(v), \zeta(v')) \in H(p_3)$.

If $\{v, v'\} \in E(q)$ and $(\zeta(v), \zeta(v')) \in H(p_3)$, a bpo-graph pattern obtained from p_3 by substituting an edge of label $\delta_q(\{v, v'\})$ for $(\zeta(v), \zeta(v'))$ also explains S. Therefore, since Δ_S**-Labeled Edge Replacement** can not apply to p_3 any more, if $\{v, v'\} \in E(q)$ then $\{\zeta(v), \zeta(v')\} \in E(p_3)$ and $\delta_{p_3}(\{\zeta(v), \zeta(v')\}) = \delta_q(\{v, v'\})$ holds.

Let d $(d \neq c)$ be an edge label which is not in Δ_S. Let θ' be a substitution for q such that it substitutes edges of label c for all bridge variables and edges of label d with a new vertex for all terminal variables. In $q\theta'$, there is no edge of label d which is adjacent to an edge of label c . Since $q\theta' \in L(p)$, there is a substitution θ_4 for p such that $q\theta' \equiv p\theta_4$. Since all terminal variables which are adjacent to bridge variables are removed by **Terminal Variable Deletion**, all bindings in θ_4 which contain edges of label d are applied to terminal variables.

Let θ_4^x be a substitution obtained from θ_4 so that all edges of label c in bindings in θ_4 are replaced with bridge variables, and all bindings to terminal variables are removed. Then $p \equiv p\theta_4^x$. Moreover since at Step 4, terminal variables are removed from p as much as possible while p explains S, we have $q \equiv p$. □

Lemma 2. *For a nonempty set of outerplanar graphs S, Algorithm 1 outputs a bpo-graph pattern in polynomial time with respect to the total sum of numbers of vertices and edges in S.*

Proof. It is easy to see that for any outerplanar graph G, $|E(G)| \leq 2|V(G)| - 3$. Then, for any bpo-graph pattern p, we have that $|E(p)| + |H(p)| \leq 3|V(p)| - 3$. Let $\mathcal{N}$ be the total sum of numbers of vertices of all outerplanar graphs in S. Let $N_{\min} = \min\{|V(G)| \mid G \in S\}$. The sets of labels Λ_S and Δ_S can be computed in $O(N_{\min}\mathcal{N})$ time. And by using a linear time algorithm for finding all blocks in a graph [1], Υ_S can be also computed in $O(N_{\min}\mathcal{N})$ time. Let $\Upsilon_S = \{B_1, \ldots, B_b\}$ $(b \geq 0)$. It is easy to see that $\sum_{i=1}^{b} |V(B_i)| = O(N_{\min})$. Since the number of vertices of the final output bpo-graph pattern is not more than $N_{\min}$, during Step 2, Λ_S**-Labeled Vertex Addition** and Υ_S**-Block Replacement** are applied to p at most $O(N_{\min}^2|\Lambda_S|) = O(N_{\min}^3)$ times and $O(N_{\min}^2 \sum_{i=1}^{b} |V(B_i)|^2) = O(N_{\min}^4)$ times, respectively. During Steps 3 and 4, Δ_S**-Labeled Edge Replacement** and **Terminal Variable Deletion** are applied to p at most $O(N_{\min}|\Delta_S|) = O(N_{\min}^2)$ times and $O(N_{\min})$ times, respectively. For each refinement operation, we have to decide whether or not a temporary bpo-graph pattern explains S, and from Theorem 1, it needs at most $O(N_{\min}\mathcal{N}^2\sqrt{d_{\max}})$ time where $d_{\max}$ is the maximum degree of outerplanar graphs in S. Then, the total computational time of Algorithm 1 is $O(N_{\min}^5\mathcal{N}^2\sqrt{d_{\max}})$. □

From Lemmas 1 and 2, we have the following theorem.

Theorem 2. *If a set of edge labels is infinite, the problem for finding a minimally generalized bpo-graph pattern explaining a given set of outerplanar graphs S is computable in polynomial time with respect to the size of S.*

From Theorems 1 and 2, we can conclude that $L_{\mathcal{O}_\mathcal{P}}$ is polynomial time inductively inferable from positive data by using the theorems in [2,9].

Theorem 3. *If a set of edge labels is infinite, $L_{\mathcal{O}_\mathcal{P}}$ is polynomial time inductively inferable from positive data.*

4 Pattern Enumeration Algorithm for Maximal Frequent BPO Graph Pattern Problem

Let S be a finite subset of $\mathcal{O}$ and p a bpo-graph pattern in $\mathcal{O}_\mathcal{P}$. Then, we denote by $O_S(p)$ the set of outerplanar graphs in S which are matched by p, called *occurrence set* of p with respect to S. The *frequency* of p with respect to S, denoted by $supp_S(p)$, is defined as $supp_S(p) = |O_S(p)|/|S|$. Let t be a real number where $0 < t \leq 1$. A bpo-graph pattern $p \in \mathcal{O}_\mathcal{P}$ is *t-frequent* with respect

```
Algorithm: GenPattern;
Input: a set of outerplaner graphs $S$ and a frequency threshold $t(0 < t \le 1)$;
Output: the set of all $t$-frequent bpo-graph patterns w.r.t. $S$;
begin
  Construct $L_0^t$ and $L_1^t$ w.r.t. $S$;
  Let $L_b$ be the set of all bpo-graph patterns in $L_1^t$ which have no bridge.
  $k := 2$;
  while $L_{k-1}^t \neq \emptyset$ do begin
    $L_k^t := \emptyset$;
    $M :=$ExtractGPattern$(L_{k-1}^t)$;
    $M'(\subseteq \{(p, O_S(p)) \mid p \in \mathcal{O}_\mathcal{P}\}) :=$GenPatternApriori$(M, S, t)$;
    foreach $(p, O_S(p)) \in M'$ do begin
      $C :=$GenRefinedPattern$(p, L_{k-1}^t, L_b)$;
      $L :=$CountFrequency$(C, O_S(p), t|S|)$;
      $L_k^t := L_k^t \cup L$
    end;
    $k := k + 1$
  end;
  return $\bigcup_{k \ge 0} L_k^t$
end.
```

Fig. 6. An algorithm for generating all frequent bpo-graph patterns with respect to a given set of outerplaner graphs

to S if $supp_S(p) \ge t$. We call this real number t a *frequency threshold*. A bpo-graph pattern $p \in \mathcal{O}_\mathcal{P}$ is a maximal t-frequent bpo-graph pattern with respect to S if p is t-frequent and there is no t-frequent bpo-graph pattern p' satisfying $L(p') \subsetneq L(p)$. First, we give an algorithm for computing the following problem.

Frequent BPO-Graph Pattern Problem
Input: A finite set of outerplanar graphs $S \subset \mathcal{O}$ and a frequency threshold t $(0 < t \le 1)$.
Output: The set of all t-frequent bpo-graph patterns in $\mathcal{O}_\mathcal{P}$ with respect to S.

For an integer $k \ge 0$, a *k-bpo-graph pattern* is defined to be a bpo-graph pattern such that the total sum of the numbers of blocks, bridge variables, and bridges is equal to k. Let S be a set of outerplanar graphs in $\mathcal{O}$ and t a real number where $0 < t \le 1$. Let L_k^t be the set of all t-frequent k-bpo-graph patterns with respect to S. The algorithm in this section is an improved version of the algorithm presented in [8], which generates L_{k+1}^t from L_k^t for any $k \ge 1$ in a breadth-first manner. Below we attach a terminal variable to each vertex of any generated bpo-graph pattern and do not remove the terminal variable from it, because we avoid to generate a huge number of patterns by the mining process. Then we assume that each vertex of any generated bpo-graph pattern has a terminal variable adjacent to it. We show the pattern enumeration algorithm GenPattern in Fig. 6.

First we describe the detail of the 1st line of the algorithm GenPattern, and then give explanations of procedures GenPatternApriori, GenRefinedPattern and CountFrequency at lines 4–14 of GenPattern.

Construct L_0^t and L_1^t w.r.t. S: L_0^t is the set of t-frequent 0-bpo-graph patterns consisting of a single vertex and a terminal variable connecting to the vertex. L_1^t is the set of t-frequent 1-bpo-graph patterns p of the following three types: (a) $p = (V(B), E(B), \{(u) \mid u \in V(B)\})$ where B is a block, and for two vertices u and v, (b) $p = (\{u, v\}, \{\{u, v\}\}, \{(u), (v)\})$ and (c) $p = (\{u, v\}, \emptyset, \{(u), (v), (u, v)\})$. L_0^t and L_1^t are computable in polynomial time with respect to the size of S.

For a set of bpo-graph patterns L and a bpo-graph pattern $p \in L$, we say that p is *a most generalized bpo-graph pattern* in L if there does not exist a bpo-graph pattern $q \in L$ such that $L(p) \subsetneq L(q)$. We have the following lemma.

Lemma 3. *A bpo-graph pattern $p \in L_k^t$ is the most generalized bpo-graph pattern in L_k^t if and only if there is no bridge in p and any block of p has at least* 3 *cutpoints.*

Proof. If there is a bridge, or a block in p which has at most 2 cutpoints, a bpo-graph pattern q obtained from p by replacing the bridge or block with a bridge variable is in L_k^t and satisfies $L(p) \subsetneq L(q)$. Conversely we suppose that q is the most generalized bpo-graph pattern in L_k^t such that $L(p) \subsetneq L(q)$. Let G be an outerplanar graph which is obtained from p by replacing all bridge variables with labeled edges, and all terminal variables with single vertices. Since $G \in L(q)$, there is a substitution θ such that $G \equiv q\theta$. Since both p and q are k-bpo-graph patterns, any binding in θ for a bridge variable is to replace it with a graph consisting of either one bridge or one block. Therefore G has a bridge or a block which has at most 2 cutpoints. By the construction of G, p also has a bridge or a block which has at most 2 cutpoints. □

From Lemma 3, any k-bpo-graph pattern which is a subgraph pattern of the most generalized bpo-graph pattern in L_{k+1}^t satisfies that (1) there is no bridge and (2) each block has at least 2 cutpoints. Procedure ExtractGPattern extracts from L_k^t the set of all bpo-graph patterns satisfying (1) and (2).

Procedure GenPatternApriori. This procedure computes the set of all pairs of most generalized bpo-graph pattern in L_{k+1}^t and its occurrence set with respect to S, from the set of frequent k-bpo-graph patterns. We use a similar method to the Apriori-like pattern enumeration algorithm in [8] in order to generate a superset of the set of the most generalized bpo-graph patterns in L_{k+1}^t and remove all bpo-graph patterns from the superset which has a block having at most 2 cutpoints. This set is computable in polynomial time w.r.t. the size of the output of ExtractGPattern, S and t. It is the same time complexity as that of the algorithm in [8].

For a k-bpo-graph pattern p ($k \geq 2$), we say that p_t is a terminal bpo-subgraph pattern if p_t is a 1-bpo-graph pattern and contains exactly one cutpoint of p. We denote by $p \ominus p_t$ the $(k-1)$-bpo-subgraph pattern obtained from p by removing all vertices of p except for the cutpoint of p_t.

```
Procedure GenRefinedPattern($p, L, L_b$);
Input: a bpo-graph pattern $p$ and two sets of bpo-graph patterns $L$ and $L_b$;
output: the set of bpo-graph patterns $p'$ such that $L(p') \subseteq L(p)$;
begin
$C := \{p\}$; $D_1 := \{p\}$; $D_2 := \emptyset$;
while $D_1 \neq \emptyset$ do begin
    foreach $p' \in D_1$ do begin
        $D :=$ReplaceVariable($p', L, L_b$); $D_2 := D_2 \cup D$
    end;
    $C := C \cup D_2$; $D_1 := D_2$; $D_2 := \emptyset$
end;
output $C$
end;
```

```
Procedure ReplaceVariable($p, L, L_b$);
Input: a bpo-graph pattern $p$ and sets of bpo-graph patterns $L$ and $L_b$;
begin
$C := \emptyset$;
foreach $((u_1, u_2), q) \in H(p) \times L_b$ do begin
    foreach $(v_1, v_2) \in V(q) \times V(q)$ do begin;
        if $v_1 \neq v_2 \wedge \lambda_p(u_1) = \lambda_q(v_1) \wedge \lambda_p(u_2) = \lambda_q(v_2)$ then begin
            $p' := p\{x := [q, (v_1, v_2)]\}$; //Let $x$ be a variable label of $(u_1, u_2)$.
            if $p' \ominus p_t \in L$ for all $p_t \in TB(p')$ then $C := C \cup \{p'\}$
        end
    end
end;
output $C$
end;
```

Fig. 7. Procedure GenRefinedPattern: We denote by $TB(p)$ the set of all terminal bpo-subgraph patterns of p

Procedure GenRefinedPattern. We describe this procedure in Fig. 7. GenRefinedPattern receives one of most generalized t-frequent k-bpo-graph pattern p and L^t_{k-1}, and returns a superset of the set of t-frequent k-bpo-graph patterns p' such that $L(p') \subseteq L(p)$. For a bpo-graph pattern p' which has ℓ bridge variables ($\ell \geq 1$) , ReplaceVariable proceeds to generate bpo-graph patterns p'' which have $\ell - 1$ bridge variables, in a similar way to refinement operators **Labeled Edge Replacement** and **Block Replacement** in Sec. 3. And then for the generated bpo-graph pattern p'', it decides whether or not $p'' \ominus p_t$ is in L^t_{k-1} for each terminal bpo-subgraph pattern p_t of p''. If $p'' \ominus p_t$ is not in L^t_{k-1}, p'' is not t-frequent. This is a useful heuristic procedure to make a candidate set for L^t_k smaller. It is easy to see the next lemma.

Lemma 4. *For the most generalized k-bpo-graph pattern p,* GenRefinedPattern *generates a set of k-bpo-graph patterns p' such that $L(p') \subseteq L(p)$ and $p' \ominus p_t$ is t-frequent for each terminal bpo-subgraph pattern p_t of p'.*

```
Procedure CountFrequency($C, U, \tau$);
Input: a set of bpo-graph patterns $C$, a set of outerplaner graphs $U$,
    and a real number $\tau$;
output: the set of bpo-graph patterns;
begin
1:  Let $C_i$ $(0 \leq i \leq k)$ be the set of all bpo-graph patterns in $C$ which have
    exactly $i$ bridge variables;
2:  $L := \emptyset$;
3:  foreach $p \in C_0$ do begin
4:      $O_U(p) :=$CountOccSet$(p, U)$;
5:      if $|O_U(p)| \geq \tau$ then $L := L \cup \{p\}$
6:  end;
7:  for $i := 1$ to $k$ do begin
8:      foreach $p \in C_i$ do begin
9:          $W := \emptyset$;
10:         foreach $p' \in C_{i-1}$ such that $L(p') \subseteq L(p)$ do begin
11:             $W := W \cup O_U(p')$
12:         end;
13:         $O_U(p) :=$CountOccSet$(p, U - W) \cup W$;
14:         if $|O_U(p)| \geq \tau$ then $L := L \cup \{p\}$
15:     end
16: end;
17: output $L$
end;
```

Fig. 8. An algorithm for counting frequencies of given bpo-graph patterns

procedure CountFrequency**.** We describe this procedure in Fig.8. This procedure exactly selects all t-frequent k-bpo-graph patterns from the output of GenRefinedPattern. Let p be the most generalized k-bpo-graph pattern of L_k^t and $U = O_S(p)$. Let C be the output set of GenRefinedPattern for an input p. Since any bpo-graph pattern p' satisfies $L(p') \subseteq L(p)$, the occurrence set of p' with respect to S can be computed from U. If there is a bpo-graph pattern $p'' \in C$ such that $L(p'') \subsetneq L(p')$ and the occurrence set W of p'' has already been computed, the occurrence set of p' can be computed only from $U - W$. In this way, we reduce the number of pattern matchings in this procedure.

First, CountFrequency computes all occurrence sets of bpo-graph patterns in C which have no variable, and decide their frequencies (lines 3–6). The procedure CountOccSet computes the occurrence set of a given bpo-graph pattern p with respect to a given set of outerplanar graphs U. Next, for all $i \geq 1$, this procedure computes the occurrence sets of all bpo-graph patterns in C which have i variables by using the occurrence sets of p' which have $i - 1$ variables (lines 7–16). Finally we have the following lemma and theorem.

Lemma 5. *For the most generalized k-bpo-graph pattern p,* CountFrequency *correctly outputs the set of all t-frequent k-bpo-graph patterns q such that*

$L(q) \subseteq L(p)$. The procedure CountFrequency *works in polynomial time with respect to its input size.*

Theorem 4. *For a given finite set of outerplanar graphs $S \subset \mathcal{O}$ and a frequency threshold t $(0 < t \leq 1)$, Algorithm* GenPattern *correctly computes the set of all t-frequent bpo-graph patterns in $\mathcal{O}_{\mathcal{P}}$ with respect to S. Moreover, for any $k \geq 2$, L_k^t is correctly computed from L_{k-1}^t in polynomial time with respect to the size of L_{k-1}^t by Algorithm* GenPattern.

Next, we give procedures for computing the following problem.

Maximal Frequent BPO-Graph Pattern Problem
Input: A finite set of outerplanar graphs $S \subset \mathcal{O}$ and a frequency threshold t $(0 < t \leq 1)$.
Output: The set of all maximal t-frequent bpo-graph patterns in $\mathcal{O}_{\mathcal{P}}$ with respect to S.

Let M_k^t be the set of all maximal t-frequent k-bpo-graph patterns with respect to S. For a t-frequent bpo-graph pattern p, we decide whether or not p is maximal t-frequent by using **Labeled Vertex Addition**, **Block Replacement** and **Labeled Edge Replacement** in Sec. 3. For each bridge variable h of p, if at least one bpo-graph pattern q which is obtained from p by applying either of the above three refinement operators to h is t-frequent, p is not maximal t-frequent. For more details, we computes M_k^t by applying the following three maximality tests on L_k^t and L_{k+1}^t generated by GenPattern.

Test1: For a most generalized t-frequent k-bpo-graph pattern p, let $R(p)$ be the set of k-bpo-graph patterns generated by ReplaceVariable. If $L_k^t \cap R(p) \neq \emptyset$, we conclude that p is not maximal t-frequent. Let Δ_0 and Υ_0 be the sets of all t-frequent edge labels and blocks appearing in S, respectively. This test corresponds to **Δ_0-Labeled Edge Replacement** and **Υ_0-Block Replacement** for each bridge variable of p. This maximality test can be done in the procedure CountFrequency.

Test2: For a k-bpo-graph pattern $p \in L_k^t$, if there exists a $(k+1)$-bpo-graph pattern $p' \in L_{k+1}^t$ which has p as its bpo-subgraph pattern, p is not maximal t-frequent. Let Λ_1 be the set of all vertex labels appearing in L_1^t. This test corresponds to **Λ_1-Labeled Vertex Replacement** for each terminal variable of p. In order to do this maximality test easily, we make the bidirectional links between p and all $(k+1)$-bpo-graph patterns p' which have p as its bpo-subgraph pattern. These links can be added when the if statement at line 6 of every ReplaceVariable application is executed.

Test3: Let Λ_2 be the set of all vertex labels of cutpoints of bpo-graph patterns in L_2^t consisting of exactly two bridge variables. For each bridge variable of p, if at least one of $(k+1)$-bpo-graph patterns p' obtained from p by applying **Λ_2-Labeled Vertex Addition** to h is in L_{k+1}^t, p is not maximal t-frequent.

The next theorem can be proved in a similar way to Lemma 1.

Theorem 5. *Let p be a t-frequent bpo-graph pattern with respect to a given set of outerplanar graphs S. If a set of edge labels is infinite, the above three maximality tests correctly decide whether or not p is maximal t-frequent with respect to S.*

	S_{100}, $t = 0.5$					S_{100}, $t = 0.3$				
k	C_k^t	L_k^t	M_k^t	time(sec)	time[8]	C_k^t	L_k^t	M_k^t	time(sec)	time[8]
0	10	4	0	0.03	0.03	10	4	0	0.03	0.03
1	53	10	0	0.1	0.1	53	15	0	0.1	0.1
2	32	23	0	0.5	1	72	39	5	1	1
3	155	96	3	5	8	253	146	2	6	8
4	320	280	8	4	13	626	555	6	5	18
5	757	713	4	8	42	1877	1798	16	14	72
6	1372	1332	15	11	102	5068	4996	21	28	276
7	1715	1696	15	14	211	12457	12390	39	55	1256
8	1374	1367	18	12	309	26352	26330	67	119	7431
9	606	606	34	6	222	46707	46677	27	241	49593
10	107	107	13	1	57	67940	67938	41	392	–
11	0	0	0	0.1	3	80945	80945	50	492	–
12	0	0	0	0	0	77220	77220	27	453	–
13	0	0	0	0	0	56164	56164	3	289	–
14	0	0	0	0	0	28744	28744	0	115	–
15	0	0	0	0	0	9112	9112	5	31	–
16	0	0	0	0	0	1360	1360	4	4	–
17	0	0	0	0	0	0	0	0	0.1	–

	S_{100}, $t = 0.1$ ($k \leq 11$)				
k	C_k^t	L_k^t	M_k^t	time(sec)	time[8]
0	10	5	0	0.03	0.03
1	63	24	0	0.1	0.1
2	150	99	8	1	1
3	503	296	17	6	9
4	1213	1041	8	6	20
5	4434	4128	17	18	93
6	15409	15010	19	47	524
7	49786	49379	60	118	4384
8	147859	147461	114	365	46995
9	401638	401296	248	1201	–
10	978292	978153	233	3814	–
11	2100705	2100696	–	11112	–

	$S_{10,000}$, $t = 0.3$			
k	C_k^t	L_k^t	M_k^t	time(sec)
0	37	4	0	0.3
1	499	16	0	3
2	79	51	9	173
3	266	149	6	742
4	498	415	17	1039
5	1038	961	35	1907
6	1962	1885	58	2980
7	3240	3137	156	4490
8	4379	4285	240	5530
9	4797	4723	342	5310
10	3554	3518	357	3692
11	1448	1424	172	1482
12	367	357	48	384
13	48	47	7	60
14	0	0	0	0

Fig. 9. Experimental results of Algorithm GenPattern for L_k^t (on S_{100} with $t = 0.5$, 0.3, 0.1 and on $S_{10,000}$ with $t = 0.3$) on Windows XP Professional SP2, JDK 1.5, Pentium D 2.80GHz, 2.00GB RAM

5 Experimental Result

We have implemented our maximal frequent bpo-graph pattern mining algorithm and tested on chemical datasets. In our experiments, we used two datasets S_{100} and $S_{10,000}$, both of which consist of outerplanar molecular graphs from the NCI database [4]. The datasets S_{100} and $S_{10,000}$ contain 100 and 10,000 outerplanar graphs, respectively. The results are given in Fig. 9. We experimented on S_{100} with respect to frequencies 0.5, 0.3 and 0.1, and on $S_{10,000}$ with respect to frequency 0.3. In the tables, C^t_k means the union of the sets of all k-bpo-graph patterns outputted by GenRefinedPattern during the k-th iteration of the while-loop at lines 4–14 in Algorithm GenPattern (see Fig. 6). We note that the frequencies of elements in C^t_k except for the most generalized k-bpo-patterns are not counted until CountFrequency starts. The average of $|C^t_k|/|L^t_k|$ is very close to 1. Therefore we succeed to reduce the runtime for computing the frequencies of candidate k-bpo-graph patterns. The column specified with M^t_k shows the numbers of bpo-graph patterns in M^t_k, which are found to be extremely smaller than $|L^t_k|$ for any frequency threshold.

The 4th column (resp. 5th column) in the tables shows the runtime in seconds for the generation of L^t_k from L^t_{k-1} by the pattern enumeration algorithm proposed in this paper (resp. in [8]). We utilize only the most generalized $(k-1)$-bpo-graph patterns from L^t_{k-1} to generate candidates of t-frequent k-bpo-graph patterns. Therefore the proposed algorithm becomes faster than the algorithm in [8]. For example, for S_{100} with frequency threshold 0.3, the total runtime of the algorithm in [8] for the generation of $L^{0.3}_9$ from $L^{0.3}_0$ is 58,655 sec. On the other hand, the total runtime of the algorithm proposed in this paper is 467 sec, which is reduced to about 1/125 of that of the algorithm in [8].

6 Conclusion and Future Works

We have considered a data mining problem of extracting structural features from semi-structured data whose data can be expressed by outerplanar graphs. First, we presented a polynomial time algorithm to find one of minimally generalized bpo-graph patterns from a given finite set of outerplanar graphs. By using this algorithm, we showed that the class of bpo-graph pattern languages is polynomial time inductively inferable from positive data. Second, we presented an Apriori-like algorithm for enumerating all maximal frequent bpo-graph patterns with respect to a given finite set of outerplanar graphs. Finally, we evaluated the performance of our graph mining algorithm by experiments on chemical datasets.

We are now studying the polynomial time learnabilities of more general classes than bpo-graph pattern languages. As future works, toward efficient graph mining systems for real-world databases, we will consider graph pattern mining problems on several other classes of graphs like planar graphs, two-terminal series parallel graphs, and so on.

References

1. Aho, A.V., Hopcroft, J.D., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. 21, 46–62 (1980)
3. Horváth, T., Roman, J., Wrobel, S.: Frequent subgraph mining in outerplanar graphs. In: Proc. KDD 2006, pp. 197–206 (2006)
4. National Cancer Institute. Chemical dataset, `http://cactus.nci.nih.gov/`
5. Miyahara, T., Shoudai, T., Uchida, T., Kuboyama, T., Takahashi, K., Ueda, H.: Discovering new knowledge from graph data using inductive logic programming. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 222–233. Springer, Heidelberg (1999)
6. Miyahara, T., Shoudai, T., Uchida, T., Takahashi, K., Ueda, H.: Polynomial time matching algorithms for tree-like structured patterns in knowledge discovery. In: Terano, T., Chen, A.L.P. (eds.) PAKDD 2000. LNCS, vol. 1805, pp. 5–16. Springer, Heidelberg (2000)
7. Okada, R., Matsumoto, S., Uchida, T., Suzuki, Y., Shoudai, T.: Exact learning of finite unions of graph patterns from queries. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 298–312. Springer, Heidelberg (2007)
8. Sasaki, Y., Yamasaki, H., Shoudai, T., Uchida, T.: Mining of frequent block preserving outerplanar graph structured patterns. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 239–253. Springer, Heidelberg (2008)
9. Shinohara, T.: Polynomial time inference of extended regular pattern languages. In: Goto, E., Nakajima, R., Yonezawa, A., Nakata, I., Furukawa, K. (eds.) RIMS 1982. LNCS, vol. 147, pp. 115–127. Springer, Heidelberg (1983)
10. Shoudai, T., Uchida, T., Miyahara, T.: Polynomial time algorithms for finding unordered tree patterns with internal variables. In: Freivalds, R. (ed.) FCT 2001. LNCS, vol. 2138, pp. 335–346. Springer, Heidelberg (2001)
11. Suzuki, Y., Shoudai, T., Uchida, T., Miyahara, T.: Ordered term tree languages which are polynomial time inductively inferable from positive data. Theor. Comput. Sci. 350, 63–90 (2006)
12. Takami, R., Suzuki, Y., Uchida, T., Shoudai, T., Nakamura, Y.: Polynomial time inductive inference of TTSP graph languages from positive data. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 366–383. Springer, Heidelberg (2005)
13. Uchida, T., Shoudai, T., Miyano, S.: Parallel algorithm for refutation tree problem on formal graph systems. IEICE Transactions on Information and Systems E78-D(2), 99–112 (1995)
14. Yamasaki, H., Shoudai, T.: A polynomial time algorithm for finding linear interval graph patterns. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 67–78. Springer, Heidelberg (2007)

Author Index

Alphonse, Erick 6
Appice, Annalisa 24

Bartlett, Mark 42
Bate, Iain 42
Bertolo, Stefano 5
Biba, Marenglen 59
Blockeel, Hendrik 315
Bratko, Ivan 77
Bryant, Christopher H. 176
Bui, Hung H. 192

Ceci, Michelangelo 24
Craven, Mark 4

d'Amato, Claudia 107, 210
Duboc, Ana Luísa 91

Esposito, Floriana 59, 107, 158, 210

Fanizzi, Nicola 107, 210
Ferilli, Stefano 59
Flach, Peter 226
Frasconi, Paolo 122

Inoue, Katsumi 261

Jaeger, Manfred 122
Joshi, Sachindra 140

Kazakov, Dimitar 42
Kersting, Kristian 2, 192
Könik, Tolga 279

Leban, Gregor 77
Lisi, Francesca A. 158

Malerba, Donato 24
Mamer, Thierry 176
McCall, John 176
Muggleton, Stephen 297

Natarajan, Sriraam 192

Osmani, Aomar 6

Paes, Aline 91
Passerini, Andrea 122
Price, Simon 226

Ramakrishnan, Ganesh 140

Saitta, Lorenza 244
Sakama, Chiaki 261
Sasaki, Yosuke 330
Shoudai, Takayoshi 330
Srinivasan, Ashwin 140
Stracuzzi, David J. 279
Suzuki, Yusuke 330

Tadepalli, Prasad 192
Tamaddoni-Nezhad, Alireza 297
Tenenbaum, Joshua B. 1

Uchida, Tomoyuki 330
Uwents, Werner 315

van Harmelen, Frank 3
Vrain, Christel 244

Wong, Weng-Keen 192

Yamasaki, Hitoshi 330

Žabkar, Jure 77
Zaverucha, Gerson 91